Delft University of Technology
Master of Science Thesis in Embedded Systems

Energy Consumption and Scalability of Transmitting Firmware Updates Over LoRa

Stan van Nieuwamerongen

TU Delft Delft University of Technology

Energy Consumption and Scalability of Transmitting Firmware Updates Over LoRa

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Stan van Nieuwamerongen
S.T.vannieuwamerongen@student.tudelft.nl
stan@dotline.nl

25th of August 2021

Author
Stan van Nieuwamerongen (S.T.vannieuwamerongen@student.tudelft.nl)
(stan@dotline.nl)
Title
Energy Consumption and Scalability of Transmitting Firmware Updates Over LoRa
MSc Presentation Date
27th of August 2021

Graduation Committee

Dr. Przemysław Pawełczak	Delft University of Technology
Dr. Jérémie Decouchant	Delft University of Technology
Thijs Buuron	TWTG

Abstract

The rapid growth of LoRa sensor networks lead to more and more maintenence challenges. One of them is wirelessly updating the firmware, especially for the ones that are hard or dangerous to reach. Is it feasible to do a firmware update over LoRa, and what is its additional power consumption of the wireless sensors? In this thesis a LoRa Class B Firmware Update Over The Air (FUOTA) is implemented and evaluated. Up to 100 end-devices are used to research the scalibilities and power consumption of the end-devices. The focus is mainly on the tranmission of the firmware itself rather than installing the firmware on the end-devices.

This study introduces a power consumption model based on measurements of real hardware. After that, experiments are performed to evaluate the packet losses when scaling up the number of end-devices. These experiments show that the setup time increases with the number of end-devices due to the duty cycle restriction of the gateway. Antother experiment, focusing on end-devices that are part of the network but do not need to be updated, shows these end-devices suffer in terms of packets loss due to packet blockings during a firmware update.

The extra setup time needed when scaling up the number of end-devices causes higher power consumption when more devices needs to be updates. To reduce the energy consumption during this setup phase, an improvement to the communciation protocol is presentated at the end of this thesis. It reduces the number of times receive windows are opened while nothing is send.

Acknowledgement

My special thanks goes to Dr. Przemysław Pawełczak who has guided me during my master thesis. His calm and flexible way of guidance helped was of big help. The regular meetings help me stay on track and his comments on my thesis where of great help. Thank you very much.

I want to thanks Thijs Buuron for working with me and the useful brainstorm session. It was a pleasant to be working with you.

I would like to express my sincerest thanks to my exam committee Dr. Przemysław Pawełczak, Dr. Jérémie Decouchant, and Thijs Buuron for taking the time to read through my thesis.

I am extremely grateful to my parents for there love and support during the less fun times of my master thesis. I am also grateful to my friends who helped me relax and take some time of.

Stan van Nieuwamerongen

Delft, The Netherlands
25th August 2021

Contents

Acronyms

ADR Adaptive Data Rate

BW Bandwidth

CSS Chirp Spread Spectrum

CR Code Rate

FUOTA Firmware Update Over The Air

MAC Medium Access Control

SF Spreading Factor

Chapter 1

Introduction

The rapid growth of wireless sensor networks leads to the development of many new devices every day. With more devices, maintenance of the network and the devices itself becomes a bigger challenge. Keeping the firmware of a wireless sensor up-to-date is challenging, especially at places that are hard or dangerous to reach. To make the process of updating wireless sensors easier, Firmware Update Over The Air (FUOTA) is one of the best solutions. FUOTA makes it possible to send firmware updates wirelessly to devices without the need to physically reprogram each device separately.

In this thesis we research the energy impact of sending a firmware update over LoRa. LoRa is a popular wireless technology mainly implemented in battery powered wireless sensors due to its long-rang and low-power characteristics. The main focus lies on the transmission of the firmware itself rather than the installation of the firmware. We look at different energy consuming events during FUOTA and how they change when more devices are added or when the firmware size increases. The purpose of this thesis is to give a better understanding of the power consumption during FUOTA, and to estimate the decrease in battery life.

The thesis is divided into four parts. The remainder of this chapter gives an introduction to LoRa/LoRaWAN and describes the research questions and motivation. Chapter 2 provides additional background knowledge needed for the remaining of the thesis. Answers to the research questions can be found in Chapter 3, where multiple experiments are performed on real hardware. This chapter also discusses the feasibility of FUOTA, and a small improvement over the existing communication protocol is given. Lastly the conclusion can be found in Chapter 4.

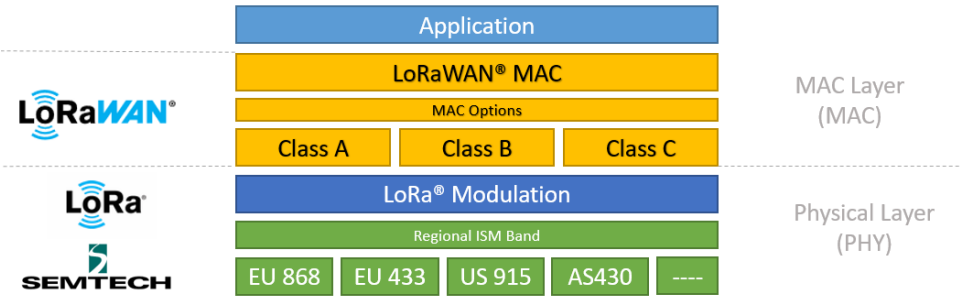


Figure 1.1: **LoRa and LoRaWAN stack. LoRa operates at the licence free ISM (Industrial, Scientific, and Medical) frequency bands, which differ for Europe (EU), United States (US), Asia (AS) and other continents. Figure taken from [9].**

1.1 LoRa and LoRaWAN

LoRa and LoRaWAN together form a wireless communication stack designed for long range and low power consumption. Typically, LoRa is found in low throughput battery powered sensor devices. LoRa describes the physical layer, where LoRaWAN describes the Medium Access Control (MAC) layer [9]. Figure 1.1 shows the structure of the LoRa/LoRaWAN stack. No dedicated frequency bands are assigned to LoRa, therefore it operates in the licence free ISM (Industrial, Scientific, and Medical) frequency spectrum. For example, in Europe the 868 MHz frequency band is part of the ISM frequency bands.

1.1.1 LoRa

LoRa is a modulation technique designed by Semtech [27] for low-power, long-range communication purposes. The communication distance can go from five kilometers in an urban environment to fifteen kilometers in open field [9]. To ensure the low-power characteristics, a compromise needs to be made between data rate and the power consumption. Typically, the data rate of LoRa is defined in bytes per second, which should be just enough to send small amounts of sensor data a few times a day.

LoRa uses a technique called Chirp Spread Spectrum (CSS) modulation. This technique spreads the data over the complete available bandwidth to make it more robust against noise. This makes LoRa better suitable for long range communication. Initially, CSS was developed for radar systems in the 1940's. However, in the last 20 years the interest to use this spreading spectrum within wireless communication has increased due to its low power characteristics [10].

Communication messages are encoded in q so-called chirp signals. An up-chirp signal increases the frequency over time, where a down-chirp signal reduces the frequency over time. At the start of each message a so-called preamble is sent, which contains a few up-chirps followed by two down-chirps. After the preamble, data is sent using up-chirps. Each chirp contains one symbol; how many bit can be encoded in one symbol depends on the Spreading Factor (SF) and the Bandwidth (BW). The LoRa regional parameters for Europe [8] states

Data rate	Spreading factor	Bandwidth (kHz)	Bitrate (Bits/s)
0	12	125	250
1	11	125	440
2	10	125	980
3	9	125	1760
4	8	125	3125
5	7	125	5470
6	7	250	11 000

Table 1.1: **Data rate configuration for the** 863-870 MHz **licence free frequency band in Europe [8].**

that the bandwidth 125 kHz and 250 kHz can be used in combination with SF12 to SF7. The combination of the SF and bandwidth is defined as data rate, and can be found in Table 1.1

The SF used for LoRa are inherently orthogonal [9]. This means that different spreading factors do not create independent channels. Although signals modulated with different SFs do not interfere, they do appear as noise to each other. When sending multiple messages simultaneously over the same channel with the same SF, they might collide. However, one of the messages may survive if its signal is more than 6 dB stronger.

1.1.2 LoRaWAN

Above the LoRa physical layer is LoRaWAN (illustrated in yellow in Figure 1.1). LoRaWAN implements the medium access (MAC) protocol, including security, and joining and rejoining the network. Advanced routing protocols are not necessary because LoRa devices use simple ALOHA-like multiple access technique and only communicate with one or multiple gateways. The lack of complexity makes it possible to maintain the low-power characteristics because the number of MAC messages are limited. However, this also results in reliability and scalability issues.

LoRaWAN can operate in three different classes: class A, B, and C, illustrated in Figure 1.2. Starting with class A which is the default class and should be implemented by all LoRa enabled devices. During class A, the end device is in sleep mode most of the time, it only awakes for specific device-related processes or when it needs to send a message to the gateway. When the end-device is in sleep mode, it is not possible to send downlink messages (messages sent from the gateway to end-device) to the end-device. Only when the end-device sends an uplink message (messages send from end-device to gateway), two small downlink receive windows will be opened which can be used by the application server to send messages to the device. This means that downlink messages can only be send after the end-device sends a uplink message. This class is the most energy efficient class and is required to be implemented by the end-device. The other two classes are optional.

