

École polytechnique de Louvain

Ultra-Low-Power Over-the-Air-Update in Secure LoRaWAN Networks

Author: **Tim HESS**
Supervisors: **David BOL, Ramin SADRE**
Readers: **Rémi DEKIMPE, Jean-Didier LEGAT**
Academic year 2019-2020
Master [120] in Electrical Engineering

Acknowledgement

I would like to thank all the people without whom this work would never have been what it is. First of all, I want to express my thankfulness to both my supervisors Prof. David Bol and Prof. Ramin Sadre for their support and willingness throughout the realisation of my Master thesis. Their help and rigour allowed me to keep a steady work pace despite the recent exceptional events. I moreover really appreciate their involvement to set me up an internship and give me the opportunity to start working on this thesis inside a company.

I also want to thank Rémi Dekimpe for his constant availability and help he provided during the whole past year. His advice allowed me to solve many problems. A special thanks to Martin Braquet too who worked in parallel on the same platform. Even if the exceptional situation at the end of the year did not allow us to thoroughly experiment both our work simultaneously, I still enjoyed working at your side.

Then I would like to thank the entire staff of CETIC and in particular, M. Damien Hubaux who welcomed me in his company for two months. Thank you Benjamin Bernaud for allowing me to work at your side on interesting projects and for assisting me during the rest of the year.

Finally, I would like to thank my family who supported me all along and who allowed me to take the path that would lead to these words.

Abstract

Firmware updates are essential during the lifetime of an End Device (ED) for reasons that go from adding new functionalities to fixing security vulnerabilities. In the Internet of Things (IoT) deployments, the emergence of Low-Power Wide-Area Network (LPWAN) has introduced a new family of EDs that operate at long range with strong energy constraints. However, enabling Firmware Update Over-the-Air (FUOTA) capabilities on one of the most promising LPWAN communication technology that is LoRaWAN is not a straightforward task due to its limitations. Moreover the idea of having to transmit and process the huge amount of data that represents a firmware image does not seem compatible with the ultra-low-power requirements of an ED that is expected to run for a decade. Although the LoRa Alliance provided mechanisms to enable efficient FUOTA on Long Range Wide-Area Network (LoRaWAN), true multicast FUOTA is not yet supported by ChirpStack, the most used LoRaWAN open-source server stack, which makes it unfit for high-scale deployments.

In this work, we propose a new firmware update server that runs along ChirpStack with the objective of handling differentiated FUOTA in a true multicast session and studying the required energy. Therefore, a dedicated test platform working on an energy harvesting technology is designed to validate the FUOTA process on an ultra-low-power real-life scenario. The chosen use-case is a Wireless Sensor Network (WSN) node for audio monitoring of forest ecosystems with the constraint of average power consumption of $10mW$.

The result of this development is that the FUOTA process of a differentiated firmware image of $15kB$ takes about 5 minutes and only needs 14 minutes to recover.

TABLE OF CONTENTS

List of Tables	viii
List of Figures	x
List of Abbreviations	xiii
Introduction	1
1 Fundamentals and state of the art	4
1.1 OTA programming	4
1.1.1 Motivations	5
1.1.2 Challenges	5
1.1.3 How it works	7
1.2 LoRaWAN for LPWAN	7
1.2.1 Why LoRaWAN ?	7
1.2.2 Architecture	9
1.2.3 Classes	10
1.2.4 Security	11
1.2.5 Figures of merit	12
1.3 OTA update for LPWAN	17
1.3.1 Mechanisms	19
1.3.2 FUOTA for LoRaWAN	20
1.3.3 State-of-the-art of FUOTA on LoRaWAN	25
2 Adapting a reference implementation for large deployments	29
2.1 A Mbed OS approach by ARMmbed	29
2.1.1 Cryptographic verification of the firmware	30
2.1.2 Memory usage	30
2.1.3 Issues and requirements	31
2.2 Design of a new firmware update server for central handling	32
2.2.1 Architecture	32

2.2.2	FUOTA process service	33
2.2.3	Using a RESTful API as an interface	35
3	Design of a test platform for an ultra-low-power application	37
3.1	Methodology, design choices and objectives	37
3.2	Board design	38
3.3	Validation	42
4	FUOTA Validation and characterisation of the designed test platform	45
4.1	FUOTA validation	45
4.2	Characterization	50
4.2.1	Setup time	50
4.2.2	Power and energy consumption analysis	51
4.3	Optimization - Multicast data rate	53
5	Discussion and Perspectives	54
	Conclusion	56

List of Tables

1.1	EU863-870 TX Data rate table [16]	15
1.2	FICORA frequency regulations [17]	15
1.3	Maximum throughput per LoRaWAN channel and ED [15]	16
1.4	EU863-870 default channels [16]	16
1.5	Semtech transceivers typical characteristics for LoRa modulation. Conditions as specified in the datasheet	17
1.6	FUOTA-enabled devices on LoRaWAN	27
1.7	Firmware update servers for LoRaWAN	28
4.1	FUOTA process power and energy summary	53
4.2	Energy consumption during the multicast Class-C session	53
5.1	Energy consumption during the multicast Class-B session	55

List of Figures

1	Expected adoption growth of IoT devices (2019) [3]	2
1.1	Required data rate vs. range capacity of radio communication technologies: LPWAN positioning [7]	8
1.2	Respective advantages of Sigfox, LoRa, and NB-IoT in terms of IoT factors [7]	8
1.3	LoRaWAN Network architecture with application server and network server, connected with base station and EDs [12]	10
1.4	Three device classes receive slot timing [13]	11
1.5	LoRaWAN security [8]	13
1.6	LoRaWAN packet and its protection [8]	13
1.7	Relationship between SF and airtime for LoRa modulation	14
1.8	Global Number of Connected IoT Devices [19]	18
1.9	Typical FUOTA Flow Diagram [20]	20
1.10	FUOTA architecture [28]	22
1.11	Clock synchronisation methods	23
1.12	Parity check matrix. $M = 26$, $N = 52$ [26]	24
1.13	FUOTA session using class C multicast [30]	26
2.1	Firmware signing strategy with asymmetric cryptography	31
2.2	FUOTA flash memory requirements	32
2.3	Device Profile example	34
2.4	Firmware Update Server API	36
3.1	Block diagram of the CMWX1ZZABZ module from Murata [32]	38
3.2	Block diagram of the WSN node	39
3.3	Schematic design of the WSN node	40
3.4	Board design of the WSN node	41
3.5	WSN node prototype	41
3.6	Sampled signal and FFT computed by the prototype	42
3.7	Power consumption when using a floating-point complex FFT	43
3.8	Power consumption when using a fixed-point complex FFT	44

3.9	Power consumption when sending a LoRaWAN frame	44
4.1	Differences in the two binary files	46
4.2	File size with different optimisation levels	46
4.3	Setup - Clock synchronisation	47
4.4	Setup - Multicast session	47
4.5	Setup - Fragmentation session	48
4.6	Setup - Class-C session	49
4.7	FUOTA processing	49
4.8	Initial phase - Time required [30]	50
4.9	Initial phase - Uplinks vs downlinks [30]	51
4.10	Power consumption during the multicast session	52
4.11	Power consumption during the processing job	52

List of Abbreviations

ED End Device.

IoT Internet of Things.

LPWAN Low-Power Wide-Area Network.

FUOTA Firmware Update Over-the-Air.

LoRaWAN Long Range Wide-Area Network.

WSN Wireless Sensor Network.

SF Spreading Factor.

OTA Over-the-Air.

TTN The Things Network.

CSS chirp spread spectrum.

OTAA Over-the-Air Activation.

AES Advanced Encryption Standard.

AppKey Application Key.

AES-CMAC Cipher-based Message Authentication Code using AES encryption.

AppSKey Application Session Key.

NwkSKey Network Session Key.

ABP Authentication By Personalisation.

CTR Counter Mode Encryption.

MIC Message Integrity Counter.

DR Data Rate.

ADR Adaptive Data Rate.

DR Wireless Personal Area Network.

DR Wireless Local Area Network.

FEC Forward Error Correction.

MAC Medium Access Control.

CR Coding Ratio.

UUID Universally Unique Identifier.

ECDSA Elliptic Curve Digital Signature Algorithm.

DSA Digital Signature Algorithm.

JWT JSON Web Token.

TCXO Temperature Compensated Crystal Oscillator.

Introduction

From a technical point of view, the IoT is a system of uniquely identified devices which have the ability to transfer data over a network in an autonomous way. However, understanding the concept of IoT should not only be limited to its technical aspect but must also focus on its societal ties and environmental implications. Nowadays, one could simply say that the IoT is connecting to the internet all the things in the world. This paradigm has brought a lot of changes in both society and global markets [1]. As the cost of these devices decreased, and to respond to an always increasing demand, the number of IoT devices has risen dramatically over the last decades and it is expected to rapidly continue increasing as shown in figure 1. This growing trend, however, raises a lot of concerns, in particular about their ecological footprint. Even if a lot of effort is put into minimising the pollution during the manufacturing process, it is also important to be able to grant those devices the longest lifetime possible. In a constantly changing world, it would, however, be utopian and even unacceptable to believe that you could install a device and just forget about it for the next decade. Indeed, it is impossible to write a software that is completely secure in the long run and in 2016 the world discovered with the Mirai botnet that even IoT is at risk [2]. A customer could also require new features for his device and thus a new software would need to be uploaded to the device. It would yet be tedious to perform this task if we face a huge number of devices spread over a big geographical area. The idea to be able to perform an Over-the-Air (OTA) update is appealing, yet this solution might not be suitable for a lot of devices. There may be strong energy limitations on the device itself (e.g. ED powered by solar cells) or limitations with its communication technology (e.g. limited uplink duty-cycle on LoRaWAN).

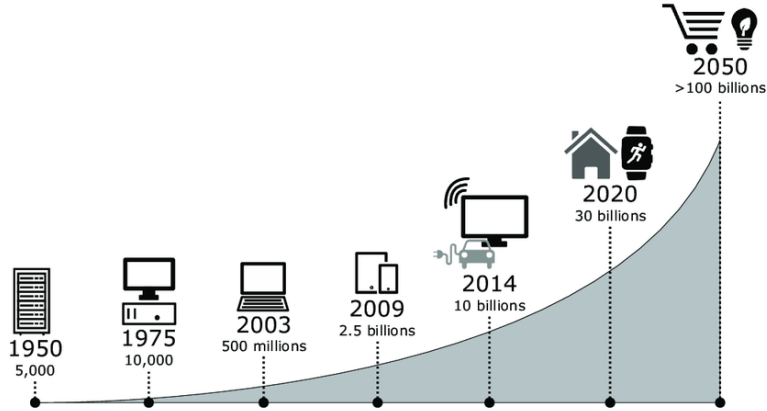


Figure 1: Expected adoption growth of IoT devices (2019) [3]

Those limitations lead to the following questions this work aims to answer :

- How can the concept of OTA update be transposed into LPWAN and LoRaWAN specifically?
- Is the concept of OTA update compatible with ultra-low-power requirements of an ED and what mechanisms can be used to decrease the energy consumption during the OTA update process?

Targets, contributions and outline

The goal of this thesis is first to understand the mechanism of performing an OTA update on LoRaWAN and secondly to identify challenges, gaps and possible solutions. The main challenge is the uplink-oriented communication of LoRaWAN which would result in an update process of several days for a firmware image of $50kB$. Although new specifications of LoRaWAN face this issue, the multicast capability is not yet implemented in open-source network servers. Based on this, a custom firmware update server is proposed as a network application, targeting the following figures of merit :

- scalability of EDs
- diversity of EDs
- ease of integration

Moreover, the main constraint added to this study is the limited average power consumption of $10mW$ to be compatible with an ultra-low-power energy-harvesting audio sensor for ecosystem monitoring [4]. A test platform is thus designed to characterise the energy budget of an OTA update. The main contributions from this work are:

- a complete state of the art of OTA update on LoRaWAN,
- a new firmware update server that handles multicast communication,
- a characterisation of an OTA update process that reveals its compatibility with ultra-low-power requirements of an ED.

These contributions are organised into five chapters:

- **Chapter 1** gives an overview of OTA programming and the motivations to choose LoRaWAN for our application. The specifications of LoRaWAN are then presented along with the challenges to perform an OTA update on this technology. Finally, a state-of-the-art of OTA update on LoRaWAN is presented.
- **Chapter 2** covers the reference implementation by ARMMbed and offers a solution to perform an OTA update on LoRaWAN with a huge deployment of devices.
- **Chapter 3** will focus on the prototype. It uses an STM32 microcontroller along with a LoRa radio and will allow us to monitor the energy consumption during an OTA update of each element.
- **Chapter 4** will focus on the validation of the FUOTA process itself along with its power consumption metrics.
- **Chapter 5** will discuss the perspectives for improvements, in particular regarding the future of LoRaWAN.

Chapter 1

Fundamentals and state of the art

In this chapter, we will present the principle of OTA firmware updates for embedded systems. We will first perform a general analysis of an OTA update to get a basic understanding of this process. Then, we will explain the motivations to choose the LoRaWAN technology for our example application along with its specifications. Finally, we will show how the mechanisms for an OTA update can be applied to an LPWAN and on LoRaWAN specifically.

1.1 OTA programming

OTA programming is a method to distribute a new code to an ED. Code examples could be a new firmware, new software, or new configuration settings. As stated in the acronym OTA, the update is performed wirelessly in contrast to, for example, plugging the device into a computer. Reasons for wanting to perform an update can be multiple, from removing bugs in an application to patching security breaches, or just to improve functionality. In this work, we will focus on firmware updates as it allows many more changes to the behaviour of an ED than just changing some configuration settings. Although there is officially no true compositional distinction between firmware and software, the firmware is commonly perceived at a lower level that does not require user interaction.

1.1.1 Motivations

The ability to perform an OTA update offers many benefits [5]. Firstly, performing a firmware update might require some specialised equipment like an ST-LINK programmer for STM32 devices. It would also be inappropriate to burden consumers with on-site cabled firmware updates at their home places, more than there might be a huge number of identical devices to be taken care of. OTA programming allows a quite transparent update without altering the user experience, provided that the update does not introduce new bugs.

Another scenario would be that the ED is located on remote sites that are difficult to access. As these devices probably send data to some telecommunication channel, it would be convenient to use the same communication technology to upload new firmware.

Devices can thus be updated faster, reducing the time that they have to work on a bugged firmware or a firmware having security issues. The user experience is improved and the risks for an attack are greatly reduced.

Last but not the least, the OTA update might offer the possibility of a multicast session, thus allowing to update all the devices at the same time, which reduces the bandwidth consumption as the new firmware image is only sent once. This feature would also be essential when it comes to major changes in infrastructure and devices required to update old firmware which prevents them from working properly.

1.1.2 Challenges

Sending a new firmware wirelessly, however, introduces some challenges. We will first focus on the challenges regarding the ED. It should be stressed that the device itself will be responsible for capturing the firmware file and updating its own flash memory. The target device must thus support this feature. As the device first needs to gather all the packets to rebuild the new firmware, some spare memory space should be available. In case the firmware is sent fully in an unaltered way, that space must have at least the size of the firmware file. A microcontroller like the STM32L072 however only has a RAM size of 20KB whereas the size of its flash memory is 192KB, which means that for a large firmware file, some external **memory** is needed. As a result, components must be added to the platform which increases energy consumption. To copy the new firmware into the flash

memory, we need to have a good understanding of the memory architecture of the microcontroller and its peripherals. The characteristics of the memory, for example, its base address or its page size must be known. It also usually requires to use some low-level instructions.