In Class B, receive windows are not only opened after every uplink message, they also open periodically, for example each 16 seconds. These periodical receive windows are called ping-slots. During each ping-slot a downlink message can be received by the end-device. To make sure ping-slots are synchronized

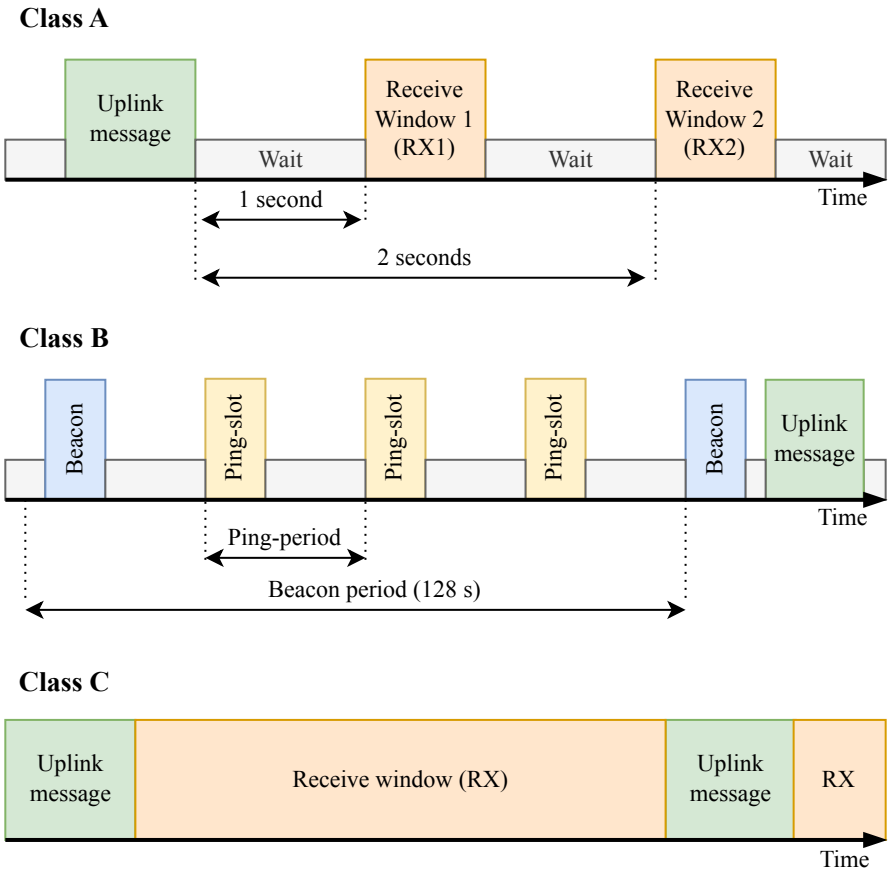


Figure 1.2: **LoRaWAN operational classes.**

with the gateway, the gateway sends beacons every 128 seconds. The end-device will schedule the ping-slots based on the time it has received a beacon. In this way, the gateway can send downlink messages at the exact same time as the end-device opens a receive window.

Considering Class C, this class keeps the receive window open at all times except when it is sending an uplink message. This means as long as the end-device is not sending any data, it can always receive a downlink message. This class is mainly implemented in devices connected to a constant power supply because it uses significantly more energy than the other two classes, which makes it unsuitable for battery powered devices.

It is possible to switch between classes at run-time. For example, if a firmware update needs to be sent, a device might temporarily switch to class B or class C until the firmware update is done.

Network Architecture

Figure 1.3 illustrates a typical LoRa network from end-to-end [9]. Starting with the end-devices, typical battery embedded sensors, they are connected to one or multiple gateways via a wireless LoRa connection. The Gateway forwards all

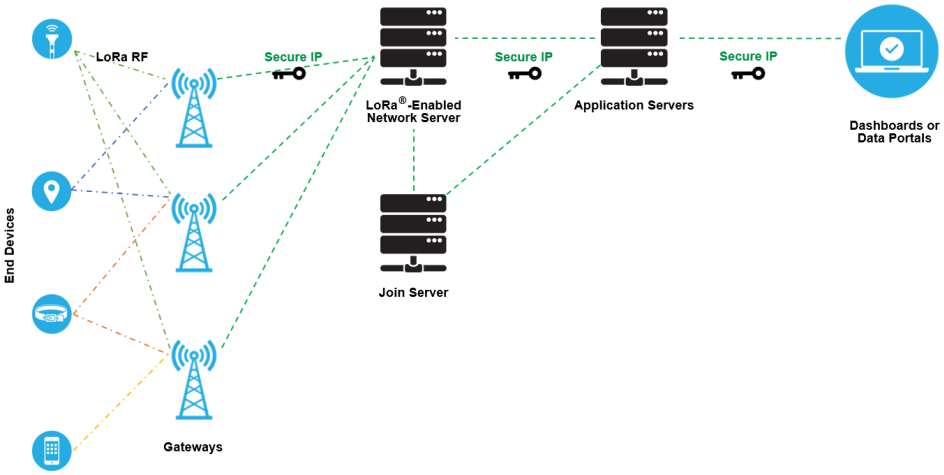


Figure 1.3: **Overview of a typical LoRa network. The blue circles on the left are the LoRa enabled end-devices, they are wireless connected via LoRa with one or more gateways. The gateways forwards all the packets between the end-device and the network server. The green dotted lines between the servers are internet connections, which can be wired or wireless. Figure taken from [9].**

the packets between the end-devices and the Networks Server using the Internet. Usually there is only one Network Server, which manages the entire network and adapts it to changes in the network. It handles device addressing, MAC layer requests from the end-device, forwarding application level packets to the application server, and it checks the integrity of each incoming message. The Join Server manages the over-the-air activation of each end-device. Typically only one join server is present in the network. Next is the Application Server; there can be multiple Application Servers in one network. The Application Server handles all the application level messages and their end-to-end encryption. For example, it can receive sensor values from an end-device and forward it to a database or a user interface.

1.2 Problem Statement and Motivation

Many companies decided to use LoRa as communication protocol for their wireless sensors because of its low-power long-range characteristics. However, as Stan Lee once wrote in one of his Spiderman comics: “With great power comes great responsibility”. Or actually in our case: ”With low-power comes great responsibility”. In fact, the low-power capability of LoRa can result in sensors that can be powered for multiple years on a single battery. That leaves us with the responsibility to maintain the security of the device by keeping these sensors up-to-date. Code is never 100 % bug free, it is almost certain that bugs will be discovered while the sensors are already deployed in the field. The ability to

updating sensors in the field will significantly improve the security of the device. Additionally, the ability to update deployed devices makes it possible to shorten the pipeline as mistakes can be corrected after deployment. This might result in an overall cost reduction once implemented correctly.

Many implementations for FUOTA are known already. For example, wireless earbuds can easily be updated with a smartphone using bluetooth or smart lighting can be updated over-the-air with ZigBee. Although the LoRa-Alliance has described a proposal on how to implement FUOTA [3], [32] has given a demonstration, and [19] provides example code, the number of actual deployed implementations are limited.

There can be many reasons why the number of implementations are still so limited. One of the main reasons is the low data rate of LoRa, typically in order of bytes per second. These low data-rates make it possible to transmit data over a long distance with an extremely small amount of power. However for FUOTA, a relatively large amount of bytes needs to be sent over the air, up to hundreds of bytes. Additionally, in Europe, one device is only allowed to transmit 1 % of the time, making it even more challenging to implement FUOTA. Because of these limitations, it is important to limit the number of downlink messages as much as possible. The fact that end-devices are at sleep most of the time to save battery, does require them to synchronize their sleep cycle to overcome the need of transmitting the same data multiple times to different devices. This extra complexity is also one of the main reasons why LoRa is not a popular protocol for FUOTA.

For these reasons, many companies have decided to move away from LoRa. However, for some of them, this is not an option as they have already implemented LoRa, or other protocols do simply not provide the specifications required. It has already been demonstrated to be possible within a small setup [32], leaving the question if it can be scaled-up to large production networks. This directly results in the motivation for this thesis.

1.3 Research Questions

Below the research question with the corresponding subquestions for this thesis are described. The subquestions needs to be answered before answering the main questions and play an important role in defining which experiments to perform.

1. **What is a reliable model to estimate the energy consumption of a firmware update?**

   (a) *What different energy consuming events occur during a firmware update?*
   The simplicity of LoRa makes it easy to estimate the power consumption based on different events. These events can for example be transmitting data, opening a receive slot, sleep mode and idle mode. Of course the energy consumption of these events depends on different configurations like data rate and message size. The expectation is that only transmission and reception time are worth measuring, as most of the microcontrollers used for LoRa are extremely low-power and almost consume nothing compared to the radio model.

(b) *What power model can be used to estimate the power for each event?* Different energy models are presented [25] [33] and chip manufacturers offer detailed power consumption sheets (for example the SX1276 radio module [11]). Is it possible to link these models to different event? Or in other words, is it possible to accurately estimate the power consumption based on these events? It is likely that the power consumption can easily be calculated based on the number of up and downlink messages, without having to actually measure the power. Most manufacturers actually provide detailed power consumption information that can be used to make these calculations.

2. **What is the increase in packet losses during a firmware update?** Packet losses increase the number of up and downlink messages that need to be sent, which directly increase the power consumption of the devices. When the power model in *Question 1* is found, the number of expected up and downlink messages (based on different parameters) can be used to estimate the total power consumption of an end-device during a firmware update.

   (a) *How many more FUOTA related packets need to be sent when scaling up the number of devices?* The devices that need to be updated receive multiple unicast messages from the gateway to setup the FUOTA process. What is the impact, in terms of FUOTA related packet losses, when increasing the number of to-be-updated devices. How many more packets need to be sent in order to successfully setup FUOTA for all devices?

   (b) *What is the increase in packet losses for devices in the network that are not being updated?*
   Some devices in the same network might not need to be updated, these devices should still be able to send data to the server. However, when other devices in that network are being updated at the same time, does this increase the packet losses of the other devices, and how does that relate to the transmitted firmware size and number of devices? It is likely that more packets are lost when increasing the number of devices that needs to be updated, especially for the downlink messages due to the duty cycle restriction.

   (c) *What is the increase in firmware packet losses when scaling up the number of devices?*
   When only a part of the network need to be updated, the other devices are still sending messages. Do these messages increase the number of lost firmware packets sent to the devices that are being updated? If so, this would require extra firmware packets to be sent, thus increasing the power consumption of the end-device.

1.4 Contribution

In this thesis, we are going to evaluate the efficiency and scalability of transmitting firmware updates over LoRa Class B on real hardware. As far as the author his knowledge, the power consumption of a firmware update over LoRa Class B and its scalability has not been evaluated using real hardware.

The following key contributions are made:

- Simple power consumption model to estimate the power consumption of an end-device.
- Evaluation of the scalability of FUOTA on real hardware with 100 end-devices.
- Evaluation of the impact on the network during a firmware update. For both the end-devices that are being updated and the end-devices that are not.
- Feasibility of the firmware update over LoRa based on a real live example.
- Small change to the LoRaWAN communication protocol to improve the power consumption of the end-devices.

During the evaluation, there will be no changes made in the standard proposed method to perform LoRa firmware updates [3], and predefined packages by the LoRa Alliance will be used for communication. Extra code for advance logging and to make class B on the end-devices work is built for the end-devices. Additionally, a FUOTA application server is built to schedule firmware updates at different times and gather the logs from the end-devices.

Chapter 2

Background and Related Work

2.1 Related Work

The number of papers available on LoRaWAN Firmware Updates is limited, especially with a LoRa Class B implementation. Ruckebusch et al. (2018) [25] presents the energy cost distribution between uplink messages, downlink messages and firmware installation. Although the energy cost for transmission is much higher than then the energy cost for reception and installation, the most energy is spent on receiving data. This indicated that the number of uplink messages is limited per firmware update. However they do not provide any information about the protocol used for the firmware update. While Ruckebusch et al. indicates the high energy consumption for RX, the research of Guinee (2019) [16] focuses mainly on reducing the uplink messages. They state that the uplink message is the most energy hungry. While this seem true, based on the numbers presented by Ruckebusch et al. and Abdelfadeel et al. (2020) [1], it can be discussed if focussing on the already limited number of uplink messages is worth the effort, especially in comparison with the large number of downlink messages.

The most important reason to use multicast for sending firmware updates is given in a simple example calculation in the introduction of [25], therein authors show that multicast is the only feasible option for firmware update for larger networks (e.g 100 nodes). Multicast can only work in Class B or Class C mode [21], therefore, [25] does not even discuss the possibility to use class A communication for firmware updates. They compare class B with class C and conclude that class B reduces the energy cost significantly compared to class C. Additionally, they conclude that the transmission time of a full firmware update increases by 17 % for class B communication. However, the huge energy saving does outweigh the transmission time increase. In addition, [25] showed that when using a "stupid" algorithm (with no form of scheduling) the time and energy usage increase linearly with the number of end-devices in the network. The biggest reason for this is the duty cycle limitation, which is limited to 1 % in Europe. The more end-devices in the network the more multicast group messages the base station has to send to each end-device. This results longer

wait times because the duty cycle limit is quickly reached. Therefore, for larger networks a better initial phase algorithm needs to be designed.

Not FUOTA related, Shiferaw et al. [29] studied the scalability of LoRaWAN Class B Multicast. They introduced the term Beacons-blocking where Class B beacons (time synchronization messages) are blocked by other messages. Additionally they showed that a higher data rate increases the throughput, but reduces the capacity. Lastly they showed that when using Class B alongside Class A, the conflicts between the two increases when the periodicity at which Class B messages are sent increase. For FUOTA it is possible to prioritize Class B over Class A, however this would result in packet losses for Class A devices and vica versa.

All the results on the papers above are based on simulations. Except for a FUOTA Class C demonstration by Stokking et al. no other actual results based on real hardware was found.

Research related to the power consumption of Class A end-devices has been done by Casals et al. [7] where they present a power consumption module based on measuring real hardware. A good overview of the power consumption for different events is given. For example the power consumption during transmission, reception and sleep. These results will later in this thesis be used to define the FUOTA power consumption model.

2.2 How To Perform FUOTA With LoRaWAN?

There is no standardized way to perform firmware updates over LoRa yet. The LoRa Alliance has published a document about FUOTA [3] where they describe different application layer packages that can be used to realize FUOTA. Based on this document, Semtech has created a test implementation using these packages [28]. Other than that, there are no guidelines on how to perform LoRa FUOTA.

2.2.1 LoRaWAN Packages

LoRaWAN packages are collections of LoRaWAN messages implementing a specific functionality. Examples of LoRaWAN packages defined by the LoRa Alliance are the *LoRaWAN Remote Multicast Setup* package [21], *LoRaWAN Fragment Data Block* [20] package and *LoRaWAN Application Layer Clock Synchronization* package [34]. The first two of these packages are used for FUOTA and will be described in more detail below.

2.2.2 Multicast

Theoretically it would be possible to send firmware updates to each end-device using unicast. However, if 100 devices need to be updated with the same firmware update, a lot of the same packets have to be sent 100 times to different devices. A much more convenient way would be using multicast, where a packet can be sent to all the devices at once. When a device operates in Class A mode, it is not possible for end-devices to transmit at the same time or the messages will collide. This means that no end-devices will open a receive window at the same time making multicast impossible. Therefore, multicast in LoRa is only possible in Class B or Class C mode [21]. Another advantage of using Class B

or Class C is that no uplink messages are required before a downlink message, this will result in a much more energy efficient implementation.

The LoRa-Alliance has described an application layer package called *LoRaWAN Remote Multicast Setup* [21] for remotely setting up a multicast session. During this session end-devices switch to Class B or Class C mode and firmware fragments can be sent to the end-devices using multicast. The session ends at a specified timeout to overcome that a end-device stays too long in Class B or Class C mode. Additionally, it is possible with this packet to configure multicast groups remotely. It is not required to configure a multicast group remotely, it is also possible to hard code the multicast configuration (like address and encryption keys) into the device. However, this would mean that a device cannot be removed or re-added from/to that multicast group.

2.2.3 Firmware Fragment Transmission

Firmware data needs to be sent in fragments to the end-devices because the maximal allowed payload size in Europe is 242 bytes (for a data rate higher than DR3) [8]. The problem with sending fragmented data using multicast is that each end-device might randomly lose some packets. For example 1000 devices lose 10 % of the packets. This means that one packet will be received by 900 devices. The server has to send the packet again to 100 devices where again 10 % loses the packet. When continuing, this results in that one packet is sent four times to be received by all the end-devices. To overcome this problem the LoRa-Alliance has introduced the *LoRaWAN Fragmented Data Block Transport* package [20] which uses forward error correction (FEC) to solve this problem. This technique is outside the scope of this thesis, however the Appendix of [20] gives a good explanation of a FEC technique that can be used.

The basic idea of the *LoRaWAN Fragmented Data Block Transport* package is to initialize a fragment data session. When setting up this session, the server tells each end-device how many fragments it can expect. Once all the data fragments are send, it asks each device to give a status on how many of the packets it has received. Base on there responses the server can send redundancy packets to fill the "gaps".

2.2.4 Class B versus Class C

The difference between Class B and Class C communication has already been described in Section 1.1.2. It is clear that class B is more energy efficient than class C. It can be argued that the extra effort of implementing class B outweighs the power savings. However, in Europe one device is only allowed to send 1 % of the time, if a class C implementation is chosen, 99 % of the time the receive window is open for nothing. Leaving the receive window unnecessary open would not be a big issue if that would not increase the power consumption. However, looking for example at the SX1276, the power consumption during a receive window is significantly more than when the chip is idle. Work of [1] supports this observation, the authors claim that using class B is 550 times more power efficient than using class C multicast. Another reason why class B is interesting to research, is because not many working class B FUOTA implementations are known. Creating a class B FUOTA implementation might benefit others, especially those who want to implement a more energy efficient FUOTA method.

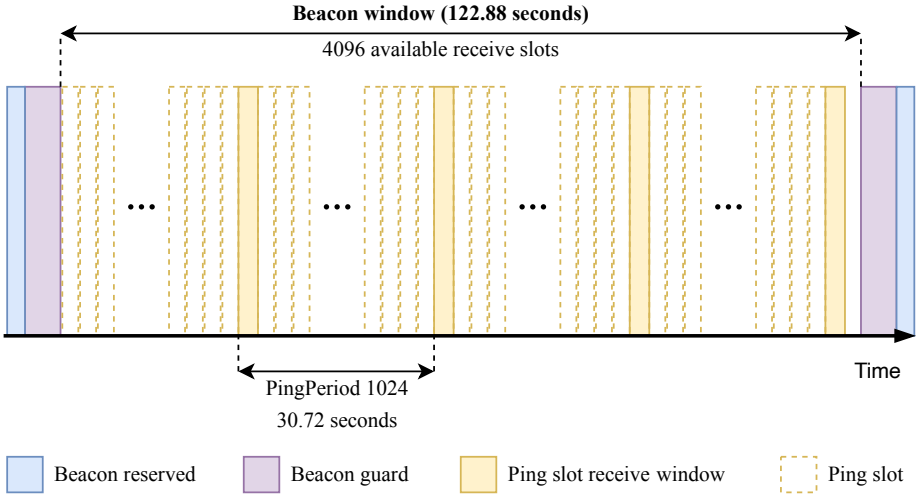


Figure 2.1: **Distribution of ping-slots between two LoRa beacons. In total 4096 ping slots can be assigned. In this example 4 ping-slots are used as a receive window.**

2.2.5 FUOTA Implementation Used for Evaluation

For evaluation, the final implementation uses Class B multicast to transmit the firmware data, and uses the *LoRaWAN Multicast Setup* package to setup the multicast group plus the necessary encryption keys. The *LoRaWAN Fragmented Data Block Transport* package is not going to be used because installing the firmware data on the devices is out of the scope of this thesis. To synchronize the time of the end-devices with the time of the gateway, the *DeviceTimeReq* and *DeviceTimeAns* messages are used which are part of the LoRaWAN specification [2].

2.3 LoRaWAN Class B

In class A only two receive windows are opened after each uplink message. The remaining time the device is in sleep mode and cannot receive any downlink message. For most sensor-based applications this is enough, however for sending lots of data to the end-device, for example a firmware update, a lot of uplink messages are needed in order to receive all the data. For battery-powered devices this causes a problem because sending uplink messages are generally power consuming. Class B offers the solution by opening more receive windows at a periodic interval without the need of an uplink message. These periodic receive windows are called ping-slots. To make sure that a downlink message is sent at the same time as an end-device opens a ping-slot, time synchronization between all devices and gateways is needed. This is done by the gateway by sending a beacon that the end device will use as a timing reference. In between two beacons, ping-slots can be scheduled at 4096 places. The fastest periodicity to open ping-slots is everyone seconds. Figure 2.1 illustrates one beacon period (the time between two beacons) with four of the 4096 ping-slots used as receive window.

2.3.1 Beacons

Beacons are scheduled ever 128 seconds since GPS epoch (i.e. January 6th 1980 at 00:00:00 UTC). To know when the beacons will arise, the end-device sends a *DeviceTimeReq* to the gateways. This message is part of the LoRaWAN specifications [2] and asks the gateway for current time. When a *DeviceTimeReq* is received by the gateway, it will respond with a *DeviceTimeAns* (also part of the LoRa specifications) containing the seconds since epoch. The end-device will synchronize its internal clock with the just received timing information, and can start opening a receive window at the time it expects a beacon. See Appendix A for more detail about the *DeviceTimeReq* and *DeviceTimeAns* message.