The second aspect to talk about is the communication medium. To ensure the **safety** of the ED, it is important that it has access to at least one working firmware at any time [6]. Reasons, why a new firmware might not be received properly and would lead to a corruption of the received firmware might be :

- Transmission errors like corrupted data packets
- Transmission failure due to loss of power or loss of connection
- Information loss due to lost packets

The third aspect deals with **security** issues. To qualify for being secure a system should enforce several features : [6] : privacy, integrity and authenticity.

- To enforce privacy, one has to ensure that the data cannot be read by unauthorised users. Encrypting the packets, however, raises the need to choose a decryption algorithm that would fit the characteristics of the ED.
- To enforce integrity, one has to ensure that the data was not altered by an attacker.
- To enforce authenticity, one has to ensure that the firmware is provided by a trusted source and not from a third-party that could provide a malicious firmware image.

Finally, the ED or communication technology could have some constraints. A sensor working on a solar cell can only use up a certain amount of energy before depleting its energy storage. Firmware update with a lot of communication can have high cost energy consumption and you would certainly not want the ED to turn off while it flashes its firmware. Some communication technology also has regulations on their bandwidth, thus it would be unfeasible to send a complete firmware image as it would take too much time. These challenges will be later addressed in section 1.3

1.1.3 How it works

As different devices may have different constraints like architecture and communication technology, it would be difficult to propose a unique solution to perform an OTA update. Based on the challenges enumerated in the previous section, a generic solution would include the following steps [5] :

1. Make sure the memory is able to perform an update.
2. The devices have to be prepared to accept an OTA update. The configuration of the ED might be changed to allocate more energy and resources to the update process.
3. The devices are ready and the update process has started.
4. The firmware is split in data packets and received by the devices. Some error detection and correction strategies might be used.
5. A bootloader analyses the firmware file. After validation and authentication, the new firmware file replaces the old one and the ED boots on to the new one. In case of some failure or error, no changes are made and the ED boots on its old firmware.

1.2 LoRaWAN for LPWAN

LPWAN has gained a lot of popularity in the choice of a communication network for IoT as they provide low power, long-range and low-cost characteristics [7]. They target usages where only small packets are sent periodically to a long-distance, which suits IoT application well, as shown in figure 1.1. Among LPWANs three candidates dominate the market : LoRaWAN, Sigfox and NB-Iot. Their respective characteristics are shown in figure 1.2.

1.2.1 Why LoRaWAN ?

The LoRaWAN technology is a popular choice amongst developers because it offers a protocol optimised for low-power applications and it allows to easily deploy its own network with millions of devices [8]. NB-IoT targets better latency performance with a high data rate at the expense of a higher power consumption. Moreover,

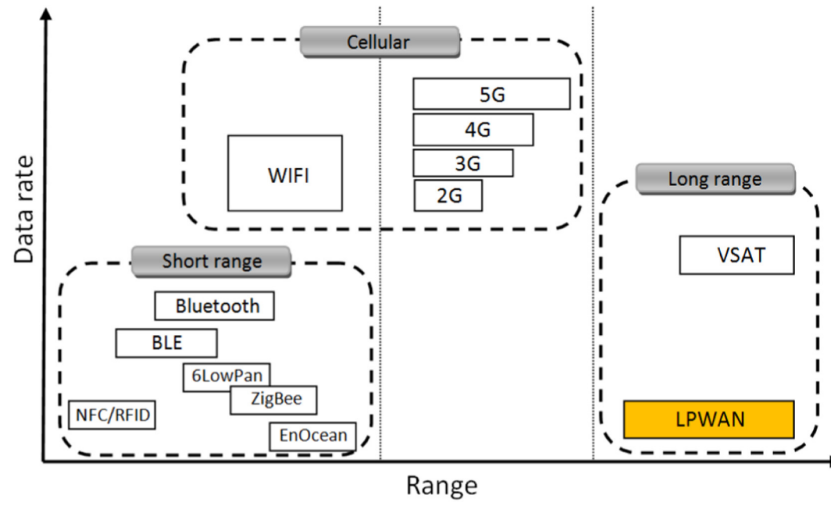


Figure 1.1: Required data rate vs. range capacity of radio communication technologies: LPWAN positioning [7]

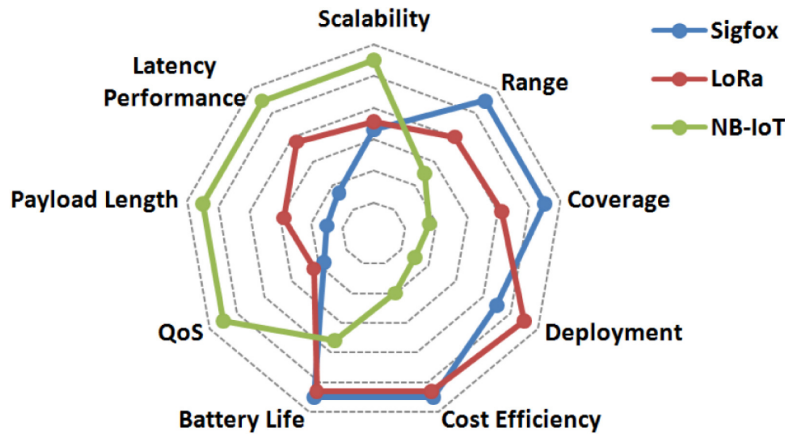


Figure 1.2: Respective advantages of Sigfox, LoRa, and NB-IoT in terms of IoT factors [7]

it doesn't allow implementing a private network. Sigfox on the other side targets the same objectives as LoRaWAN : applications with low power and low data rate. Sigfox however, does not also allow deploying its own private network. If you do not want to have to build a whole infrastructure to support a LoRaWAN network, many gateways are already available thanks to the platform The Things Network (TTN)¹ and new ones can easily be added to extend its coverage.

¹<https://www.thethingsnetwork.org/>

Whereas the communication protocol and the system architecture is defined by LoRaWAN, the physical layer is defined by LoRa. In Europe, LoRa uses the unlicensed 868 MHz band (EU863-870). LoRa was developed in 2009 by Cycleo, a French company and later acquired by Semtech, an American company. LoRa is based on a chirp spread spectrum (CSS) modulation which spreads a narrow-band signal over a wider channel bandwidth. If a device wants to include the LoRaWAN technology, it has to use a LoRa radio developed and sold by SemTech. They can be bought as a standalone package or integrated into a microcontroller. The standards of the LoRaWAN technology are defined by the LoRa Alliance².

The LoRaWAN Specification v1.0.2 released in July 2016 [9] is the most deployed version today as it is the most stable and the only one where devices can apply for the LoRaWAN Certification³. That's the reason why everything related to LoRaWAN will be based on this version unless specified otherwise. The v1.1 was released one year later in October 2017 [10]. Among all the changes, an interesting one was the addition of a new class, the Class-B (see section 1.2.3). It is however not recommended to use this version in a product as there are some discrepancies in the specifications and they won't be eligible for a LoRaWAN Certification. v1.0.3 was released in July 2018 [11] to allow developers to use the Class-B on the same specifications as v1.0.2. v1.0.4 should homogenise both v1.0.2 and v1.0.3 in a future release and would be the new recommended version. All the changes in v1.1 should be picked up in v1.1.1, but a lot of work has to be done yet.

1.2.2 Architecture

The topology of a LoRaWAN network is described as a star-of-stars topology. LoRaWAN gateways, also sometimes called LoRaWAN Base Station or LoRaWAN Concentrator, are responsible for relaying messages between the ED and the network server, as shown in figure 1.3. A particularity of a LoRaWAN network is the fact that ED is not assigned to a specific gateway, thus a message will be received by all the gateways that are in range. It is the job of the network server whom the received packets will be forwarded to deduplicate those before forwarding them to the corresponding application server. This feature allows localising ED thanks to very accurate time synchronisation between the different gateways [7]. For downlink communication, the network server can choose to which gateway to forward the packet based on the best quality information acquired during the last uplink.

²<https://lora-alliance.org/>

³<https://lora-alliance.org/lorawan-certification>

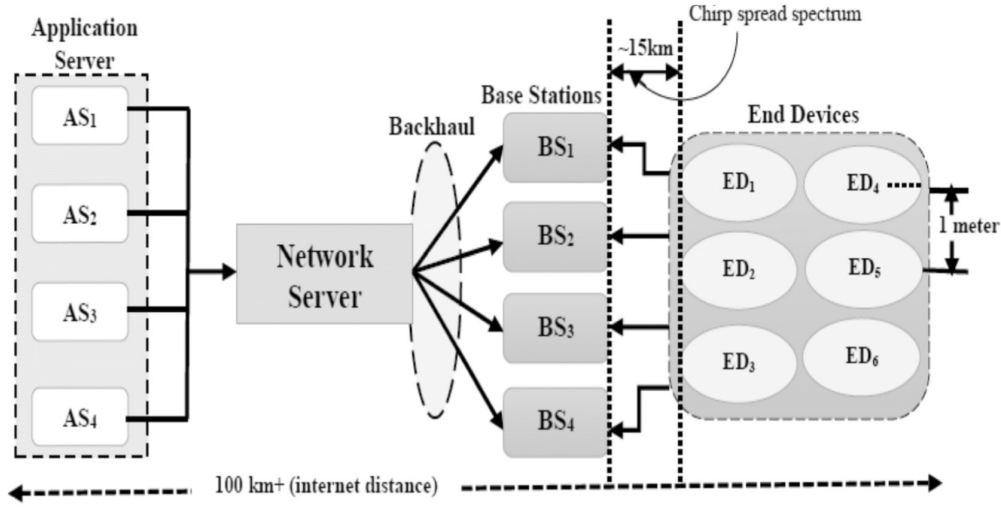


Figure 1.3: LoRaWAN Network architecture with application server and network server, connected with base station and EDs [12]

1.2.3 Classes

There are three different device classes that can be chosen from depending on which downlink communication latency one wants to achieve at the cost of a higher energy consumption. The three classes present a different downlink window pattern as pictured in figure 1.4.

The first class, the Class-A, offers an uplink oriented communication. Two downlink reception windows (Rx slot1 and Rx slot2) are opened with a delay of respectively one and two seconds after finishing an uplink transmission, allowing the radio to sleep for the rest of the time and thus offering the maximum battery lifetime. If a downlink message is received during the first downlink reception window, the second one does not open. This class is only suitable for applications that require short downlink communication, as a downlink message has to wait for an uplink message to occur.

The second class, the Class-B, offers scheduled receive slot to the ED beside the two regular windows after each uplink communication. To allow time synchronisation between the ED and the gateway, the gateway sends beacons at a regular time interval. The ED on its side opens a reception window, also called "ping slots" at scheduled times. This class is suited for applications where a certain predictability for downlink communication is needed at the cost of a higher energy consumption.

Mutual Authentication

When an ED wants to dynamically join a network, called the network join procedure or Over-the-Air Activation (OTAA), mutual authentication is performed between both parties. In this way, only authorised EDs can join an authentic network. Each LoRaWAN ED is personalised with a unique 128-bit Advanced Encryption Standard (AES) key, called the Application Key (AppKey), and a globally unique identifier, the EUI-64-based DevEUI. During the join procedure, both the ED and the network need to have knowledge of the AppKey. The ED first sends its Cipher-based Message Authentication Code using AES encryption (AES-CMAC) computed with the AppKey to the network server in a join request message. The network server then verifies the received AES-CMAC by matching it with his own computed AES-CMAC. After successful authentication, the network server derives two session keys from the AppKey : the Application Session Key (AppSKey), used by the application server for payload encryption and decryption, and the Network Session Key (NwkSKey), used by the network server to validate the integrity and authenticity of each message. The global LoRaWAN security scheme is pictured in figure 1.5. As the security of the ED depends on the AppKey and its derived session keys, they should be stored in a secure element to be protected against physical threats.

An alternative to the OTAA join procedure is to use Authentication By Personalisation (ABP). Here, the session keys are configured manually on the ED. It thus simplifies the join procedure as it doesn't need a handshake to exchange the keys anymore. The session keys must, however, be unique to prevent a widespread breach as a single device might be more easily compromised.

Data integrity and confidentiality protection

To ensure the full protection of the LoRaWAN traffic, both session keys are used as can be seen in figure 1.6. The payload is first encrypted by AES with Counter Mode Encryption (CTR). The packet carries a frame counter to protect against packet replay and a Message Integrity Counter (MIC) protects against packet tampering.

1.2.5 Figures of merit

The performance of the LoRaWAN technology can be characterised by several quantities. As we are mainly interested in IoT applications, we will thus focus on

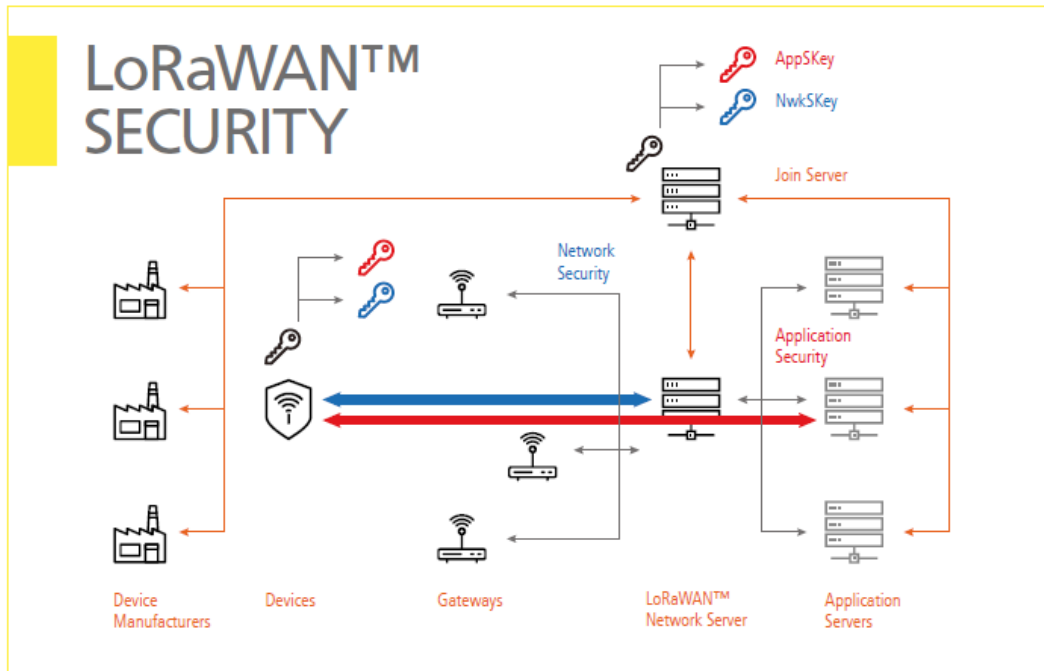


Figure 1.5: LoRaWAN security [8]

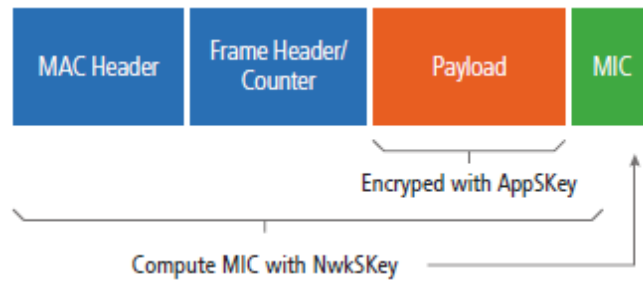


Figure 1.6: LoRaWAN packet and its protection [8]

the two following aspects : how much data can be sent during a period of time and how much energy does it cost. In this section, we will only focus on pure LoRaWAN specifications and metrics of LoRa transceivers. The result will be used in the next chapters.