Before each beacon receive window a guard period of 3 s is introduced during which no ping-slots or uplink messages are allowed to be scheduled. This protects the beacon receive window from being overwritten by a large up or downlink message. For the beacon itself 2.12 s is reserved, and again no ping-slots or uplink messages can be scheduled during this time. That leaves 122.88 s in between two beacons (called the *beacon window*) during which ping-slots can be scheduled.

Although beacons are sent every 128 seconds, it might happen that end-devices lose some beacons due to movement of the devices or changes in the environment. If the timing of the end-device is accurate enough this should not be an issue and the end-device will find one eventually. To increase the chance of finding a beacon again, [2] states that the beacon receive window should be enlarged every time it misses a beacon (until a certain maximal receive window time). Furthermore according to the specifications [2], an end-device should be able to operate in class B for at least 2 hours without receiving a beacon. If still no beacon is received after that, it should go back to class A mode.

2.3.2 Ping-Slots

In between two beacons receive windows can be scheduled at 4096 ping-slots (see again Figure 2.1). Dividing the *beacon window* time (122.88 s) by the number of ping-slots results in a ping-slot duration of 30 ms. The number of ping-slots receive windows (PingNb) and the time in between them (PingPeriod) can be derived from the ping-slot periodicity (Periodicity) as

$$\text{PingNb} = 2^{7-\text{Periodicity}}\,,$$

$$\text{PingPeriod} = \frac{2^{12}}{\text{PingNb}}\,.$$

The PingPeriod is expressed in the number of ping-slots (of 30 ms each). For example in Figure 2.1 a Periodicity of 5 is used: $\text{PingNb} = 2^{7-5} = 4$.

To overcome downlink packet collisions between multiple devices with the same *Periodicity*, a *PingOffset* is added to the first ping-slot. The *PingOffset* is based on the beacon time and the *DevAddr* which should be unique for each device. This results in each device start opening their receive window at a different time.

2.3.3 Class B Beacon and Ping-Slot Blocking

Shiferaw et al. [29] discussed the scalability of multicast class B. One of the main issues in class B is *beacon blocking*. By default class B uses the 869.525 MHz frequency sub-band (G3) for both the uplink and downlink messages. In contrast to the other sub-bands, the G3 sub-band has a duty-cycle restriction of 10 % for the downlink channel (the uplink channel is still restricted to 1 %), although this duty-cycle is ten times higher than the other sub-bands, the Gateway cannot send downlink messages 90 % of the time. If the time-on-air of the pings oversteps the duty cycle restriction, beacon may not be sent due to the duty-cycle restriction, resulting in *beacon blocking*.

Another problem that can occur within class B is ping-slot blocking. Shiferaw et al. [29] mentioned these blocking problems when sending (large amounts of) data over class B. A gateway with one antenna can only send an uplink message or receive a downlink message, not both at the same time. For that reason, a ping-slot can be blocked when a non-FUOTA device (a device that is not being updated) sends an uplink message just before a ping-slot opens (devices that are being updated know when these slots opens, so they do not sent anything right before a ping-slot). The other way around, uplink messages from non-FUOTA might not be received by the gateway and downlink messages might not be send back to the non-FUOTA devices during a ping-slot.

Chapter 3

Experiments and Results

The main purpose of this chapter is to answer the research questions based on experiment results. These research questions mainly focus on the transmission of the data rather than the installation. Therefore, the experiments performed in this chapter will only implement the *LoRaWAN Remote Multicast Setup* package (see 2.2). Answers to research questions will be written in a bolder font with the corresponding research question. Additionally, this chapter contains in-depth information about the experiments and how the answers are found.

The first section gives a detailed description of the communication and states during the chosen implementation of FUOTA, followed by a section explaining the experimental setup. Section 3.3 defines the power consumption model used to estimate the additional power consumption of during a firmware update. This model can later be used to calculate the actual power consumption of an end-device during FUOTA. The actual experiments and the corresponding result to answer the research questions can be found in Section 3.4 and 3.5. This chapter ends with Section 3.6 where all the experiments and the power model are combined to evaluate the feasibility of firmware updates over LoRa using a real firmware example developed by TWTG [36].

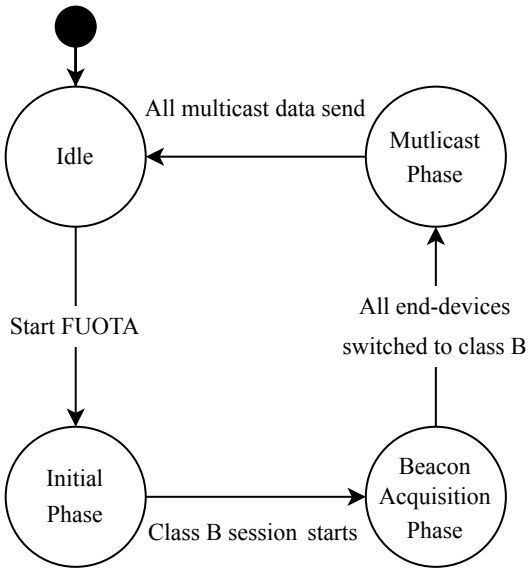


Figure 3.1: **State diagram of the different FUOTA phases for the implementation used for the experiment. A complete FUOTA would also include sending redundancy packets at the end of the Multicast Phase.**

3.1 Experiment Phases and Communication

In this section we introduce three different phases to make the complete FUOTA cycle more insightful and easier to reference. Section 2.2.5 already describes the techniques and LoRaWAN packages used for the FUOTA experiments. Figure 3.1 shows an overview of the different phases, which will be described in more detail below. The communication during the phases can be found in Figure 3.2. Notice that all the messages in italics are part of the Multicast Remote Setup Package [21], this includes *McGroupSetupReq/Ans* and *McClassBSessionReq/Ans*. The exact format of these messages can be found in Appendix A. Starting with the Initial Phase, where all the end-devices are prepared to go into multicast mode, four uplink messages are required. However, there is one additional uplink message sent which is not shown in the figure: *pingSlotInfoReq* to tell the server what ping-slot periodicity is used (see Appendix A). This uplink message is part of the LoRaWAN 1.0.3 specifications [2]. To sum up, for the used implementation, **the end-device needs to send 5 uplink messages before entering the multicast state**. Except for the *Fragment Data Block Transport* package, the implementation follows the LoRa-alliance FUOTA recommendations [3].

3.1.1 Initial Phase

Before the Initial Phase is started, the *McGroupSetupReq* needs to be sent to each device, this message ensures that all the devices are part of the same multicast group and uses the right encryption keys. The reason why this is not part of the Initial Phase is because it is also possible to hard code the multicast

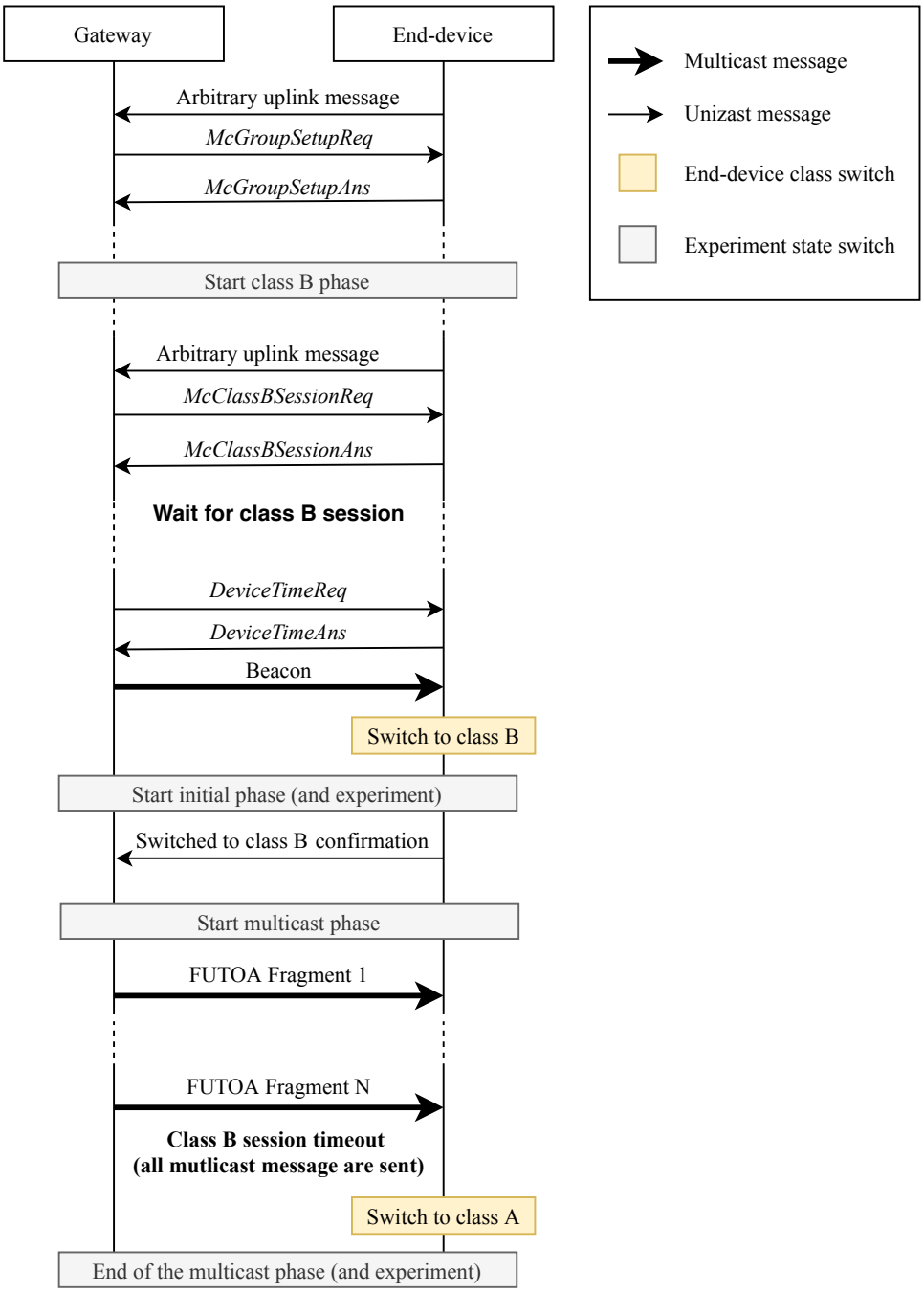


Figure 3.2: **LoRa communication sequence between an end-device and the gateway during an experiment. Logging uplink messages are not included. When in class A before an arbitrary uplink message is required before a downlink message can be sent from the gateway. The messages in italics are messages part of the Multicast Remote Setup Package (see Section 2.2.1). This figure shows only one device, however up to 100 devices will be used for the experiments. Each unicast message has to be sent one time for each end-device.**

settings into the end-device. Additionally, the *McGroupSetupReq* can be sent days before the actual firmware update if preferred.