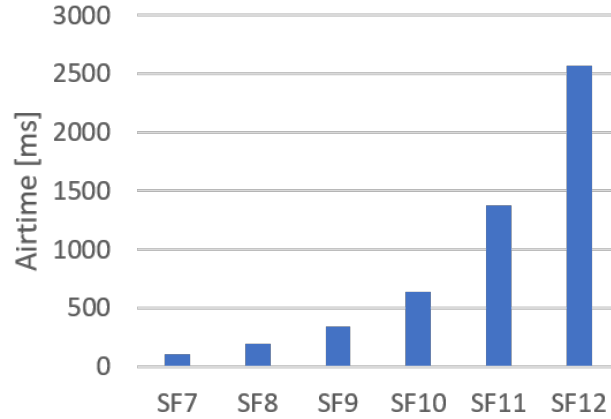


Figure 1.7: Relationship between SF and airtime for LoRa modulation

Maximum Throughput

The modulation bite rate R_b of LoRa can be expressed by equation 1.1, where SF is the Spreading Factor and BW the modulation bandwidth [14].

$$R_b = SF * \frac{1}{\left(\frac{2^{SF}}{BW}\right)} \text{bits/sec} \quad (1.1)$$

The Spreading Factor (SF) is a key characteristic in LoRaWAN. In LoRa modulation, six orthogonal SF ranging from 7 to 12 are featured, each of them defining how many chirps, who carries the data, are sent per second [15]. SF allows to improve spectral efficiency and increases the capacity of the network, as multiple EDs with different SF can send on the same frequency channel [14] without the communication being degraded. Choosing a lower SF increases the number of chirps sent per second, and thus the data rate. This implies that sending at lower SF decreases the airtime of a specific amount of data, which reduces the amount of time the radio has to be on and thus saves energy. Choosing a higher SF results however to a better sensitivity as the gateway has more time to sample the signal power. The coverage is thus increased. Increasing SF by one step roughly doubles the airtime as can be seen in figure 1.7. The airtime values were computed with the help of the Air time calculator provided by LoRaTools⁴ using the default values.

⁴<https://www.loratools.nl/#/airtime>

DR	Configuration	Maximum payload length [<i>byte</i>]
DR0	SF12/125kHz	51
DR1	SF11/125kHz	51
DR2	SF10/125kHz	51
DR3	SF9/125kHz	115
DR4	SF8/125kHz	222
DR5	SF7/125kHz	222
DR6	SF7/250kHz	222
DR7	FSK/125kHz	222

Table 1.1: EU863-870 TX Data rate table [16]

Frequency band [<i>MHz</i>]	Duty cycle [%]
863–868.6, 868.7–869.2, 869.4–869.65, 869.7–870	0.1
868.0–868.6	1
868.7–869.2	0.1
869.4–869.65	10
869.7–870.0	1
870.0–873.0	1

Table 1.2: FICORA frequency regulations [17]

LoRaWAN then defines different combinations of SF and bandwidths, called the Data Rate (DR). The DR ranges from 0 to 7 and offers different throughput and maximum payload size, as summarised in table 1.1. It is recommended for most of the EDs to turn the Adaptive Data Rate (ADR) functionality on, which enables the network to choose the appropriate DR based on the channel conditions between the ED and the gateway during the last transmission. This is however only true for EDs that are not continuously moving.

However, an important limiting factor for the performance of LoRaWAN are the restrictions imposed by section 7.2.3 of the ETSI EN300.220 standard as summarised in table 1.2. We can see that the different available frequency bands have a maximum duty cycle ranging from 0.1% to 10%. The frequency bands with a maximum duty cycle of 0.1% will not be mentioned in the future analysis as they are not relevant in this work. By taking the maximum physical and application layer throughput along with the frequency regulation restriction, we can thus obtain the maximum application throughput per ED as summarised in table 1.3.

DR	Max. application throughput per ED per channel [<i>bit/s</i>]	
	10% duty cycle	1% duty cycle
DR0	14.61	1.46
DR1	26.14	2.61
DR2	58.42	5.84
DR3	135.92	13.59
DR4	273.81	27.38
DR5	484.47	48.45
DR6	968.93	96.89
DR7	1851.6 ²	456.6

²-due to the need for opening RX windows after each frame, the maximum possible duty cycle is 4.1%

Table 1.3: Maximum throughput per LoRaWAN channel and ED [15]

Modulation	BW [kHz]	Channel Frequency [MHz]	DR	Duty cycle
LoRa	125	868.1	DR0 to DR5	1%
		868.3		
		868.5		

Table 1.4: EU863-870 default channels [16]

As a network operator can freely attribute the network channels up to a duty cycle of 1%, three default channels for uplink and downlink communication must be implemented in every ED to ensure compatibility with all the network gateways. Those channels are visible in table 1.4. In addition, one channel at $869.525MHz$ for downlink communication only must be supported. In fact, this channel is used as a fixed configuration for the second downlink window (Rx slot2 at DR0).

Power consumption

Currently, several LoRa transceiver chips are available from Semtech. Their typical main characteristics of LoRaWAN are summarised in table 1.5. The SX1272⁵ and SX1276⁶ transceivers are the oldest modules available on the market and can be found in most ED. The SX1276 claims to achieve a sensitivity of over $-148dBm$,

⁵<https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1272>

⁶<https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276>

	SX1272	SX1276	SX1261	SX1262
First release	06/2013	09/2013	10/2017	10/2017
Standby current [mA]	1.4	1.6	0.8	0.8
Receiver mode [mA] ($BW = 125kHz$)	9.7	10.3	DC-DC : 4.6 LDO : 8.8	DC-DC : 4.6 LDO : 8.8
Receiver sensitivity [dBm] ($SF = 12$, $BW = 125kHz$)	-137	-136	-137	-137
Maximum RF output power [dBm]	+20	+20	+22	+15

Table 1.5: Semtech transceivers typical characteristics for LoRa modulation. Conditions as specified in the datasheet

which is far better than the $-137dBm$ sensitivity of the SX1272. The maximum sensitivity of the SX1276 is however given for a bandwidth of $7.5kHz$, which gives it a better sensitivity only for pure LoRa communication. For LoRaWAN communication, the SX1272 might be a better choice as it consumes slightly less energy in standby and receiver mode. The current in transmitter mode is, however, the same. The newly released SX1261⁷ and SX1262⁸ feature a DC-DC buck converter which allows them to roughly divide the current in standby and receiver mode by two with a size four-time smaller compared to the SX127x. The difference between the SX1261 and SX1262 is their output power capability with respectively $+22dBm$ for the SX1261 and $+15dBm$ for the SX1262. As they are optimised for their maximum power output, the SX1262 consumes 50% more current than the SX1272 for the same output power [18].

1.3 OTA update for LPWAN

The motivations to perform an OTA update of embedded devices and the arising challenges have been discussed in section 1.1. If we take as an example a smartphone, there are nowadays no real difficulties to successfully update it. If the battery is empty, the smartphone just needs to be put on charge by the user, which is usually anyway done on a regular basis. There are usually no big constraints on the communication medium either since the firmware file does not put a big overhead on a WiFi communication. If we look at figure 1.8, we can see that the majority of IoT devices fall under the Wireless Personal Area Network (DR) and Wireless

⁷<https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1261>

⁸<https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1262>

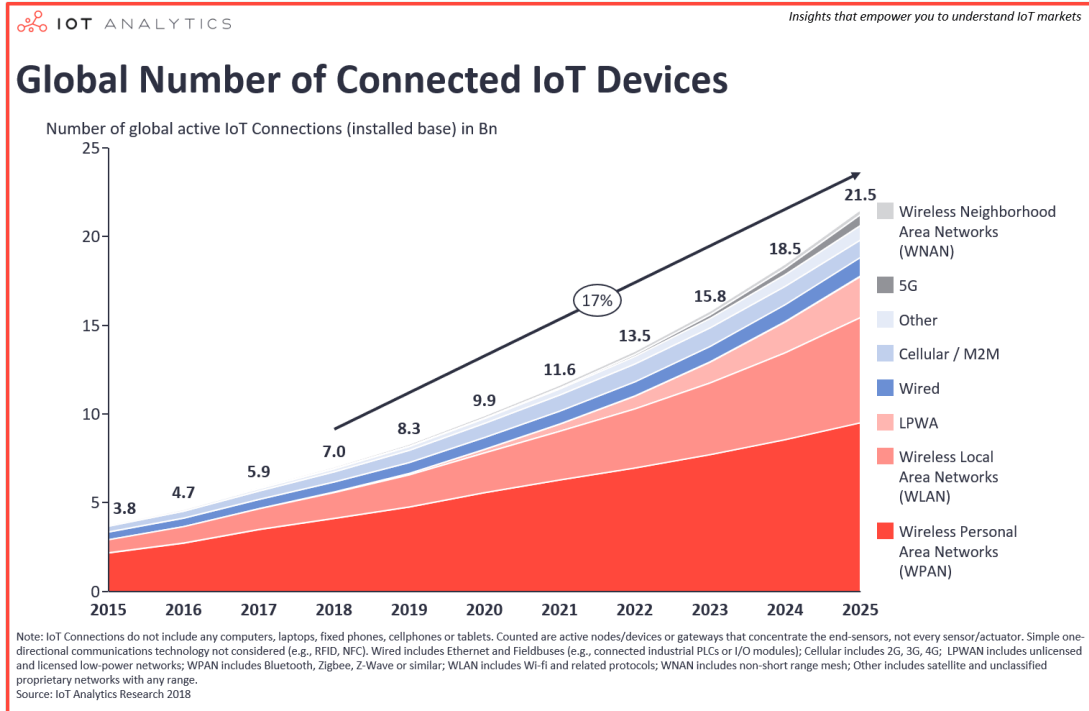


Figure 1.8: Global Number of Connected IoT Devices [19]

Local Area Network (DR) category. The most common communication technology found in those categories is Bluetooth and Wi-Fi as those devices are mostly used in domestic and industrial environments. Those devices can be considered similar to smartphones when analysing their OTA update capabilities. The devices are wired or often put on charge, especially devices with Wi-Fi technology which has a quite high energy consumption, and the communication technology does not have such limitations. The category of interest for us are the emerging LPWAN EDs. Their communication technology offers an extremely high battery life and a large communication range, which is particularly well suited for applications like WSN. When talking about OTA update on LPWAN, the process is commonly described as Firmware Update Over-the-Air (FUOTA). For this reason, this abbreviation will be used from now on.

The benefits of LPWAN, however, comes at a cost. The low data rate of these technologies makes high bandwidth applications such as FUOTA more difficult to implement. Moreover, WSN nodes have their own specific constraint : they are designed to have a small form factor and have limited available energy, computational power and energy [20]. Among those constraints, energy is the biggest one for a WSN [21]. In a simple WSN node, three entities consume energy :

the sensor, the microprocessor and the communication module. It is estimated that transmitting one bit wirelessly consumes the same energy as 1000 instructions [22], which is compelling when talking about the communication overhead of a FUOTA that would dramatically reduce the lifetime of a battery-powered ED. To be able to find the best strategy for conducting a FUOTA, 6 requirements are proposed [20]:

- The constraints of WSN/LPWAN must be maintained.
- The WSN nodes must be able to perform their task.
- The security of WSN nodes and the network must not be compromised.
- Energy consumption must be as low as possible.
- Packet losses or errors must be handled.
- The capability for FUOTA must be easy to integrate into the existing application.

1.3.1 Mechanisms

We will analyse an approach to perform a FUOTA on LPWAN. The typical interactions between all the EDs and services during an active FUOTA process are shown in figure 1.9. The update server is responsible for managing the whole FUOTA process. The traffic is routed from the update server to the nodes through the LPWAN system architecture. An architecture for a LoRaWAN network can be found in section 1.2.2.

The first task for the update server is to compute the difference between the new firmware and the one currently running on the node. This allows for reducing the amount of data that needs to be transmitted. To ensure this functionality, the update server needs to have access to the old firmware file. This can either be done manually or all the firmware files can be stored in some storage or database. The computed difference needs then to be encoded in a way that it can be easily decoded by a node. An optimised algorithm should try to minimise the size of the update file while requiring minimal computational time and memory on the node when decoding. As the update file is still too big to be sent all at once to the node, it needs to be fragmented into smaller packets.

In the next step, the node is notified and prepares itself for the upcoming update. Once ready, the update server starts to send the fragments to the node. To allow

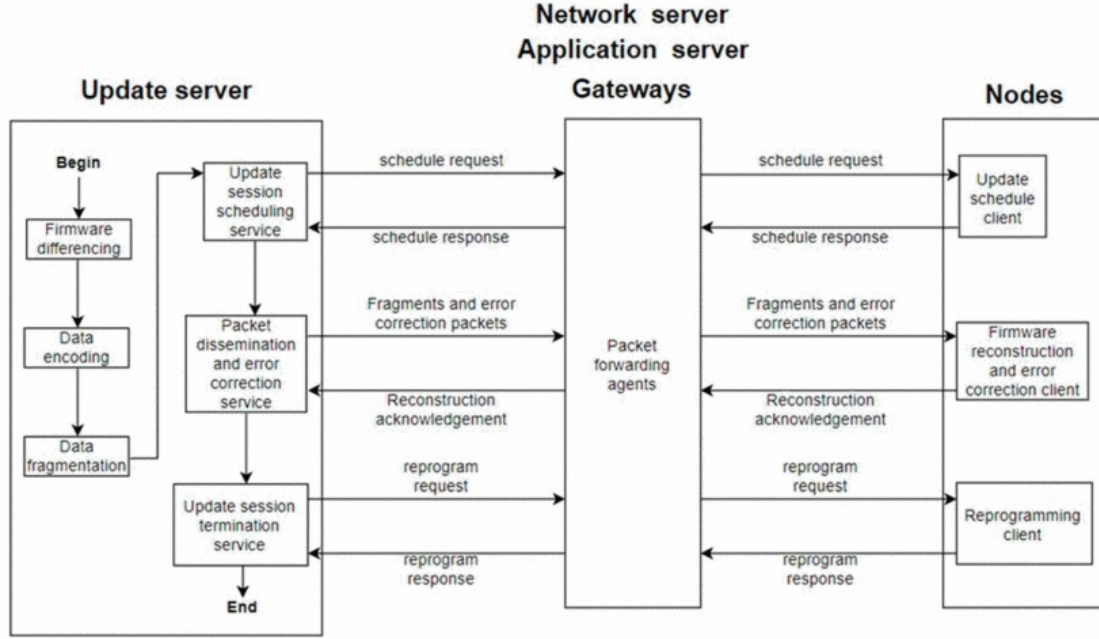


Figure 1.9: Typical FUOTA Flow Diagram [20]

the node to recover from packet losses, redundant packets should be sent using a Forward Error Correction (FEC) algorithm. When all the packets are received and verified for their authenticity and integrity, the node can start reconstructing the new firmware file. Eventually, the update session is terminated and the ED can reboot on the new firmware image. However, FUOTA is a risky process and can lead to device malfunction if not handled properly. A good practice would be to have the possibility to roll back to the previous firmware file should a problem arise. Review of past work in relation with FUOTA for LPWAN has shown that most of the suggested protocols do not put weight on security and go out of bounds of LPWAN constraints [20].

1.3.2 FUOTA for LoRaWAN

As LoRaWAN promises a long device lifetime, e.g. 10 years [23], it is important to be able to perform a FUOTA with this technology. In this section, we will address the specific challenges to perform a FUOTA on LoRaWAN and see how we can transpose the mechanisms seen in the previous section for this particular technology.

To understand the need for an optimised strategy, let us first imagine sending a firmware image of $50kB$ using the worst data rate DR0. As the maximum payload length for DR0 is 51 bytes, about 1004 downlink packets would be needed for the whole firmware. Taking into account the 1% duty cycle limitation, it would take over 3 days to upgrade only one ED. Accordingly, in a situation where hundreds of WSN nodes would have to be updated, the update process would take months, which is inconceivable especially if the update is critical and the ED cannot perform their task correctly until they get their new firmware image. Moreover, to be able to receive one downlink packet, an ED must first send an uplink packet. Knowing that one uplink communication consumes roughly five times more energy than a downlink communication, a lot of energy is also wasted, which does not fit the requirements of low-power EDs.