The Initial Phase starts when the first *McClassBSessionReq* is sent to a device, during this phase all devices operate in class A mode. The *McClassBSessionReq* contains a time at which the end-device should start its class B session. All end-devices will receive the same session time, resulting in each device starting the class B session at the same time. Once the class B session timer goes off, the Initial Phase is completed and it moves to the Beacon Acquisition Phase.

Notice in Figure 3.2 that the end-device sends a *McClassBSessionAns* back after it receives a *McClassBSessionReq*, this message is only to inform the Application Server that is received the session setup request and that it has started its timer.

The time each device spends in the Initial Phase is not important in terms of power consumption, however the number of extra uplink messages are. This is because the devices stay in class A and operates as normal, only the extra uplink messages needed for FUOTA add requires extra energy.

3.1.2 Beacon Acquisition Phase

This phase starts when the class B session starts. Each device will search for a beacon, once received it will inform the Application server that it has successfully switched to class B. Once all devices are switched to class B the next Multicast Phase will start.

In contrast to the Initial Phase, the Beacon Acquisition Phase does add additional power consumption because end-devices will switch to class B which has a higher power consumption than the regular class A.

3.1.3 Multicast Phase

During the Multicast Phase, each end-device operates in class B mode and the Experiment Server will start sending the FUOTA multicast messages. Earlier during the Initial Phase, not only a session start time has been sent to the devices, also a session end time (timeout) has been sent. Once the session ends, all end-devices switch back to class A and the Multicast Phase ends. The session timeout is calculated based on the number of multicast messages that needs to be sent to the end-devices.

3.2 FUOTA Experimental Setup

For the experiments, a simple setup is built with one gateway and multiple end-devices. Even though the setup is simple, it simulates a real-time scenario in terms of LoRa network. Figure 3.3 illustrates the setup with one gateway. A maximal of $N = 100$ end-devices are connected to the gateway. More detailed information about each block can be found in the following sections.

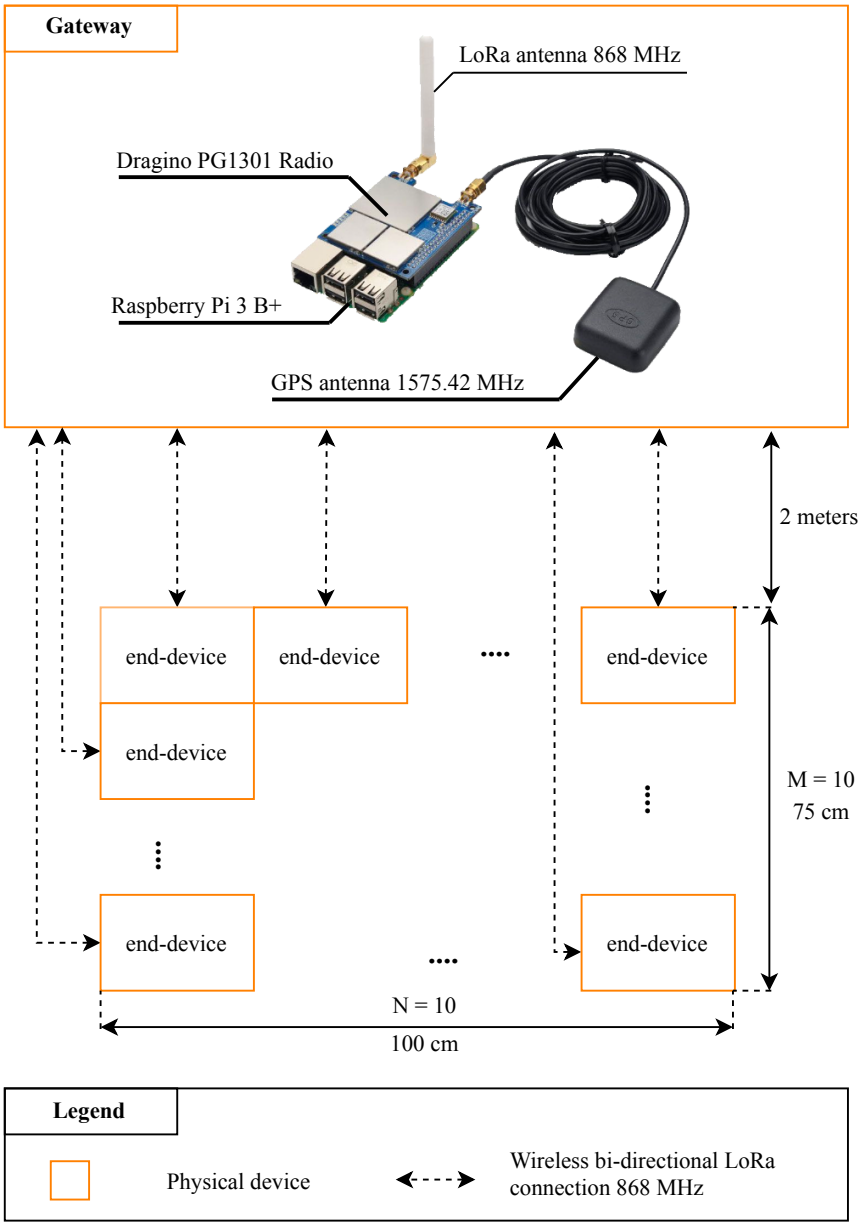


Figure 3.3: **Experiment setup with a total number of $M \times N = 100$ devices. There is no space in between the end-devices. The standalone gateway does not require additional connections during an experiment. To readout the experiment data, an Ethernet connection can be established with the Raspberry Pi. See Section 3.2.2 and Section 3.2.1 for more detail about the gateway and end-devices.**

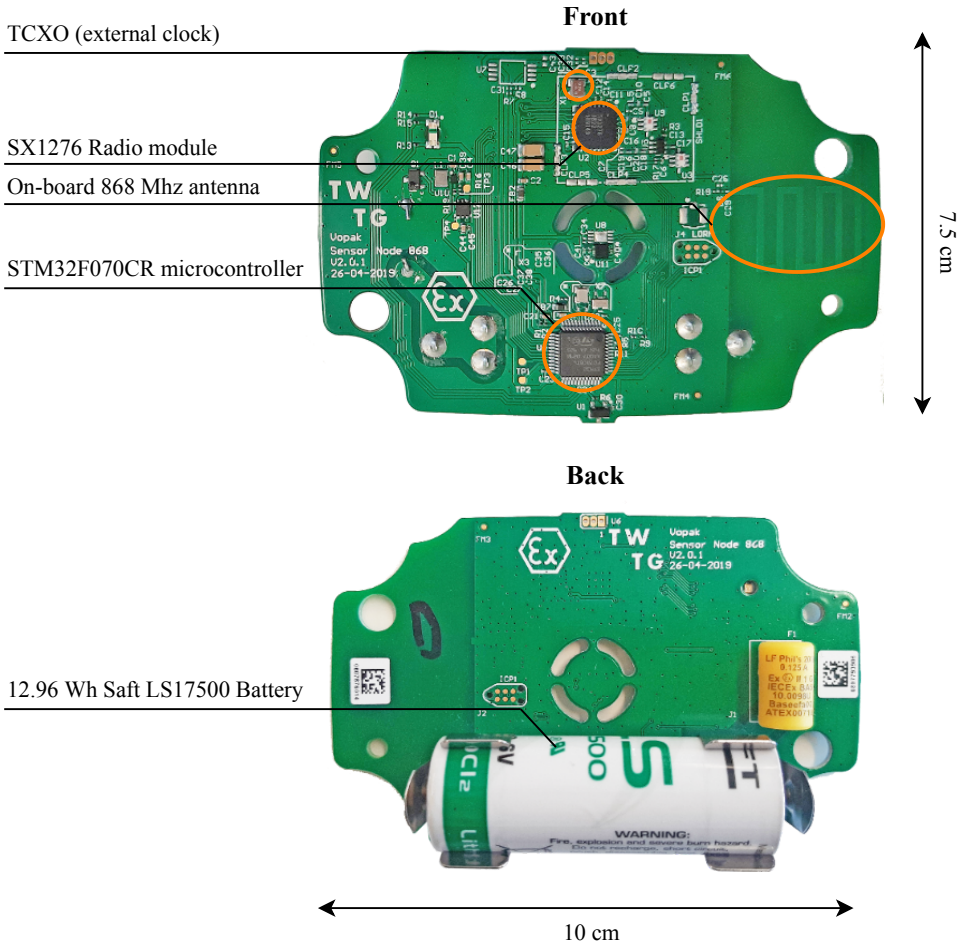


Figure 3.4: **The font and backside of the end-device developed by TWTG [36] and used for the experiments. The end-device is powered by a** 12.96 W h **Saft LS17500 battery [26].**

3.2.1 End-Devices

Hardware

The hardware for the end-device is developed by TWTG [36] and consists of the STM32F070CR microcontroller [31] in combination with the SX1276 radio module [11]. For accurate timing a TCXO[17] is used. Figure 3.4 shows the dimension of the end devices and the location of the components on the PCB.

Software

The software is based on the LoRaMac-node example by Semtech [28]. Additions are made to make class B multicast working, to add logging, to make it work on the STM32F070CR microcontroller, and to schedule experiments.

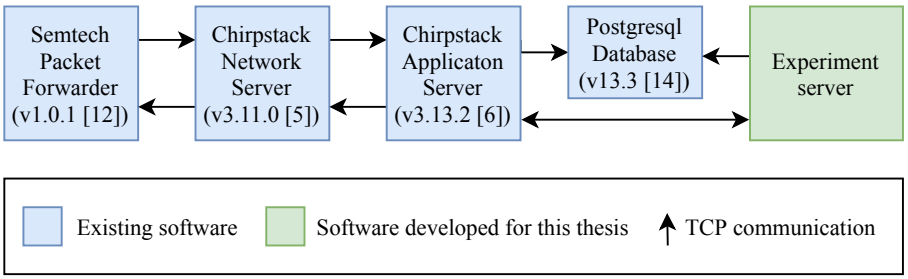


Figure 3.5: **Overview of the software running on the Raspberry Pi 3 B+.**

3.2.2 Gateway

The gateway is a standalone gateway, which means that no external connections are needed to the gateway. Only to read out the experimental data, the gateway can be connected to the Internet using an Ethernet cable.