In June 2017, Jan Jongboom, former Principal Applications Engineer at ARM, and Johan Stokking, CTO and Co-Founder of The Things Industries, demonstrated how to properly allow FUOTA on LPWAN at the LoRa Alliance All Members Meeting and Open House in Philadelphia based on their proposal [24]. As both are active members of the LoRa Alliance, it is no wonder that their main focus was put on the LoRaWAN technology. The LoRa Alliance thus created the FUOTA working group which in 2018 published three new specifications for the application layer to enable efficient FUOTA over LoRaWAN which will be detailed later : multicast [25], fragmentation [26] and clock synchronisation [27]. To help to create an interoperable solution, they also released in 2019 a process summary with the technical recommendations for a FUOTA [28]. The recommended architecture is shown in figure 1.10 where only the interfaces with solid lines are described in the specifications. This architecture is similar to the one shown in figure 1.9 as the firmware update server acts as the brain of the process and communicates its commands with the application server. The network server on its side is responsible for managing the multicast groups and for broadcasting the firmware packets to a minimum set of the gateway to ensure all targeted EDs are reached.

Multicast [25]

The first new specification ensures minimum latency and minimum gateway utilisation. As seen previously, performing a FUOTA on Class-A is not efficient at all. A first step is to temporarily switch to Class-C or Class-B to allow a better throughput while reducing energy consumption as no uplink communication is required anymore. Another advantage is that downlink communications are less restricted as they can achieve a maximum duty cycle of 10%. This change would

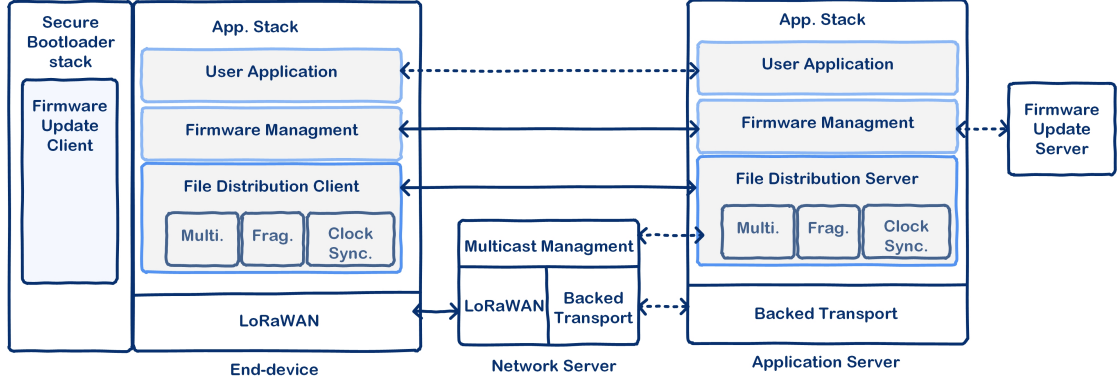


Figure 1.10: FUOTA architecture [28]

be temporary because after performing the update the ED would switch back to the lower energy Class-A. The new specification goes even further by enabling the creation of multicast groups. All the EDs would switch in receive mode at the same time and share the same session keys to be able to decode the same packets. A first command, *McGroupSetupReq*, is sent to all the EDs to share the encrypted temporary session keys. All the multicast messages are sent on port number 200 to distinguish them from the normal application. The ED acknowledges the first message with *McGroupSetupAns*. A second message, *McClassCSessionReq*, informs the EDs when to switch to Class-C. The session time is encoded in GPS epoch time (seconds since 6th January 1980) module 2^{32} . It also indicates when to revert back to Class-A in case of a timeout along with the data rate and channel to use. *McClassCSessionAns* acknowledges this second message. Similarly, a class-B session can also be requested with the *McClassBSessionReq* command.

Clock Synchronisation [27]

In order to minimise the time EDs have to stay in receive mode, it is important to have their clock synchronised. To deal with a minimal time drift, it is best to wait for a certain delay after the requested time to switch in Class-C before sending the firmware packets. The delay should, however, be kept as small as possible to reduce the waste of energy. The specifications, however, give some recommendations before using the new package. EDs which already have access to an accurate external clock like a GPS should use this clock source. EDs using Class-B should use the beacon time information. Finally, a more efficient method has been added to the new specification of LoRaWAN (v1.1 and above [10]) as time synchronization can directly be asked with the *DeviceTimeReq* Medium Access

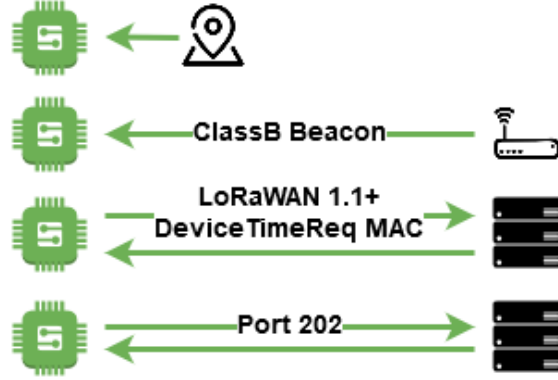


Figure 1.11: Clock synchronisation methods

Control (MAC) command. If those three conditions are not fulfilled, the ED can send the *AppTimeReq* command on port 202 to indicate its current device time. It will get as response *AppTimeAns* which contains the time delta correction in seconds. The accuracy of this approach is expected to be around one second. The four different methods for clock synchronisation are summarised in figure 1.11.

Fragmentation [26]

The last new specification allows big data blocks to be sent to EDs in a reliable way. Since firmware images can reach up to hundreds of kilobytes, they have to be split in small fragments. However, when an ED is in Class-C, it receives all the data packets in succession without being able to acknowledge them. If a packet is lost, the ED needs to have a way to rebuild the missing packets. A solution for this issue is also covered in the package.

To set up the fragmentation session, the ED receives the *FragSessionSetupReq* command which specifies the number of fragments, the fragment size, the recovery algorithm in case of packet losses and the padding size. The *FragSessionSetupAns* command is sent as a reply.

In the latest version of the specifications, only one algorithm for packet loss recovery is proposed. It works by adding additional FEC code at the end of the original firmware image. Let us consider a firmware image split into M data fragments of equal arbitrary size. The uncoded fragments are noted B_i and the full firmware image is concatenated as $[B_1, B_2, \dots, B_m]$. The N coded fragments $P_M^N(n)$ are constructed by performing a bitwise Xor operation between the uncoded fragments B_i and a parity check matrix $C_M^N(n, i)$ as shown in equation 1.2. $N \geq M$

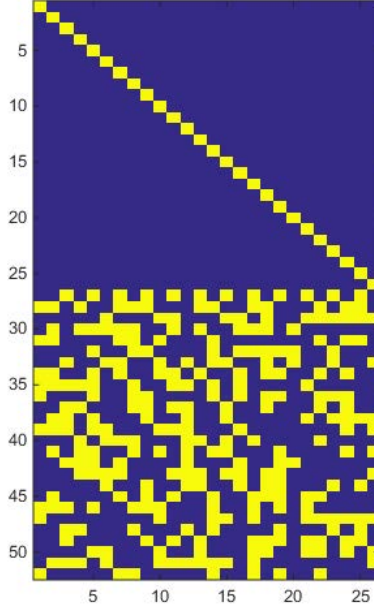


Figure 1.12: Parity check matrix. $M = 26$, $N = 52$ [26]

and $\frac{M}{N}$ is called the Coding Ratio (CR). The aim is to make sure that if for example, you add 5% redundancy, the ED is still able to reconstruct the original firmware image even though it is experiencing roughly 5% packet loss.

$$P_M^N(n) = C_M^N(n, 1) \cdot B_1 + C_M^N(n, 2) \cdot B_2 + \dots + C_M^N(n, M) \cdot B_M, \quad 0 < n \leq N \quad (1.2)$$

If $N = M$, no FEC code is added and the parity check matrix is simply an identity matrix with ones on the main diagonal and zeros elsewhere, meaning $P_M^N(n) = B_n$. If $N > M$, a parity check vector is added after the identity matrix which contains statistically as many zeros as ones in a pseudo-random order. The parity check matrix used is the one defined by Gallager [29] and an example is shown in figure 1.12.

Process summary

The firmware update server should first take decisions to best optimise the firmware process, e.g. choose the class, the data rate and the topology, like running several FUOTA in parallel. In fact, running 30 FUOTA sessions using the highest data rate is still more efficient than running one FUOTA using the lowest data rate as detailed in the performance evaluation of [30]. Multiple gateways could also target a single ED in a round-robin fashion to achieve an overall 100% duty cycle [30]. Whereas the specifications by the LoRa Alliance try to standardise the FUOTA process, the FUOTA routine is open for contribution. Figure 1.13 shows a simple FUOTA session using class C multicast. During the setup phase, the ED is still running in a normal mode. This means that if it is in Class-A, it first has to send an uplink message to trigger the downlink commands. Once all the firmware packets have been received and checked for integrity, the ED can rebuild the whole firmware file and reboot on it. The ED can then resume its updated application.

1.3.3 State-of-the-art of FUOTA on LoRaWAN

Table 1.6 summarises the state-of-the-art for FUOTA-enabled devices on LoRaWAN. The available firmware update servers are shown in table 1.7.

The FUOTA solution from MultiTech only works with the MultiTech mDot with firmware v3.1 or higher and requires the MultiTech Conduit with AEP 1.6 or higher [31]. The MultiTech Conduit acts as a gateway and contains all the needed servers. As the firmware running on the mDot is not available for the public, a separated microcontroller is required to run the application.

Pycom offers a variety of platforms running on microPython, a simplified Python 3 programming language optimised to run on microcontrollers. An implementation example to enable FUOTA capabilities on those platforms is provided⁹. A client library is run along with the main application on the EDs and a firmware update server running on standard python interfaces with the application server. The used strategy, however, does not comply with the standardised FUOTA process stated by the LoRa Alliance.

In March 2018, Arm published a blog article where they presented a reference implementation to perform a FUOTA on Mbed-enabled development boards. The source code they used for the demonstration is available on the GitHub of

⁹<https://docs.pycom.io/tutorials/all/ota/>

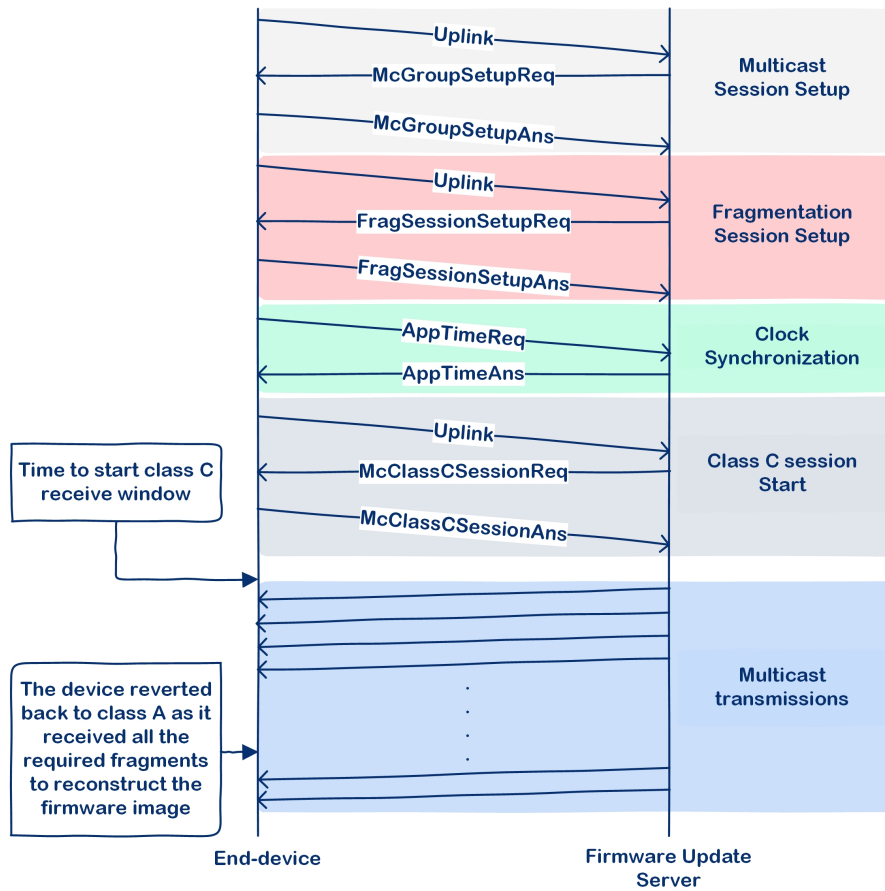


Figure 1.13: FUOTA session using class C multicast [30]

ARMmbed¹⁰ and can be directly flashed on two different platforms. They also added a graphical interface to start a FUOTA on ChirpStack, but it remains an experimental feature until now and doesn't support multicast yet.

In November 2019, STMicroelectronics enhanced I-CUBE-LRWAN to support the latest FUOTA specifications. However, the package is only fully compatible with STM32L4 microcontrollers.

Finally, Actility added the FUOTA feature to ThingPark, an interface to easily manage a huge deployment of EDs.

¹⁰<https://github.com/armmbed/mbed-os-example-lorawan-fuota>

Compatible devices	MultiTech mDot	Pycom modules	Mbed-enabled development boards	STM32L476xx
Provider	MultiTech	Pycom	Arm	STMicroelectronics
Development tool	AT Command	microPython	Mbed OS	I-CUBE-LRWAN
LoRa Alliance compliant	Yes	No	Yes	Yes
Requirements	MultiConnect Conduit	ChirpStack	ChirpStack	-

Table 1.6: FUOTA-enabled devices on LoRaWAN

	Actility ThingPark	ChirpStack	MultiConnect Conduit
Multicast	Yes	No	Yes
Free	No	Yes	Yes
Requirements	-	-	MultiTech mDot

Table 1.7: Firmware update servers for LoRaWAN

Chapter 2

Adapting a reference implementation for large deployments

In this chapter, we will first discuss the reference implementation published by ARMmbed to perform a FUOTA on LoRaWAN. We will identify the parts that still need development and propose a solution to address those gaps.

2.1 A Mbed OS approach by ARMmbed

Arm Mbed OS is a free and open-source embedded operating system specifically designed to be used in IoT platforms. It offers a large repertoire of modular libraries and gained a community of hundreds of thousands of developers all over the world. As the Mbed platform is collaboratively developed by Arm, it is no wonder that it was chosen for the first demonstration of FUOTA on LoRaWAN by Jan Jongboom and Johan Stokking in 2017 (see section 1.3.2 for further details). In December 2018, they publicly released the source code used during the demonstration. It could readily be used on the same platform employed during the demonstration, the L-TEK FF1705, and support for another one was added later. They also added a small script that acted like a firmware update server, however in a simple way. As soon as a device was detected, a FUOTA session was started. A few months after, the two main contributors left Arm and they announced that they did not expect their work to be updated. The source code is nonetheless a solid start to

work with as it complies to the specifications from the LoRa Alliance.

2.1.1 Cryptographic verification of the firmware

On top of handling multicast, fragmentation and clock synchronisation as specified by the LoRa Alliance, cryptographic protection of the firmware is also implemented. For this purpose, a manifest is added to the firmware update file. Among different parameters like the firmware version, it contains information about the source and target of the firmware in the form of a 16 bytes manufacturer and device Universally Unique Identifier (UUID), ensuring both the sender and the recipient are authenticated. It also contains an encrypted SHA256 hash of the firmware that acts as a signature to verify its integrity as shown in figure 2.1. The hash is encrypted with a single-curve Elliptic Curve Digital Signature Algorithm (ECDSA), a variant of the Digital Signature Algorithm (DSA) that can be efficiently implemented on constrained devices while still providing adequate security [24]. If the firmware hash computed on the ED does not match the decrypted hash, the process is aborted and the device resumes its main application.