Hardware

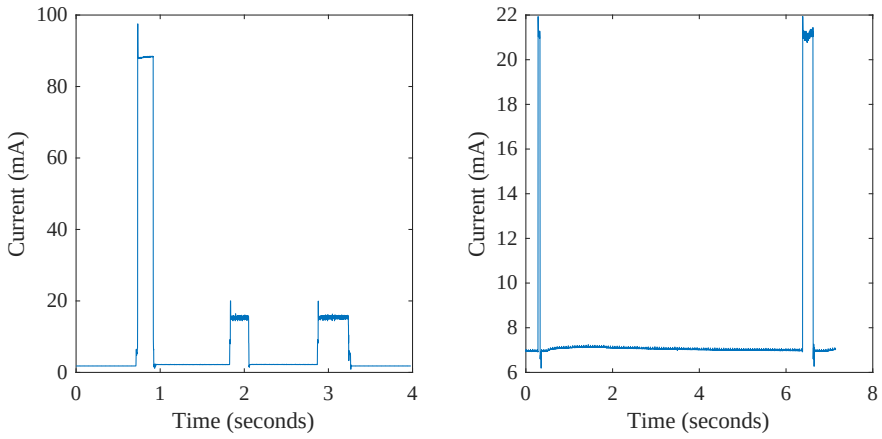
The gateway consists of a Raspberry Pi 3 B+ [14] in combination with the Dragino PG1301 LoRa transmitter/receiver [12]. The PG1301 has a built-in GPS receiver for accurate timing. An SPI connection between the Raspberry Pi and the PG1301 is used to communicate with the radio module of the PG1301, and a UART connection is used to readout the GPS data.

Software

To make the gateway standalone, the whole software stack is located on the Raspberry Pi. This includes the gateway Packet Forwarder, Network Server and the Application Server (see Section 1.1.2). A modified version of the Semtech Packet Forwarder [13] is used for the packet forwarder. This piece of software is responsible for forwarding all the LoRaWAN packets between the end-devices and the network server. Additionally, it is responsible for scheduling the Class B beacons and ping-slots. The Packet Forwarder is the only software communication communicating with the radio module. Chirpstack is used for the Application Server [5] and the Network Server [6]. The communication between these software components is done using an internal TCP connection.

An additional piece of software is written to control the experiment. This software tells the Application Server which (multicast) message it has to send during a FUOTA experiment. It controls which end-device is part of the experiment and which should go to sleep. Additionally, it keeps track of the state of each end-device and stores this information in a Postgresql database [15]. This is also where the Application Server stores all the received uplink messages from each end-device.

An overview of these software components running on the Raspberry Pi can be found in Figure 3.5.



(a) **Uplink message with two corresponding receive windows**

(b) **Empty ping-slot (left peek) and beacon reception (right peek).**

Figure 3.6: **Snapshot of the power consumption of the STM32F070CB microcontroller [31] in combination with the SX1276 radio module [11] during different events.**

3.3 Power Consumption of FUOTA over LoRa

Let us start with answering research question *1.a* and *1.b* in Section 1.3, the individual components that draw power need to be investigated. In the case of FUOTA there are a few components that draw power: the radio module, microcontroller and (external) memory to store the firmware update. Additional sensors, ADC or LEDs are not used during a firmware update so they do not need to be considered.

Storing data on external flash or in the EEPROM does also require extra power, however in this study this will not be considered because the firmware data is not stored and installed on the device. Comparing it with the radio module, storing data typically does not add much to the overall power consumption.

Shnayder et al. [30] considered simulating the power consumption for large-scale sensor networks. This paper confirms the different components that draw power from the battery. They provided a table with the power consumption for each component in different modes. Such a table will also be provided for the hardware used for this experiment.

3.3.1 Power Consumption Microcontroller Model

During FUOTA, the microcontroller used for experiments (STM32F070CB [31]) operates in two modes, sleeping mode and operating mode. When receiving and transmitting data, the microcontroller runs in operation mode, all the other time it is in sleep mode. After all the firmware data is received, the installation process will start. The power consumption during installation will not be added to the power consumption model because it is out of the scope of this thesis.

For simple microcontrollers such as the STM32F070CB, power consumption during different modes is almost constant. Especially in sleep mode where gener-

ally only the clock is activated. In operation mode, the power consumption does depend on the instructions executed by the microcontroller, however as long as these instructions are not memory access instructions, the difference between them is negligible [30]. Therefore the power consumption during operation mode can be seen as constant.

Based on two different modes we define de power consumption of the microcontroller as

$$E_{\text{mic}} = T_{\text{sleep}} P_{\text{sleep}} + T_{\text{op}} P_{\text{op}} \,,$$

where T_{sleep} and T_{op} are the time spent in sleep mode and operation mode respectively, and P_{sleep} and P_{op} are the power during sleep mode and operation mode, respectively. P_{sleep} and P_{op} can be found in the datasheet of the used microcontroller.

When the microcontroller exits the sleep mode, it needs a few milliseconds of processing before it can start transmitting or receiving, and afterwards it needs a few milliseconds to enter the sleep mode. These processing times will be added to the power consumption model. We define T_{op} as

$$T_{op} = T_{\text{rx}} + T_r + T_t + T_p \,,$$

where T_{rx} is the total time of the receive windows opened during the firmware update (both class A receive windows and class B ping-slots). When data is received T_r adds the extra time needed to receive the data. T_t is the total time transmitting data, and T_p is the processing time before and after each transmission and reception. When operating at a frequency of 32 MHz, for the STM32F070 T_p is on average 18 ms for each transmission and 12 ms for each time the microcontroller opens the receive window.

Next, we define T_{sleep} as

$$T_{\text{sleep}} = T_{\text{FUOTA}} - T_{\text{op}}$$

where T_{FUTOA} is the total time of a firmware update, including the Initial Phase and Beacon Acquisition Phase. T_{op} is again the total time the microcontroller spent in operation mode.

Mode	Condition	Power consumption	Unit
Sleep		0.2 – 1	µA
Idle		1.5	µA
Receive	LnaBoost Off, band 1	10.8	mA
	LnaBoost On, band 1	11.5	
	Bands 2 and 3	12.0	
Transmit	Power 20 dBm	120	mA
	Power 17 dBm	87	
	Power 13 dBm	29	
	Power 7 dBm	20	

Table 3.1: **SX1276 power consumption during different modes. LnaBoost is the low noise amplifier which can be turned on or off [11].**

3.3.2 Power Consumption Radio Module Model

According to the SX1276 datasheet [11], the radio module can operate in different modes (see Table 3.1). The Standby mode and the Synthesizer mode of the SX1276 shown in the datasheet are not used and therefore not included in Table 3.1.

The most power consuming modes, as expected, are the receive and transmit mode. The idle and sleep modes both consume less than 2 µA. Comparing this with the low power mode of the microcontroller, it adds less than 1 % to total power consumption. Therefore to reduce the complexity of the model, power consumption during this mode is not added to the model.

Based on these specifications, the power consumption of the radio module can be modeled as

$$E_{\text{radio}} = T_t P_t + T_r P_r \,,$$

where P_t and P_r are the power for transmitting and receiving respectively. T_t and T_r are the time needed for the transmission and reception, respectively.

There are a lot of factors that influence the value of T_t/T_r and P_t/P_r such as transmission power (SX1276 has four transmission power modes, see Table 3.1), data rate, bandwidth and code rate. Accoriding to [4], there are 6720 possible transmission configurations.

The transmission power directly influences P_t/P_r. Although the SX1276 can transmit with a power up to 20 dBm, the maximal allowed uplink transmit power in Europe is limited to 16 dBm [8]. This means that only two power modes of this specific chip can be used. Other radio modules, however, might allow a larger range of transmission powers. In practice the transmission power of each device is programmed to be fixed. The reason for this is that the power consumption can also be manipulated by reducing the data rate and thus reducing the T_t and T_r. While the LoRaWAN specifications do not specify any communication to support adaptive power transmission, it does implement Adaptive Data Rate (ADR) communication.

More parameters are involved in defining T_t and T_r: the SF, bandwidth, message size and code rate. The code rate is most of the time fixed for the same reason as the fixed power transmission. The bandwidth and SF on the other hand depends on the data rate. This leave us with only the message size and data rate as non-fixed variables, reducing the number of possible configuration

Radio		Microcontroller	
Mode	**Power** (mW)	**Mode**	**Power** (mW)
Transmit	87	Sleep + TCXO on	5.34
Receive	34.5	Operating + TCXO on	21.15

Table 3.2: **Power consumption of the radio module (SX1276) and the microcontroller (STM32F070CB) during different modes. The TCXO (highly accurate external clock) is always on. The end-device operates at a voltage of** 3 V

significantly.

The time needed for transmission can be calculated as [10, Section 4]

$$R_b = SF \times \frac{\mathrm{CR} \times \mathrm{BW}}{2^{\mathrm{SF}}} \mathrm{bits/s}\,,$$

where CR is the code rate, BW the bandwidth and SF the spreading factor.

With the bit rate defined, the actual transmit time can be calculated by

$$T_t = 8n \times \frac{1}{R_b} = 8n \times \frac{2^{\mathrm{SF}}}{\mathrm{SF} \times \mathrm{CR} \times \mathrm{BW}}\,,$$

where $n > 0$ is the number of bytes to transmit.

For T_r almost the same model can be used, however it is highly likely that the receive window is opened before receiving data. If a receive window would be 30 ms, in the worst case the total T_r equals the 30 ms + T_r' where T_r' is the actual time it takes to receive a message. Additionally it is also possible that no data is received at all. In that case T_r equals the duration of the receive window (T_{rx}). The model for reception time can be defined as follows:

$$T_r = T_{\mathrm{rx}} + 8n \times \frac{2^{\mathrm{SF}}}{\mathrm{SF} \times \mathrm{CR} \times \mathrm{BW}}$$

where $n \geq 0$ is the number of bytes received. Notice that when n equals zero $T_r = T_{\mathrm{rx}}$.