2.1.2 Memory usage

Although the source code is readily runnable on only two platforms, it is possible to port them to a new one provided some memory requirements are met. First, the ED needs free storage with a size of about three times the expected firmware image size. This storage is split into three distinct firmware slots. The first one contains the incoming firmware data, the second is used to store the results of a delta update and the third contains the current active firmware. The latter is used when computing the delta and to revert back to the former firmware version if a problem occurs with the new one. A bootloader is responsible for copying the new firmware to the device flash memory. Many different storage types can be used. As the RAM size is often about ten times smaller than the flash memory, it is recommended to use an external memory such as an SPI flash memory or an SD card.

The FUOTA capability also comes at the cost of taking up some space in the flash memory as shown in figure 2.2. Without taking into account the mbed-os library, roughly $62kB$ are needed. The memory map was generated using the default Mbed optimisation (-Os) and an external SPI flash memory. The latter would be obviously replaced by another library with a different storage type.

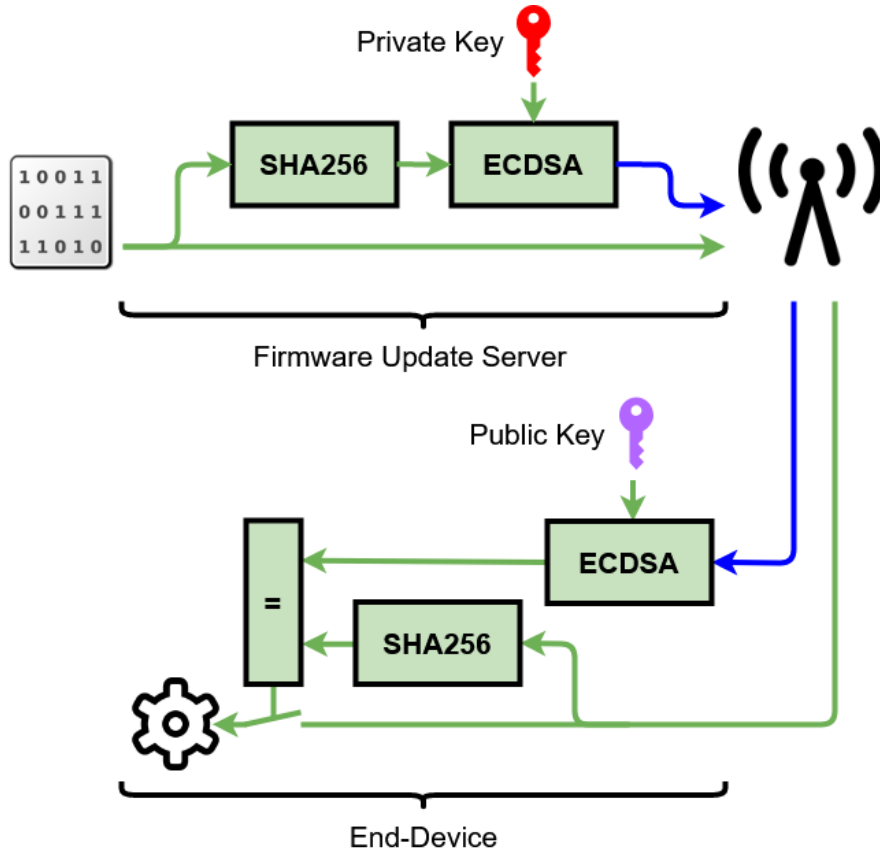


Figure 2.1: Firmware signing strategy with asymmetric cryptography

2.1.3 Issues and requirements

Even though multicast is implemented, the provided firmware update server only supports firmware update jobs to single devices and the interface of ChirpStack suffers from the same flaw. It is thus a necessity to develop a new firmware update server that would be able to handle multiple devices. In a scenario where different types of EDs are deployed, the server should also be able to distinguish them. Last but not least, the server should be easy to integrate among other servers and should offer a convenient interface for humans or for device management platforms.

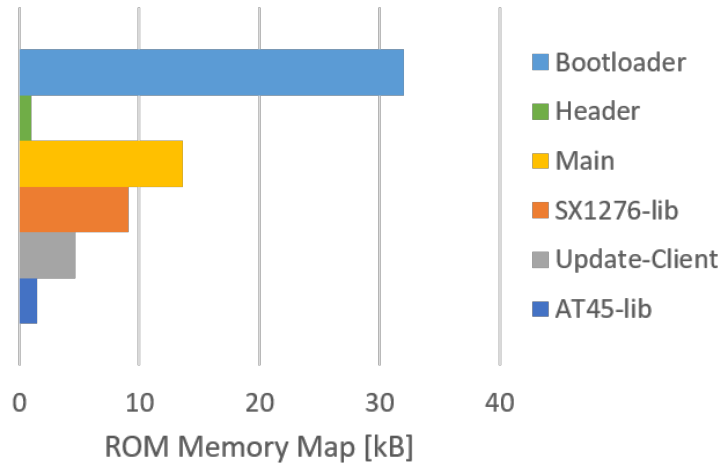


Figure 2.2: FUOTA flash memory requirements

2.2 Design of a new firmware update server for central handling

In order for the new firmware update server to be easily integrated along with other servers, it is designed as a network application coded in Node.js. The asynchronous event-driven trait of Node.js makes it a good choice to run the different services needed to handle the FUOTA process. The LoRaWAN network server stack is implemented with ChirpStack which provides open-source components for LoRaWAN networks. It includes the network server and the application server along with the needed databases and offers a web-interface that can be used to manage users, organisations, applications and devices.

2.2.1 Architecture

The firmware update server is built in the form of small services.

- One service is responsible for communicating with the LoRaWAN application server which offers a RESTful JSON interface secured with JWT authentication (see section 2.2.3 for further details). It allows for example to retrieve the pieces of informations of an ED like the Device Profile but also to send commands for the network server like creating a new multicast session.

- Another service is responsible for listening on the MQTT interface of the application server. MQTT is a lightweight publish-subscribe network protocol used here to receive and send packets to the EDs.
- A next service is responsible for user interaction. A RESTful JSON interface similar to the one of the application server is offered and detailed in section 2.2.3.
- Finally, the last service is responsible for managing a FUOTA process and will be detailed in section 2.2.2.

In order for an ED to be detected as FUOTA-enabled, it needs to send a message containing its current firmware version on a specific port. The device is then added to an internal database and eligible for an update job. Different kinds of EDs are distinguished with a different Device Profile on the application server. The Device profile defines the device capabilities like Class-B or Class-C support and the LoRaWAN parameters like the downlink window delays and data rates as shown in figure 2.3. As those parameters should be provided by the ED manufacturer and may not be identical between different kinds of EDs, it makes sense to have different Device Profile for different kinds of EDs with distinctive firmware images. As such, an individual update job is created for each Device Profile containing devices to be updated. In the case of a delta update, the distinction goes one step further by separating the devices in new jobs depending on their current firmware version.

2.2.2 FUOTA process service

If the update is differential, both firmware images otherwise only the new one are first passed through the `lorawan-fota-signing-tool` provided by Jan Jongboom¹. It computes the difference of both files, adds the signature and creates the fragmented packets. The multicast downlink communication can thus be scheduled at the required time.

In parallel, the firmware update server waits for uplink communications to set up the multicast session and fragmentation session like detailed in section 1.3.2. This means that at least three uplink communications need to happen before starting the multicast downlink communication in order for the update process to succeed. Therefore, a proper time schedule needs to be chosen depending on the expected uplink communication rate of the devices. The multicast downlink

¹<https://github.com/janjongboom/lorawan-fota-signing-tool>

```

{
  "deviceProfile": {
    "id": "a361a7ef-e215-46ab-a95b-766e959588d1",
    "name": "murata",
    "organizationID": "1",
    "networkServerID": "5",
    "supportsClassB": false,
    "classBTimeout": 0,
    "pingSlotPeriod": 0,
    "pingSlotDR": 0,
    "pingSlotFreq": 0,
    "supportsClassC": false,
    "classCTimeout": 0,
    "macVersion": "1.0.2",
    "regParamsRevision": "A",
    "rxDelay1": 0,
    "rxDR0Offset1": 0,
    "rxDataRate2": 0,
    "rxFreq2": 0,
    "factoryPresetFreqs": [],
    "maxEIRP": 0,
    "maxDutyCycle": 0,
    "supportsJoin": true,
    "rfRegion": "EU868",
    "supports32BitFCnt": false,
    "payloadCodec": "",
    "payloadEncoderScript": "",
    "payloadDecoderScript": "",
    "geolocBufferTTL": 0,
    "geolocMinBufferSize": 0
  },
  "createdAt": "2020-04-01T13:14:45.949222Z",
  "updatedAt": "2020-04-01T13:16:28.860507Z"
}

```

Figure 2.3: Device Profile example

communication is cancelled if no devices are properly setup. The multicast session keys are randomly generated and a new temporary multicast group is created on the network server. The same keys are sent to all the EDs included in the update job.

2.2.3 Using a RESTful API as an interface

User interaction with the firmware update server is achieved with a RESTful JSON interface built with Express². A RESTful API allows to GET, PUT, POST and DELETE data with HTTP requests. Its interface is secured with a JSON Web Token (JWT) which is required when performing most requests. The only one that does not need the token is the GET login request that generates a new JWT under the condition that the provided credentials are valid. Using a RESTful API has a huge benefit when talking about a network application as it can be easily interfaced with a foreign application like a device management platform. To visualise and interact with the API's resources, Swagger UI from the Swagger framework³ is added to the application as shown in figure 2.4. It allows getting a complete documentation about the available requests like their usage or their parameters and event to execute them.

The RESTful API allows to retrieve the list of FUOTA-enabled EDs along with their properties like their Device Profile or their current firmware version. The major functionality is however to manage an update job. Devices can be added or removed from the update queue based on their identifier or by their Device Profile. New firmware images can be uploaded and new update jobs can be scheduled or cancelled.

²<https://expressjs.com/>

³<https://swagger.io/>

LoRaWAN FUOTA WebApp REST API

1.0.0OAS3

LoRaWAN FUOTA WebApp

[Terms of service](#)
[Tim Hess - Website](#)
[Send email to Tim Hess](#)

Servers

http://localhost:3000/ - Local server

Devices

GET

/api/devices

Status

GET

/api/status

Get client status

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Status obtained	No links
Media type application/json Controls Accept header Example Value Schema <pre>{ "message": "Welcome to RESTHub for LoRaWAN FUOTA", "mqttIsUp": true, "loraserverIsUp": true}</pre>		
401	Not authenticated	No links

Figure 2.4: Firmware Update Server API

Chapter 3

Design of a test platform for an ultra-low-power application

In this chapter, we will first present the technical choices of the WSN node and propose a general description of the prototype. Finally, we will validate the main application of the PCB.

3.1 Methodology, design choices and objectives

Following the implementation of the new firmware update server, the new objective is to validate the whole FUOTA stack and characterise the energy budget needed during a FUOTA process. However, as the FUOTA stack is not intended to be used on its own but along a running application, a generic use-case is first proposed: audio monitoring of forest ecosystems seems to be a good example for a WSN application. The coverage area is large, the data rate is low as some data processing is performed on the node and the energy budget is limited as energy harvesting solutions would be needed to allow a long device lifespan, which makes LoRaWAN a good choice for communication technology. Audio signals are recorded and patterns are analysed in the spectral domain to count or match the song of birds. A change of library for data processing would thus justify the need to perform a FUOTA.

The L-TEK FF1705 platform used by Jan Jongboom to demonstrate the FUOTA process is not suited for power characterisation. The on-board xDot module from MultiTech, which contains a ARM Cortex-M3 processor and a SX1272 LoRa radio,

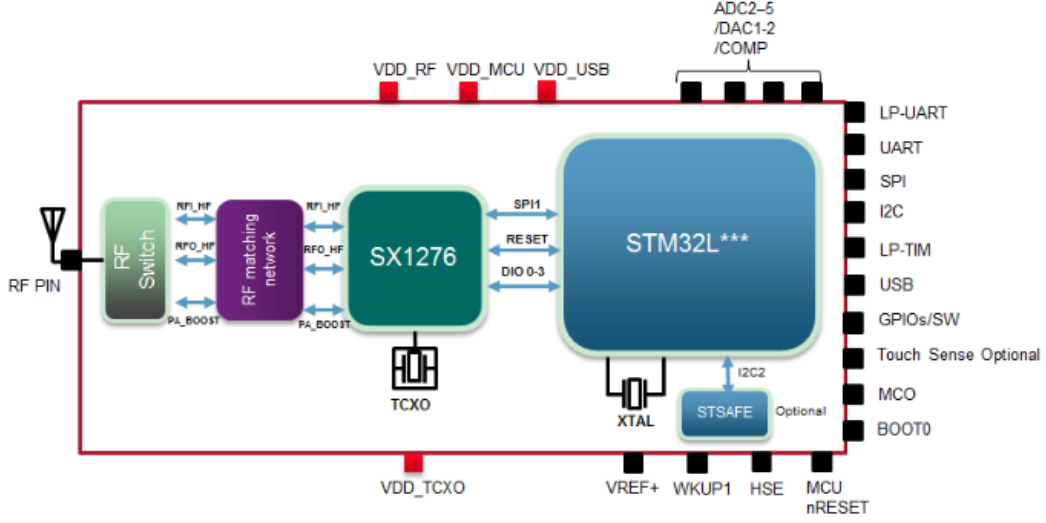


Figure 3.1: Block diagram of the CMWX1ZZABZ module from Murata [32]

only provides one power supply pin. The power consumption of both chips can thus not be distinguished. As we are looking to work with the reference implementation on Mbed-OS, several Mbed-enabled platforms that include a LoRa transceiver are available to choose from, each with its own architecture. Choosing a module that both includes a STM32 microcontroller and a LoRa transceiver offers a huge benefit: its small footprint and the fact that most of the RF components like the RF matching network or the Temperature Compensated Crystal Oscillator (TCXO) for the RF transceiver are already present make it easy to integrate into a platform. Apart from the xDot module already mentioned above, only one other can be taken: the CMWX1ZZABZ module from Murata. By chance, it is a good candidate as it offers separate power supply pin for both the included ARM Cortex-M0+ processor and SX1276 LoRa radio as shown in figure 3.1. Moreover, it takes a lower supply current in receiver LoRa mode at $21.5mA$ compared to the xDot at $26.8mA$.