3.3.3 FUOTA Power Consumption Model

Now that the energy consuming components are constructed, a complete energy consumption model can be defined. Table 3.2 shows the power usage of the microcontroller and the radio module during the discussed modes. The power consumption of the microcontroller is measured multiple times by using a device named Otii [23]. Table 3.2 shows the average of these measurements. The power usage of the radio module is based on the datasheet and verified by the measurements performed on the microcontroller.

The sizes of each message required during FUOTA are shown in Table 3.3. Multiplying the time on the air of each message (calculated based on the message size and data rate) with the power consumptions in Table 3.2 results in the **additional** power consumption. This means the extra power needed for a firmware update. It is the additional power consumption because the time it is not sending or transmitting FUOTA related messages it is doing the same as

Direction	Message	Size (bytes)
Downlink	McClassBSessionReq	26
Uplink	McClassBSessionAns	22
Uplink	PingSlotInfoReq	19
Downlink	PingSlotInfoAns	16
Uplink	DeviceTimeReq	18
Downlink	DeviceTimeAns	21
Uplink	Class B confirmation (empty message)	18
Downlink	Beacon	17
Downlink	Multicast firmware message	16 + payload size

Table 3.3: **Uplink and downlink messages sent during a FUOTA cycle and the corresponding sizes.**

Data rate	Initial Phase Energy (mJ)	Beacon Acquisition Phase Energy (mJ)	Multicast Phase Energy (mJ)
DR3	12.8	28.3	1915.0
DR4	7.9	18.1	1077.9
DR5	5.3	12.5	616.7
DR6	3.5	8.8	309.2

Table 3.4: **Power consumption of an end-device for each phase based on a firmware update of** 10 kB**. This is in the ideal situation when no packets are lost and all devices are switched to Class B at the same time.**

what it would do without a firmware update. This is without taking the energy needed to store and install the firmware update into account.

Table 3.4 shows the additional power consumption for an update of 10 kB for each phase. This table shows that, although an uplink message requires more power, the downlink multicast messages are far more energy consuming due to the large number of messages.

3.4 Scalability of FUOTA

Before sending the firmware update, each device needs to receive a series of unicast messages. Scaling up the number of devices will highly likely result in more packet losses and thus the need to send more uplink messages. Section 3.1 explains that a total number of five uplink messages is needed before stating the multicast phase. The first regular uplink message is not counted by the system, therefore **the total expected additional uplink messages is four**.

When sending a large amount of data, receiving the firmware data consumes the most power. During the Beacon Acquisition Phase devices partly operate in class B mode which additionally opens receive windows periodically. The problem in this phase is that not all the devices switch to class B at the same time due to possible beacon misses or uplink/downlink message collisions/blocking. Blocking happens when the gateway has already scheduled a downlink message during that time, like a beacon or a ping-slot downlink message. When for example devices *I* is switch to class B and devices *II* is not, device *I* spends

Data rate	DR3
Code rate	5/4
Firmware fragments	50
Firmware fragment size	50 bytes
Ping-slot periodicity	4 (every 16 seconds)
End-device distance to Gateway	<2 meters

Table 3.5: **Configuration used in experiments to evaluate the scalability of FUOTA. These values are fixed through the entire experiment.**

more time in class B than device *II*, resulting in a higher power consumption for device *I*. Remember that multicast messages can only be send to the end-devices once all end-devices are switch to class B, therefore the longer it takes for a devices to switch to class B, the more energy other devices might use.

To evaluate the scalability of FUOTA multiple experiments are executed. In these experiments a firmware update cycle, without the actual installation, is performed with a variable number of devices. Each device logs their uplink and downlink messages and sends it to the server. Based on these logs, the possible increase in uplink messages can be determined when the number of devices increases. Additionally, it should tell us the duration of the Beacon Acquisition Phase.

The parameter configuration used in the experiments to evaluate the scalability are shown in Table 3.5. The experiments will be executed with $N = \{15, 30, 60, 100\}$ end-devices. Each experiment is repeating twelve times distributed over different day parts (morning, afternoon, evening, night). Additionally, at least four of the experiment are performed in the weekends.

The gateway is configured to prioritize Class B over Class A, this means that the reception or transmission of a class A message can be interrupted by a scheduled class B downlink message. Therefore, ping-slots will not be blocked, and the number of multicast messages received by an end-device should not decrease when scaling up the number of end-devices. The to-be-updated devices should not experience any up/downlink message blocking as they know when these ping-slots happen and can schedule their up/downlink messages around it.

3.4.1 Number of FUOTA setup uplink messages

Figure 3.7 shows a plot of the number of FUTOA setup uplink messages needed during the Initial Phase. The average number of uplink messages lies close to the minimal required uplink message of four, this means that most of the devices only send the minimal required number of uplink messages. However, by increasing the number of devices, the maximal uplink messages sent by a end-device increases. Although there are only a few end-devices that have to send a lot more uplink messages, the number of extra uplink messages needed for these devices is quite significant. When using 100 devices, at least one device had to send 11 uplink messages during the Initial Phase resulting in a significant power consumption increase during the Initial Phase.

Each device sends a message every two minutes with a random deviation of 5 seconds. It is possible that one of the end-devices is constantly blocked by the

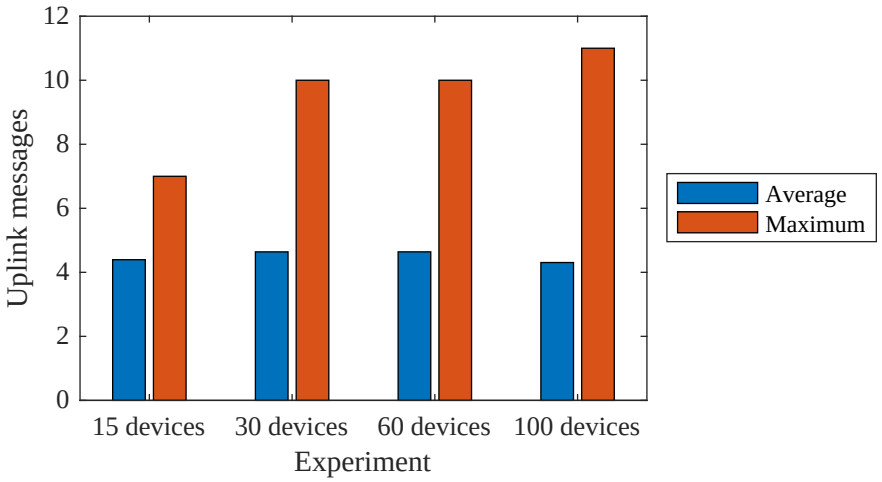


Figure 3.7: **Maximal and average uplink FUOTA setup messages needed by the end-device during the Initial Phase. On average no significant increase can be seen, however some devices need to send much more uplink messages during the initial state, resulting in a significant increase in the power consumption of these devices.**

same end-device due to the small deviation. A larger deviation might reduce the maximal uplink messages needed for that specific end-device.

To answer **Research Question 2a**: **on average the number of uplink messages stays almost the same when scaling up the number of devices, however the maximum needed uplink messages increases when the number of end-devices increases resulting in some end-devices consuming more power than others.**

3.4.2 Number of received multicast messages during FUTOA

During the FUOTA cycle, 50 multicast messages are send to all the end-devices. Figure 3.8 shows the number of received multicast messages on average during the four experiments. No correlation can be seen between the received multicast messages and the number of devices used. This is not unexpected because all the devices in the network operate in Class B mode, therefore they know when a ping-slot is scheduled and schedule there uplink messages around it.

To test that the gateway indeed prioritizes Class B over Class A, 60 non-FUOTA devices (devices that are not being updated) are added to the network. **The average number of received multicast messages when adding** 60 **non-FUOTA devices to the network is** 46.6**. This shows that the non-FUTOA devices do not interfere with the firmware update. (Research Question 2a)** However, this means that it is likely that the non-FUOTA devices will lose packets during the firmware update. The next experiment will test this hypothesis.

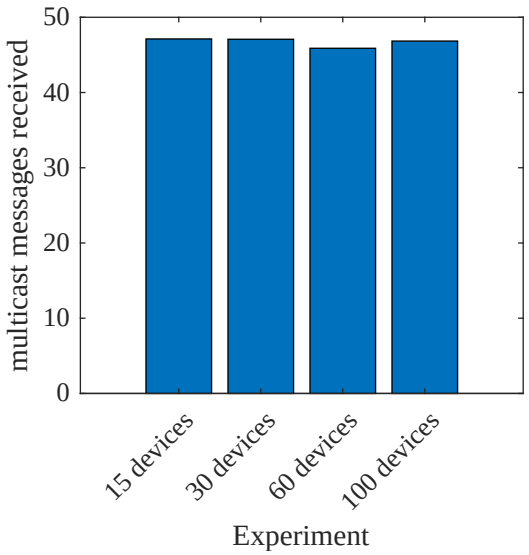


Figure 3.8: **Average received multicast messages per experiment. In total 50 multicast messages are sent. No clear correlation can be found between the number of multicast messages received and the number of devices.**

3.4.3 Beacon Acquisition Phase Duration

The longer an end-device stays in the Beacon Acquisition Phase the more additional power it uses. Therefore it is important to switch to class B as fast as possible, decreasing the time it spends in the Beacon Acquisition Phase. Figure 3.9 shows the average and maximal time an end-device spends in class B during Beacon Acquisition Phase. More devices in the network results in longer time spent in class B mode.

To understand what is happening here, it is important to repeat that the *PingSlotReq* and *DeviceTimeReq* messages are send by each end-devices at the start of the Beacon Acquisition Phase. Only when it received a valid answer to both messages and found a beacon, it sends a class B confirmation message to the server. The previous experiment (Section 3.5) showed that uplink message collisions are (on average) not happening often, even with 100 devices. After analysing the logs of the gateway, it seems that the duty cycle restriction and downlink packet blocking causes the problem. When 100 devices all need to receive two downlink messages at the same time, it takes quite a while before the gateway can actually send these messages to all of them because it is only allowed to send 1 % of the time.