3.2 Board design

The block diagram of the WSN node is shown in figure 3.2. The area left of the dotted line is responsible for delivering the needed power to the processing and radio unit. It also contains the microphone for audio sensing along with the analog front-end unit. This part is however out of scope of this work and is part of the work of a colleague for his own master thesis. The key point for this part

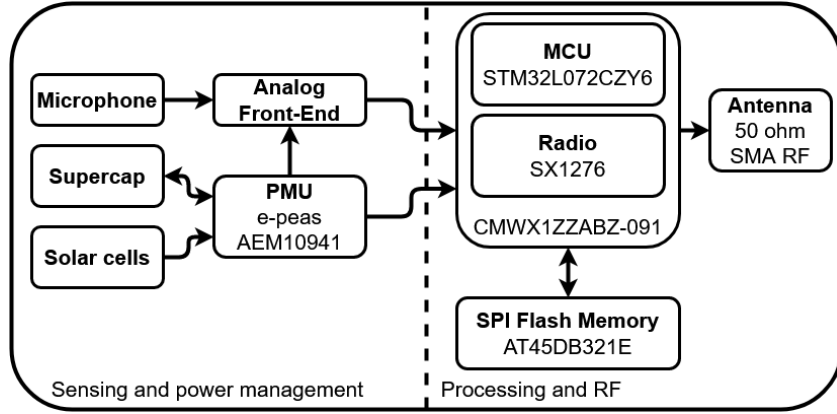


Figure 3.2: Block diagram of the WSN node

is the maximum available energy as the solar cells and supercapacitor are sized to only deliver a maximum average current of $4mA$ at $2.5V$, thus a maximum average power of $10mW$ [4]. An external SPI flash memory is used to store the three firmware slots required for a FUOTA. Internal flash memory would also be supported but not applicable here as microcontroller only offers a size of $192kB$ and firmware images usually exceed $100kB$. Regarding the external SPI flash memory, two technologies dominate the market: NOR and NAND [33]. NOR flash offer a much higher read speed and lower standby current than NAND flash at the cost of a lower write speed. As read operations dominate the FUOTA process and both technologies have similar instantaneous power figures, it is more energy efficient to choose a NOR flash over a NAND flash. The choice fell on the AT45DB321E of Adesto Technologies as it is already used in the reference implementation. The library needed to use it as an external storage layer is thus already provided. The memory offers a size of 32 megabits organised as 8192 pages of 528 bytes each. As the board should be pluggable to the power management board while allowing to easily perform measurements, a stackable design with both male and female pin headers was chosen.

The design was implemented on KiCad, an open-source electronic design automation software where the most advanced tools were implemented by CERN developers. The schematics design is shown in figure 3.3. It includes a LED and a button for debugging purpose and the power supply pin of each component is accessible on the header to allow distinct measurements. Along the ADC pin needed for audio sensing, several other pins are also accessible to offer a generic platform.

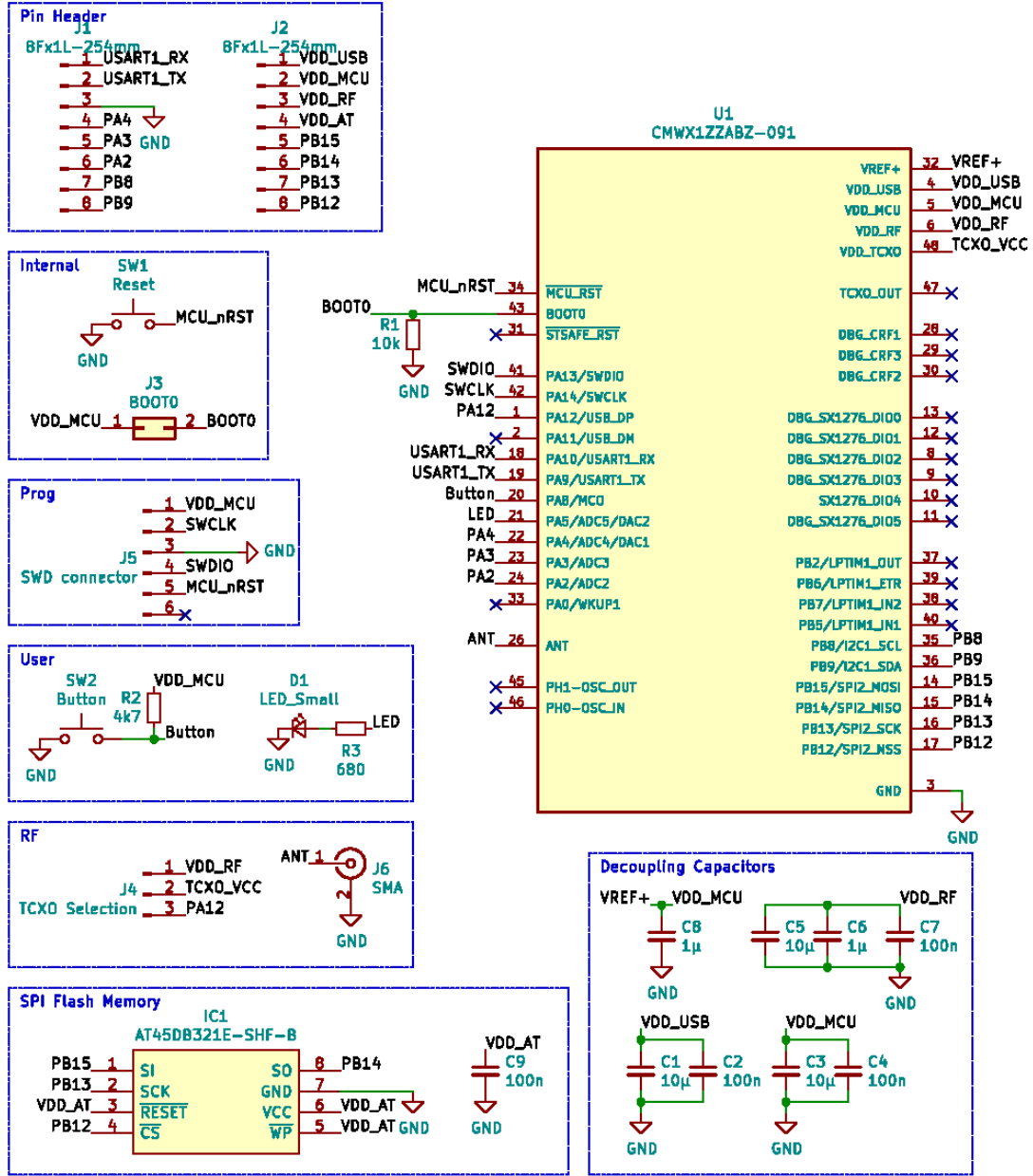


Figure 3.3: Schematic design of the WSN node

The board design is shown in figure 3.4 and is formed by 4 copper layers. The two inner layers are used for VDD and GND power planes and the two outer layers are used for components and signal routing. The width of the path for the antenna is matched at an impedance of 50Ω .

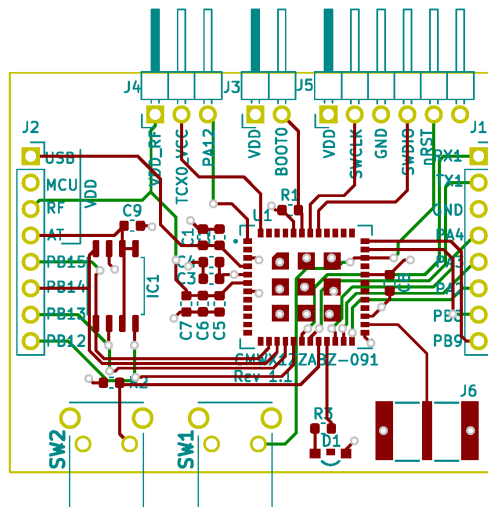


Figure 3.4: Board design of the WSN node

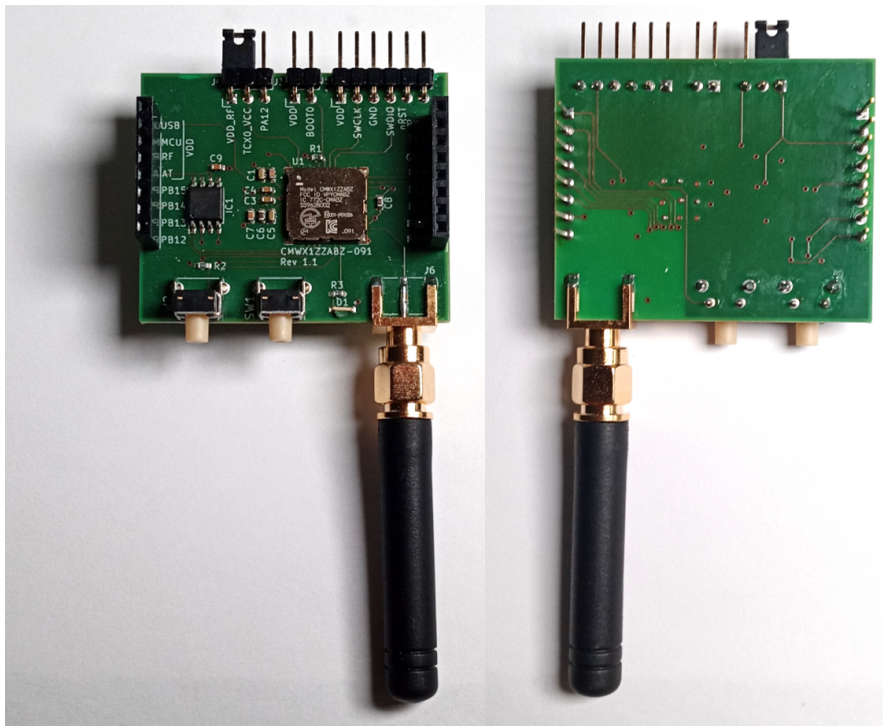


Figure 3.5: WSN node prototype

The board was manufactured by Multi-CB and the resulting prototype after soldering the components is shown in figure 3.5.

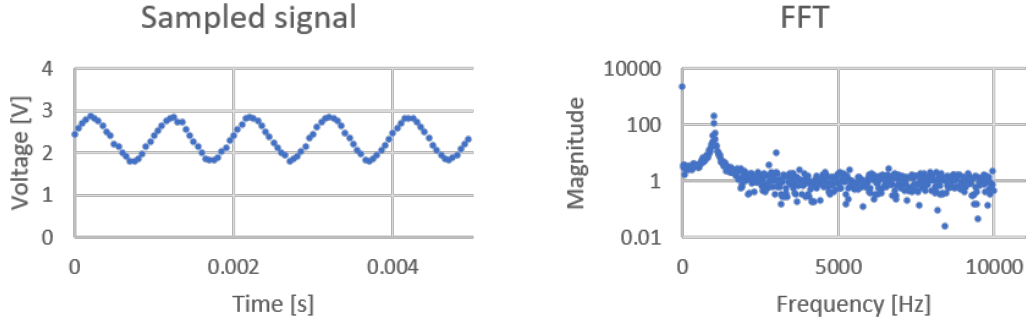


Figure 3.6: Sampled signal and FFT computed by the prototype

3.3 Validation

To validate the prototype, it is tested by running a simple program using Mbed-OS. To minimise the overhead of the operating system and thus its energy consumption, the bare-metal version of the Mbed-OS library is used. The program goes through several states in a loop : sampling, processing and sleep. The sleep state is necessary as the microcontroller exceeds the $4mA$ average energy budget during the sampling and processing state as will be seen later.

The first step is to test the application itself. For this purpose, a $1kHz$ sinus wave is injected to the ADC pin. The signal is sampled at a rate $f_s = 20kHz$ and a FFT with size $N = 1024$ generates the spectral representation of the signal. The FFT is computed using a floating-point complex FFT function from the DPS software library by Arm for Keil. The resolution $\Delta f = \frac{f_s}{N}$ is equal to roughly $19.5Hz$. The sampled signal and its FFT computed by the prototype are shown in figure 3.6. The peak frequency is found at $f_{max} = 1.015kHz$, which matches the estimated resolution. In our use-case however such a resolution is not needed and only a FFT size $N = 128$ is taken.

The next step is to perform a power usage analysis of the application. The current drawn by the prototype is measured with a DMM7510, a high-resolution digital multimeter from Tektronix. A voltage of $2.5V$ is generated with a voltage source. The sleep state time is chosen to not exceed an average power consumption of $10mW$ as shown in figure 3.7. In reality, the average power consumption is equal to $9.87mW$ with an application frequency of $23.8Hz$. The energy required for one cycle is $0.413mJ$ whereas the majority of it is consumed during the processing part as the calculation of the FFT takes about $8.1ms$ whereas the total acquisition time

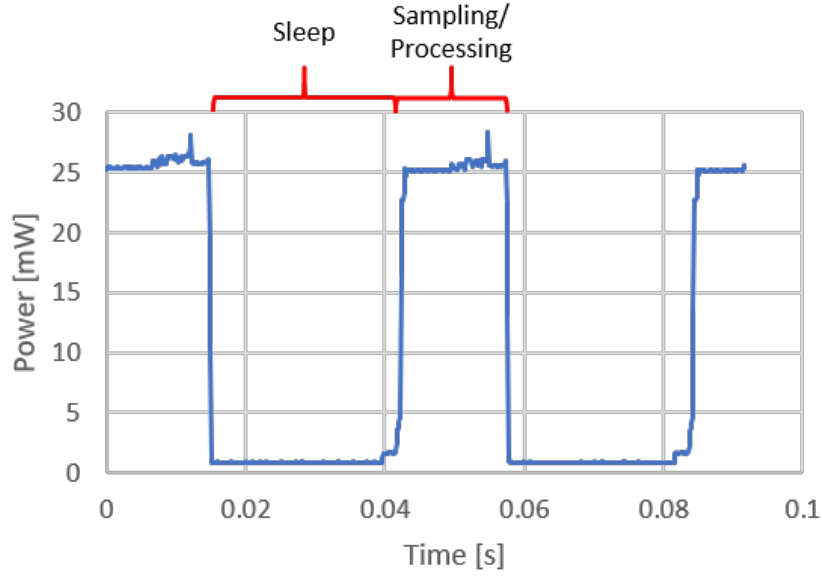


Figure 3.7: Power consumption when using a floating-point complex FFT

is equal to $6.4ms$.

A solution to reduce energy consumption during the processing part is to use another FFT function. As a matter of fact, computing an FFT in a fixed-point format is much more efficient than computing it in a floating-point format as shown in figure 3.8. This allows the processing time to be reduced down to $2.9ms$ with an energy consumption of $0.281mJ$ per cycle without altering the functionality of the platform. This change of a library will thus be experimented during the FUOTA validation.

To characterise the power consumption when using the LoRa radio, an iC880A-SPI LoRa Concentrator is used as a gateway and coupled to a Raspberry Pi which includes the ChirpStack stack. Using a custom LoRaWAN infrastructure at close distance ensures that all the data rates will be available when performing the tests. The power consumption when sending a message is shown in figure 3.9. For this purpose, a $51B$ message is sent with ADR enabled on Class-A. As we are only interested in pure LoRaWAN power consumption, the processor has been put in sleep mode the rest of the time. The average power needed for the total transmission, thus from the start of the uplink transmission to the end of the second downlink window, is $9.3mW$, which is within the available energy budget.

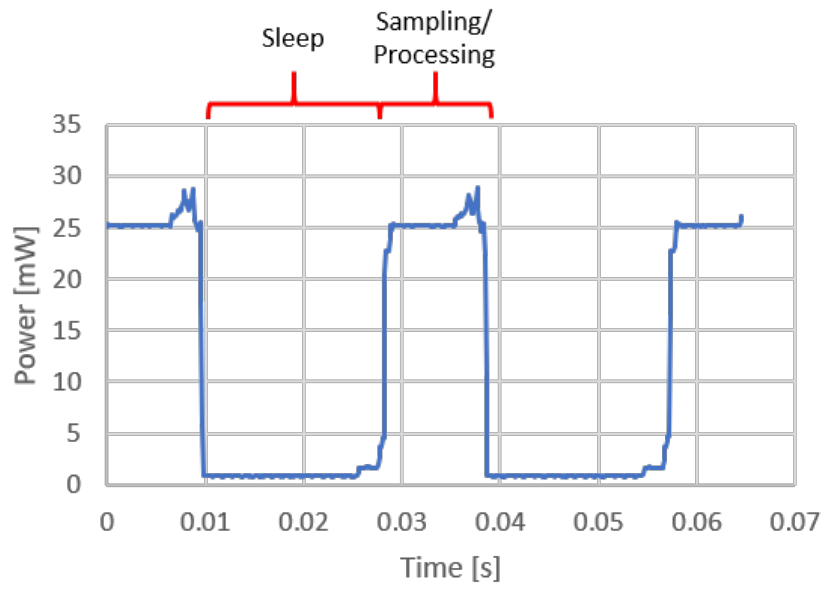


Figure 3.8: Power consumption when using a fixed-point complex FFT

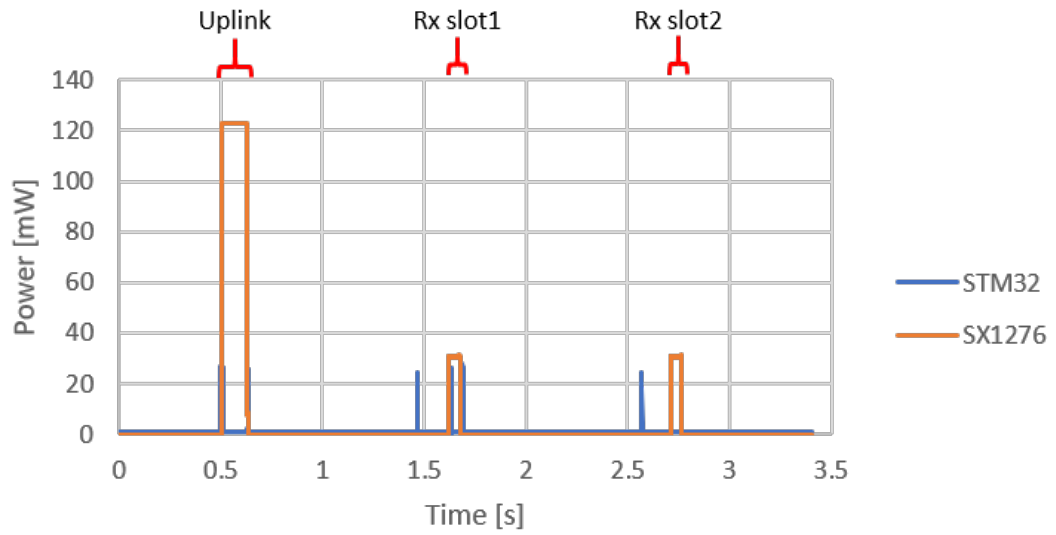


Figure 3.9: Power consumption when sending a LoRaWAN frame

Chapter 4

FUOTA Validation and characterisation of the designed test platform

In this chapter, we will first validate the firmware update server from chapter 2 using the platform from chapter 3. We will next characterise the power consumption needed during a FUOTA process and finally explore techniques to optimise it.

4.1 FUOTA validation

To validate the whole FUOTA stack, the platform is updated to switching from a floating-point to a fixed-point FFT library as laid down in section 3.3. Another advantage is that the new library takes 4946 bytes in the binary file whereas the old one needed 6414 bytes, thus reducing its size by 1468 bytes. By using a file comparison program like the Deep Diff software from Deep Cold Software, the difference between the two binary files can be visualised. Whereas the beginning of both files does not show any similarities as visible in figure 4.1a, the rest of the file is quite similar with only sparse differences as shown in figure 4.1b. This is good news as it should result in a smaller delta file.

The size of the delta file does not change much under the different optimisation levels provided by the GCC_ARM compiler as shown in figure 4.2. Using the default -Os space optimisation, the firmware image is reduced from 135.3kB down

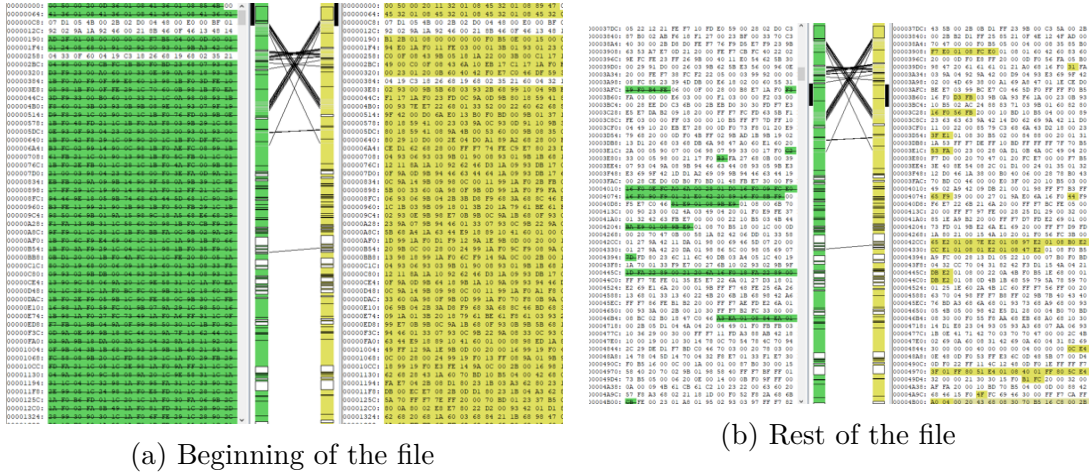


Figure 4.1: Differences in the two binary files

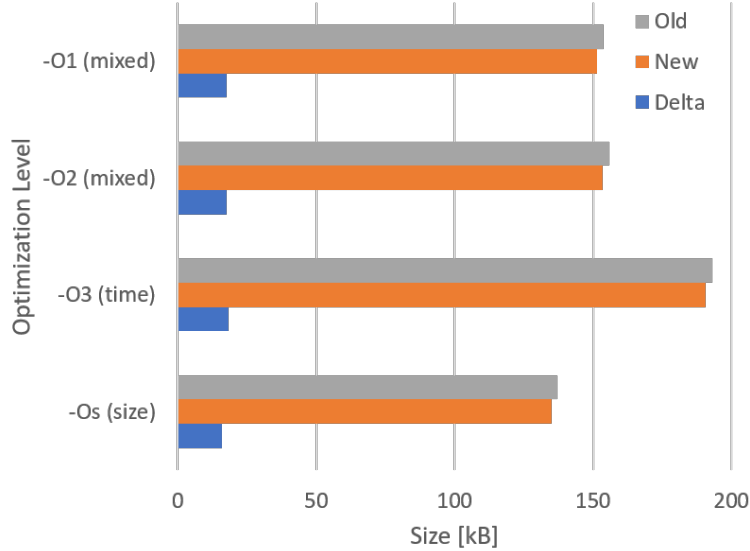


Figure 4.2: File size with different optimisation levels

to delta of $15.8kB$, which is a reduction of about 88%. The delta file is then split into packets suitable for LoRaWAN. As the test conditions allow to use a high data rate, the highest fragmentation size is chosen. The maximum fragmentation size by limitation of lorawan-fota-signing-tool used both for delta calculation and image signing is 204 bytes. The result id thus 79 fragments of 207 bytes as 3 bytes are added to each fragment for correct sequencing.

```

Message Sent to Network Server<CR><LF>
Received message from Network Server<CR><LF>
Received 6 bytes on port 102<CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: handleClockAppTimeAns, correction=
1275034938<ESC>[0m<CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: updateMcGroupsBasedOnNewTime - time is now
1275036610<ESC>[0m<CR><LF>

```

Figure 4.3: Setup - Clock synchronisation

```

Message Sent to Network Server<CR><LF>
Received message from Network Server<CR><LF>
Received 30 bytes on port 100<CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: handleMulticastSetupReq mcIx=0<ESC>[0m<CR>
<LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: <HT>      mcAddr:      0x1FFFFFFF<ESC>[0m
<CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: <HT>      NwkSKey:<ESC>[0m<CR><LF>
<HT>      AB AD 14 A2 84 75 E5 52 D5 7F AB A5 F8 CF 15 9D <CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: <HT>      AppSKey:<ESC>[0m<CR><LF>
<HT>      7E 98 50 7B FF E4 33 8E 8E 84 87 DE 93 CD 1F F8 <CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: <HT>      minFcFCount:  0<ESC>[0m<CR><LF>
<CR>
<ESC>[2K<ESC>[90m[DBG ][LWUC]: <HT>      maxFcFCount:  255<ESC>[0m<CR>
<LF>
<CR>

```

Figure 4.4: Setup - Multicast session

By connecting a serial terminal to the UART port of the platform, the logs during the FUOTA process can be shown. Immediately after booting up, the ED performs a clock synchronisation as shown in figure 4.3. After sending its current time value, it receives a delta time correction that allows it to compute the real current time. Next, after an uplink transmission, the ED receives the multicast session keys as shown in figure 4.4. The fragmentation session is then configured as shown in figure 4.5. Finally, the ED is informed about the parameters of the Class-C session and the multicast transmission starts as shown in figure 4.6. When that time is reached, the ED switches in Class-C and once the fragments are all received, the ED can process the received firmware image like verifying its signature to finally copy the new firmware image to its internal flash memory as shown in figure 4.7.

```

<CR><LF>
Message Sent to Network Server<CR><LF>
Received message from Network Server<CR><LF>
Received 11 bytes on port 101<CR><LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: handleFragmentationSetup fragIx=0<ESC>[ 0m
<CR><LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: FragmentationSessionSetupReq<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      Index:      0<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      McGroupBitMask: 0<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      NbFrag:      40<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      FragSize:     204<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      FragAlgo:     0<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      BlockAckDelay: 0<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      Padding:      163<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][LWUC]: <HT>      Descriptor:   0<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: FragmentationSession starting:<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: <HT>      NumberOfFragments: 40<ESC>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: FragmentationSession starting:<ESC>[ 0m<CR>
<LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: <HT>      NumberOfFragments: 40<ESC>[ 0m
<CR><LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: <HT>      FragmentSize:   204<ESC>[ 0m
<CR><LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][FSES]: <HT>      Padding:        163<ESC>[ 0m
<CR><LF>
<CR>
<ESC>[ 2K<ESC>[ 90m[DBG ][

```

Figure 4.5: Setup - Fragmentation session

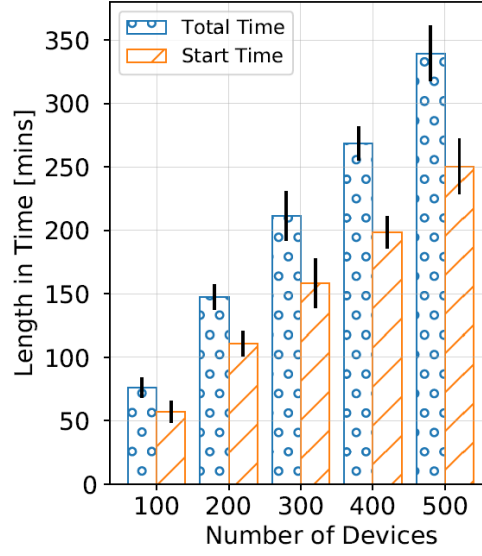


Figure 4.8: Initial phase - Time required [30]

4.2 Characterization

4.2.1 Setup time

The first step during the FUOTA process is to set up the session. For sparse uplink communication, the setup time mostly depends on the time needed to send the 7 uplink packets and receive the 4 downlink transmissions as shown in figure 1.13. The minimum setup time for one ED depends thus on the data rate as a certain amount of time needs to be set between two uplink transmission due to duty cycle restriction. However, if the number of devices in the network increases, the performance is degraded as demonstrated during a study using a discrete-event simulation tool [30]. The total setup time increases with the number of devices as shown in figure 4.8. The start time metric indicates when an ED is ready for FUOTA as it may be started even if the last uplink message is not received. The reason for the increased setup time is that a lot of collisions happen during the transmission phase due to the poor scalability of the LoRaWAN MAC which uses a simple ALOHA protocol. If during a transmission another transmission is detected, the transmission fails and a new one is scheduled. The number of transmissions thus increases as shown in figure 4.9. In a network with 100 EDs, about 94.7 uplink transmissions are required to compensate the collisions.

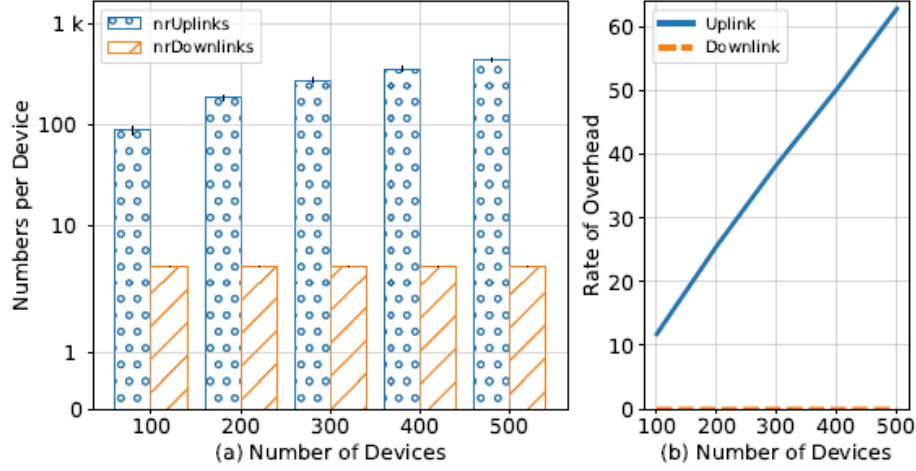


Figure 4.9: Initial phase - Uplinks vs downlinks [30]

4.2.2 Power and energy consumption analysis

The air time for each downlink fragment is $327.94ms$ with $DR5$ and a packet size of 207 bytes. Considering a maximum duty cycle of 10% and a total of 78 fragments, the ED needs to remain during 255.8 seconds in Class-C. The power consumption of the different components is shown in figure 4.10. At each downlink message, the processor exits the sleep mode and stores the fragment into the SPI flash memory. The average power consumption during this state is $32.17mW$ and the total energy needed for each frame is equal to $105.5mJ$, thus resulting in a total energy budget of $8.23J$ for the multicast session.

The power consumption during the processing job of the FUOTA process is shown in figure 4.11. During the first step, the hash of the received fragments is computed to validate the received fragments. Afterwards, the reverse delta is computed from the received fragments and the old firmware image. The power consumption of the external flash memory is increased as write operations draw more power than reading operations.. Finally, the ED is rebooted and the bootloader copies the recomputed firmware image into the internal flash and the application is started. Table 4.1 summarises the average power and total energy during each step. The FUOTA process without taking into account the setup time takes about 5 minutes with an average power consumption of $35.41mW$ and total energy consumption of $10.68J$. As this exceeds the available average power budget, the ED needs to sleep during a certain amount of time to compensate for the higher average power. The sleep mode having an average power consumption of $0.972mW$, the device needs to sleep about 14 minutes. This is an acceptable time considering

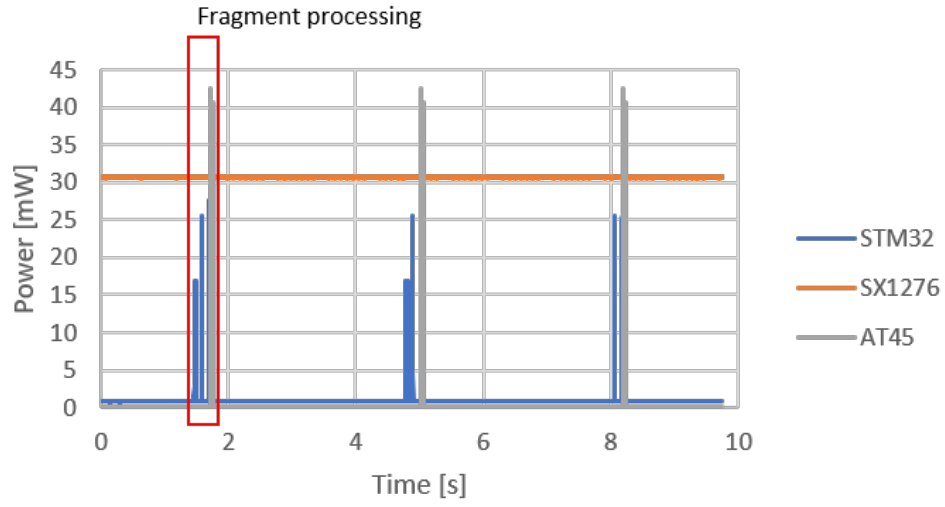


Figure 4.10: Power consumption during the multicast session

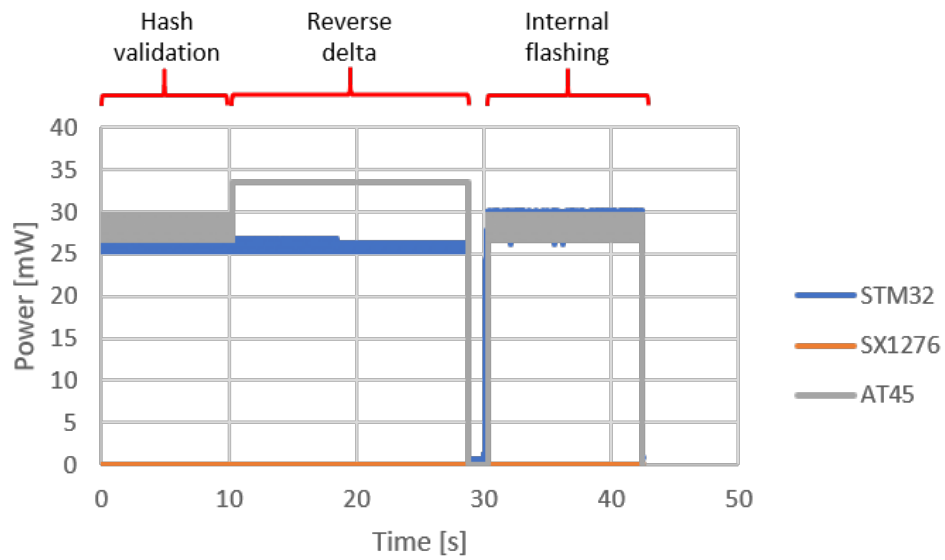


Figure 4.11: Power consumption during the processing job

that a FUOTA is not performed on a regular basis.