To compare the Beacon Acquisition Phase in terms of power consumption, when using 100 devices, it needs 16 additional beacon periods before it can start the multicast phase. When a periodicity of 4 is used, this would add a power consumption $16 \times 9.3 = 148.8$ mJ to the end-device that has switched to class B as first. This is a significant amount comparing it with the power consumption of the Initial Phase 3.4

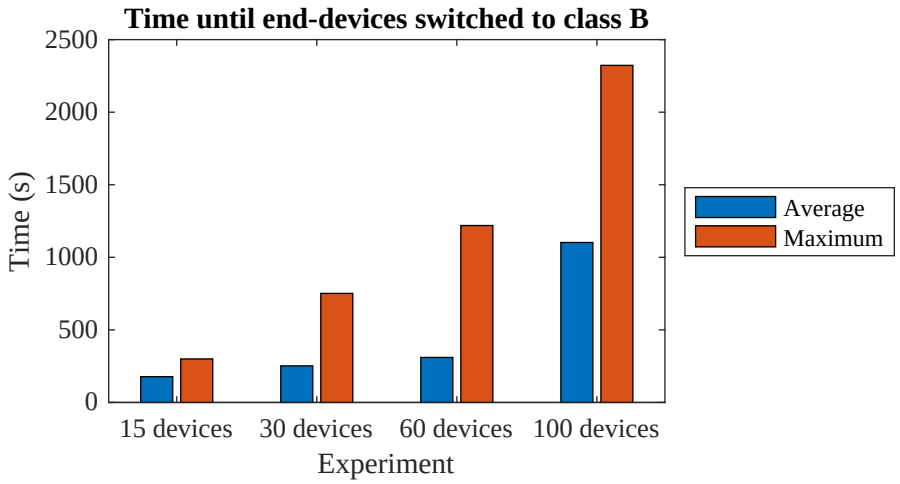


Figure 3.9: **Average time between first and last devices switched to class B. There is an extreme increase in time compared to the number of devices. Most likely due to the duty cycle restriction on the gateway side, preventing *DeviceTimeAns* downlink messages.**

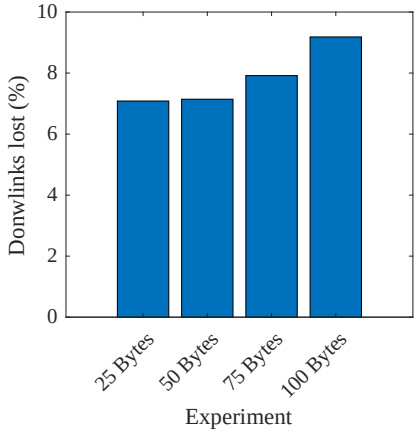
Data rate	DR3
Code rate	5/4
Number of end-devices	10
End-device distance to Gateway	<2 meters

Table 3.6: **Parameter configuration used for experiments to evaluate the impact on Non-FUOTA devices.**

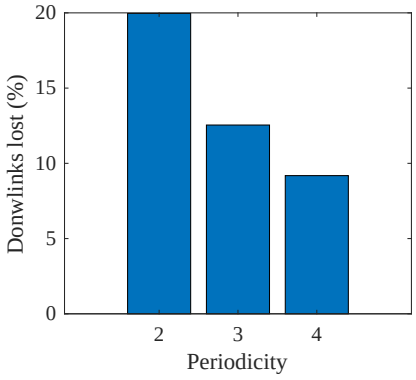
3.5 Impact on Non-FUOTA Devices for Different Firmware Sizes and Ping-Slot Periodicities

Within an existing network it is possible that not all the devices have to be updated. The devices that do not need to be updated, called the non-FUOTA device, continue their normal tasks by sending uplink and receiving downlink messages once in a while. However, during a FUOTA, the uplink and downlink messages of these non-FUOTA devices might be blocked due to the transmission of multicast messages to the FUOTA devices (see 2.3.3), especially because the gateway prioritizes Class B over Class A. The duty cycle restriction on the other hand is not expected to be a problem, because class A devices operate on a different sub-band than class B devices.

To evaluate the impact on non-FUOTA devices, one non-FUOTA device is sending uplink messages every 1 minutes, after which the gateway responds with a downlink message. The total uplink messages, and total downlink messages are counted to see the number of lost packets. The experiments are performed with the parameters configured as shown in Table 3.6. The impact of different ping-slot periodicity on the non-acfuota devices is evaluated by executing four



(a) **Fixed periodicity of 4.**



(b) **Fixed firmware fragment size of** 100 B**.**

Figure 3.10: **Downlink message losses for non-FUOTA device during a firmware update. The higher the firmware fragments transmission time, the higher the packet losses. Sending more bytes in one fragment result in better performance than increasing the periodicity, due to the extra LoRaWAN header bytes per packet.**

experiments of 1.5 hour with different ping-slot periodicities of $p = \{2, 3, 4\}$, and a firmware fragment of 100 bytes. In the same way the impact of the firmware size is evaluated, only with a fixed periodicity of 4 and a variable firmware size of $s = \{25, 50, 75, 100\}$ bytes.

3.5.1 Results

Figure 3.10 shows the results of the experiments. It can clearly be seen that the non-FUOTA devices suffer in terms of downlink packet losses during a firmware update. When sending firmware data close to the duty cycle restriction, periodicity 2 with 100 B per fragment, the packet loss is almost 20 %.

Each downlink multicast packet contains an additional 16 B for the LoRaWAN header (see Tabel 3.3). This means that in general, and also according to the data in Figure 3.10, it is better to send large messages with a higher periodicity.

To answer **Research Question 2b**, **there is high downlink packet loss of almost** 20 % **for non-FUOTA devices when transmitting firmware with a duty cycle close to the duty cycle restriction of** 10 %**, reducing the firmware transmission duty cycle also reduces the packet losses.**

This also leads to the recommendation to use a dedicated gateway for firmware updates. This means that non-FUOTA devices can still be served as normal by the current gateway while the firmware update is handled by another gateway. As long as the non-FUOTA devices do not use the Class B sub-frequency band with a center frequency of 869.525 MHz, there will be no up/downlink message collisions.

Another possibility with a dedicated gateway is to make it mobile. Using a mobile gateway makes it possible to update sections of the network by moving the gateway close to that section. When the gateway is closer to the end-

Version	Type	**Description**	**Image size (kB)**	**Delta image size (kB)**
1.0.2	Full image update	Base image	120	-
2.0.1	Delta update (1.0.2 ->2.0.1)	Large update. Main changes: - Communication protocol - Code restructuring	119	69
2.0.2	Delta update (2.0.1 ->2.0.2)	Small security update with bug fixes	119	7.1

Table 3.7: **TWTG [36] valve sensor firmware version and the corresponding images size and delta image size. To install version 1.0.2 the complete image has to be sent, where only the delta images need to be sent for version 2.0.1 and 2.0.2.**

devices, a higher data rate can be used resulting in less power consumption for each end-device. Although this seems to be a good improvement theoretically, more research should be done to validate this in practice.

3.6 Feasibility of FUOTA over LoRa

The firmware update size depends on what needs to be updated. [24] And [25] distinguish firmware update in three levels:

1. Full system update, where everything including the operating system can be updated.
2. Application Update.
3. MAC update, to update the part responsible for the radio communication.

All three levels can be updated using firmware-based update [25], where the complete image can be replaced. This can be done by sending a complete new image to the end-device, however when only a small part of the code is changed, sending a complete image is an overkill. Better is to send a so-called delta update (or patch) [18] where only the difference between the old and the new images will be sent. Even though delta updates are already much smaller than a complete image update, it can still get pretty large, especially when code is shifted to another place in the memory. This can happen when new code is added that does not fit in the current memory arrangement, then parts of the code need to be rearranged resulting in lots of reference changes and thus a larger delta image.

At the application and MAC level, more sophisticated methods like dynamic linking can be used. New applications can be added to memory at run time, and the symbol lookup table containing all the unresolved references can be updated.

The evaluation of the feasibility of FUOTA is based on software developed by TWTG for valve sensors [35]. The software is compiled with Mbed OS [22]. Mbed OS compiles everything into a single image making it only possible to use firmware-based updates. Therefore, other firmware updated techniques will not be discussed in this thesis. Table 3.7 shows three different updates. The first one (version 1.0.2) is a full image update, the complete 120 kB needs to be

Data rate	Maximal payload size (Bytes)
DR0 - DR2	51
DR3	115
DR4 - DR6	242

Table 3.8: **Maximal payload size in bytes per data rate in Europe [8].**

sent. The second and the last updates are delta updates where only the delta image (difference between installed images and updated images) needs to be sent. From version 1.0.2 to version 2.0.1 is a relatively large firmware update where the code has been restructured. The update from version 2.0.1 to version 2.0.2 is much smaller as only a few bugs are fixed.

The energy used by each end-device can be calculated using the power model presented in Section 3.3. To get a realistic calculation of the power consumption the ideal calculation of Section 3.3 needs to be modified with the results of Section 3.4. This means that when using 100 devices on average an end-device spends 1072 s in class B mode without receiving any downlink message, which adds (when using periodicity 4) a total of 67 empty ping-slot windows to the calculation. The uplink messages on the other hand does not seem to change on average so they will be kept at a total of 4. Next, the number of multicast messages depends on the maximal allowed payload size shown in Table 3.8 and the firmware (delta) image size. Each multicast message will use this maximal allowed payload size to minimize the LoRaWAN header overhead. Lastly we consider a packet loss of 10 %. This means that when a total of 100 packets are sent, 10 extra redundancy packets need to be sent. Of these 10 packets again 10 % will get lost, so another redundancy packet needs to be send, resulting in a total of 111 packets that needs to be sent.

Figure 3.11 on the next page shows the energy consumption for each firmware update in Table 3.7 when using seven different data rates. Comparing data rate DR0 with DR6 when sending a firmware update of 120 kB, the difference is 337 J. DR0 used 167 times more power than DR6, considering the same packet loss.

The battery used for the end-devices has a capacity of 12.96 W h which is 46 656 J. Even when using DR0 the additional power consumption for the firmware update requires 0.73 % of the battery. *This makes firmware updates over LoRa in terms of power consumption feasible.*

In Section 3.5 we recommended a dedicated mobile FUOTA gateway. This recommendation got even stronger based on the results of Figure 3.11. When using a mobile gateway it is possible to update end-devices in clusters close to the gateway. This makes it possible to use higher data rates and significantly reduces the power consumption of an end-device during FUOTA.

3.7 Extend the LoRaWAN Fragment Data Block Transport Package

In Section 2.2 a way of implementing FUOTA is described. No changes were made to the LoRaWAN packages used. However, two major problem where discovered in Chapter 3 when using this implementation: (1) end-devices that