	Multicast	Validation	Delta	Flashing	Total
Time [s]	255.76	10.24	18.57	13.67	301.5
Average power [mW]	32.17	53.08	59.14	51.28	35.41
Total energy [J]	8.23	0.543	1.098	0.701	10.68
Time ratio [%]	85.8	3.43	6.23	4.59	100
Energy ratio [%]	77.84	5.14	10.38	6.63	100

Table 4.1: FUOTA process power and energy summary

4.3 Optimization - Multicast data rate

The most energy demanding step of the FUOTA in section 4.1 is the multicast session as it represents 77.84% of the total energy. It can thus be interesting to analyse what parameter can be changed to reduce it.

An important factor is the data rate choice during the multicast session. The data rate both influences the air time and the maximum packet size as shown in table 1.1. Using the same delta image size of 15804 bytes, the total energy needed for the multicast session is shown in figure 4.2. Increasing the data rate by one roughly decreases the energy by two. It is thus important to choose the highest possible data rate to achieve the lowest energy consumption.

Data rate	DR0	DR1	DR2	DR3	DR4	DR5	DR6
Number of fragments	330	330	330	142	78	78	78
Air time [ms]	2467	1316	617.5	615.4	584.2	327.9	164.0
Total time [s]	8141	4342	2038	873.9	455.7	255.8	127.9
Total energy [J]	261.9	139.7	65.57	28.12	14.66	8.23	4.12

Table 4.2: Energy consumption during the multicast Class-C session

Chapter 5

Discussion and Perspectives

Being able to already validate the FUOTA process on the ultra-low-power requirements of an ED shows that LoRaWAN technology has a promising future. At some point, FUOTA capabilities will become a mandatory feature and communication technologies that will not be able to offer it will probably not be able to keep up. However, there is still a lot of room for improvement. This section aims to explore solutions to enhance the FUOTA efficiency and provide some perspectives for future work.

As a matter of fact, FUOTA on LoRaWAN is still in the phase of a proof of concept and it is not recommended yet to use this feature in mature projects. The FUOTA capability on LoRaWAN is only possible thanks to the three specifications from the FUOTA Working Group. They however work exclusively on the application layer which is far from ideal as they add a lot of overhead to the code of an existing application. It is expected that a lot of new MAC commands will be added as it is already the case for the clock synchronisation in LoRaWAN v1.1 to make the FUOTA process much more efficient. This will greatly relieve the overhead on the microcontroller and should thus reduce the overall energy consumption. Adding FUOTA capabilities to an ED will also be much easier. Moreover, the new specifications should eliminate the discrepancies currently present about the Class-B functionality. Using this class instead the Class-C during the multicast session will dramatically reduce the energy consumption of the whole FUOTA process.

As the duty cycle on the 869.525MHz frequency is limited to a maximum of 10%, nothing is received during 90% of the time. This means a lot of energy is wasted since the radio still needs to remain active in receive mode. A better

alternative would be to use the Class-B during the multicast session instead of the Class-C. However, as the Mbed-OS stack does not support the Class-B yet, the following values are estimated using the LoRaWAN calculator. The resulting total energy is shown in table 5.1. At the highest data rate, the energy is reduced down to 15.8%.

Data rate	DR0	DR1	DR2	DR3	DR4	DR5	DR6
Total energy [J]	33.19	17.98	8.746	3.751	1.963	1.162	0.6502

Table 5.1: Energy consumption during the multicast Class-B session

Although the standalone firmware update server is functional, it is far from being perfect. In fact, the strength of ChirpStack is to offer a complete open-source LoRaWAN stack without having to go through the hassle of installing all the individual components needed to have a running LoRaWAN infrastructure one by one. The integration of the much needed FUOTA concept in the ChirpStack ecosystem would be ideal and even more so because ChirpStack already offers all the basic concepts our firmware update server is based on. Indeed, ChirpStack allows managing a diversity of EDs spread all over the world whether this is done through a user-friendly graphical interface or through a RESTful API. FUOTA capabilities limited to one device was added as an experimental feature in v3.0.0 in May 2019 and never updated since. They wanted to first gather some feedback, but they are probably waiting for the release of the new LoRaWAN specifications before finalising their work.

Beyond the many doors for improvements LoRaWAN offers, semiconductors and microcontrollers will also continue to evolve toward better energy efficiency. The new generation SX1261 and SX1262 LoRa radio offer a power consumption two times lower than the power consumption of the first one. This will allow to roughly divide the energy consumption during the multicast session by two. Beyond the energy required for the communication, a lot of computational time is spent during the hash calculation and the calculation of the reverse delta. As this feature is not only specific to FUOTA on LoRaWAN, we could imagine having a dedicated external memory for OTA update that would perform these tasks in an energy-efficient manner.

Finally, as the LoRaWAN technology gains a lot of popularity, many new gateways will be installed in the following years. Having multiple gateways at the reach of an ED will allow to choose a higher data rate and to increase the throughput by using multiple gateways at once [30].

Conclusion

Concerns about security vulnerabilities or feature update in IoT deployments have made FUOTA capabilities a much-needed feature. This concept might however not be suitable to a lot of devices. There may be strong energy limitations on the device itself (e.g. ED powered by solar cells) or limitations with its communication technology (e.g. limited uplink duty-cycle on LoRaWAN) . Those limitations lead therefore to the following questions:

- How can the concept of OTA update be transposed into LPWAN and LoRaWAN specifically?
- Is the concept of OTA update compatible with ultra-low-power requirements of an ED and what mechanisms can be used to decrease the energy consumption during the OTA update process?

To respond more precisely to these questions, our work was divided into five chapters. The fundamentals behind OTA update and LoRaWAN were firstly reviewed in chapter 1. We explained the challenges that occur when performing an OTA update and reminded the specifications of LoRaWAN. Moreover, we defined some of its relevant figures of merit (maximum throughput and power consumption). The new challenges raised when performing an OTA update on LPWAN and on LoRaWAN specifically were then introduced. A state-of-the-art of FUOTA on LoRaWAN was also presented in chapter 1 and we identified that currently, no fully open-source solution exists. Although ChirpStack includes an experimental feature for FUOTA, it does not allow true multicast sessions with multiple devices and is hence not usable in real-life deployments.

Based on this analysis, a reference implementation on Mbed-OS was first reviewed in chapter 2 before giving the requirements of a new firmware update server. A web application that handles differentiated FUOTA was then proposed. The firmware update server is launched as a network application and allows to easily manage a FUOTA on a large device scale while offering a RESTful API as

an interface. Its compatibility with the widely used ChirpStack LoRaWAN network server stack makes also its installation pretty easy.

A test platform design working on solar cells was proposed in chapter 3. The purpose of this device was to validate the FUOTA process with an energy constraint as the maximum average power consumption could not exceed $10mW$. Different tests were realised to validate the test platform and characterise its power consumption during its normal application.

The FUOTA process was validated and fully characterised using our firmware update server and test platform in chapter 4. We first demonstrated the benefit of performing a differentiated update as the initial firmware image was reduced from a size of $135.3kB$ down to a size of $15.8kB$, which represented a reduction of about 88% in size. We then measured the power consumption during the whole FUOTA process which resulted in average power consumption of 35.41 with a time duration of about five minutes. Hence we showed that a FUOTA is compatible with our platform as it only needs to rest for about 14 minutes to recover which is an acceptable time since a FUOTA is not performed on a regular basis.

The benefit of using Class-B instead of Class-C during the multicast session was discussed in chapter 5. This change of class resulted in an energy consumption reduced down to 15.8%. Finally, perspectives for future work were proposed. Transferring the specifications for FUOTA in MAC commands would reduce the overhead on the microcontroller and ease the integration. A necessary improvement for ChirpStack is to provide a complete FUOTA solution inside its ecosystem. This would, therefore, result in better integration of the firmware update server providing a complete open-source LoRaWAN stack that stands out from the other available LPWAN technologies.

Bibliography

- [1] Dave Evans. “How the Next Evolution of the Internet Is Changing Everything”. In: 2011.
- [2] Manos Antonakakis et al. “Understanding the Mirai Botnet”. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110. ISBN: 978-1-931971-40-9. URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis>.
- [3] Maurizio Capra et al. “Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World”. In: *Future Internet* 11 (Apr. 2019), p. 100. DOI: 10.3390/fi11040100.
- [4] Martin Braquet. “Design of an ultra-low-power energy-harvesting audio sensor for ecosystem monitoring”. Master Thesis. UCL - Ecole polytechnique de Louvain, 2020.
- [5] N. Jain, S. G. Mali, and S. Kulkarni. “Infield firmware update: Challenges and solutions”. In: *2016 International Conference on Communication and Signal Processing (ICCSP)*. 2016, pp. 1232–1236.
- [6] “Atmel AT02333: Safe and Secure Bootloader Implementation for SAM3/4”. In: (), p. 41.
- [7] Kais Mekki et al. “A comparative study of LPWAN technologies for large-scale IoT deployment”. In: *ICT Express* 5.1 (2019), pp. 1–7. ISSN: 2405-9595. DOI: <https://doi.org/10.1016/j.icte.2017.12.005>. URL: <http://www.sciencedirect.com/science/article/pii/S2405959517302953>.
- [8] Actility Gemalto and SemTech. “LoRaWAN Security : Full end-to-end encryption for IoT application providers”. In: *A white paper prepared for the LoRa Alliance* (Feb. 2017).
- [9] LoRa Alliance Technical Committee. “LoRaWAN 1.0.2 Specification”. In: (July 2017), pp. 1–70.

- [10] LoRa Alliance Technical Committee. “LoRaWAN 1.1 Specification”. In: (Oct. 2017), pp. 1–101.
- [11] LoRa Alliance Technical Committee. “LoRaWAN 1.0.3 Specification”. In: (July 2018), pp. 1–72.
- [12] LoRa Alliance. “LPWA Technologies Unlock New IoT Market Potential”. In: *Machina Research* (Nov. 2015).
- [13] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. “A survey on LPWA technology: LoRa and NB-IoT”. In: *ICT Express* 3.1 (2017), pp. 14–21. ISSN: 2405-9595. DOI: <https://doi.org/10.1016/j.ictexpress.2017.03.004>. URL: <http://www.sciencedirect.com/science/article/pii/S2405959517300061>.
- [14] Semtech Corporation. “AN1200.22 LoRa Modulation Basics”. In: (May 2015), pp. 1–26.
- [15] K. Mikhaylov, -. Juha Petaejaejaervi, and T. Haenninen. “Analysis of Capacity and Scalability of the LoRa Low Power Wide Area Network Technology”. In: *European Wireless 2016; 22th European Wireless Conference*. 2016, pp. 1–6.
- [16] LoRa Alliance. “RP002-1.0.0 LoRaWAN Regional Parameters”. In: (Nov. 2019), pp. 1–88.
- [17] Finnish Communications Regulatory Authority (FICORA). “Regulation on collective frequencies for licence-exempt radio transmitters and on their use”. In: (Dec. 2015), p. 22.
- [18] STMicroelectronics. “AN5408 Migrating from STM32L0, STM32L1, STM32L4 Series associated with SX12xx radio, to STM32WLEx microcontrollers”. In: (Dec. 2019), pp. 1–22.
- [19] Knud Lasse Lueth. “State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating”. In: (Aug. 2018).
- [20] M. Pule and A. M. Abu-Mahfouz. “Firmware Updates Over the Air Mechanisms for Low Power Wide Area Networks: A Review”. In: *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. 2019, pp. 1–7.
- [21] Jaydip Sen. “A Survey on Wireless Sensor Network Security”. In: *CoRR* abs/1011.1529 (2010). arXiv: 1011.1529. URL: <http://arxiv.org/abs/1011.1529>.
- [22] Philip Levis and David Culler. “Maté: A Tiny Virtual Machine for Sensor Networks”. In: *SIGPLAN Not.* 37.10 (Oct. 2002), pp. 85–95. ISSN: 0362-1340. DOI: 10.1145/605432.605407. URL: <https://doi.org/10.1145/605432.605407>.

- [23] Jetmir Haxhibeqiri et al. “A Survey of LoRaWAN for IoT: From Technology to Application”. In: *Sensors* 18 (Nov. 2018), p. 3995. DOI: 10.3390/s18113995.
- [24] Johan Stokking Jan Jongboom. “Enabling firmware updates over LPWANs”. In: (2018), pp. 1–3.
- [25] FUOTA Working Group of the LoRa Alliance Technical Committee. “LoRaWAN Remote Multicast Setup Specification v1.0.0”. In: (Sept. 2018), pp. 1–18.
- [26] FUOTA Working Group of the LoRa Alliance Technical Committee. “LoRaWAN Fragmented Data Block Transport Specification v1.0.0”. In: (Sept. 2018), pp. 1–30.
- [27] FUOTA Working Group of the LoRa Alliance Technical Committee. “LoRaWAN Application Layer Clock Synchronization Specification v1.0.0”. In: (Sept. 2018), pp. 1–13.
- [28] FUOTA Working Group of the LoRa Alliance Technical Committee. “FUOTA Process Summary Technical Recommendation”. In: (Jan. 2019), pp. 1–11.
- [29] Robert G. Gallager. “Low-density parity-check codes”. In: *IRE Trans. Inf. Theory* 8 (1962), pp. 21–28.
- [30] Khaled Abdelfadeel et al. *How to make Firmware Updates over LoRaWAN Possible*. 2020. arXiv: 2002.08735 [cs.NI].
- [31] MultiTech. “MTDOT Developer Guide”. In: (2019), pp. 1–79.
- [32] Murata. “Sub-G Module Data Sheet - CMWX1ZZABZ-091”. In: (Oct. 2018), pp. 1–16.
- [33] Arie Tal. “Two flash technologies compared: NOR vs NAND”. In: *White Paper of M-SYstems* (2002).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl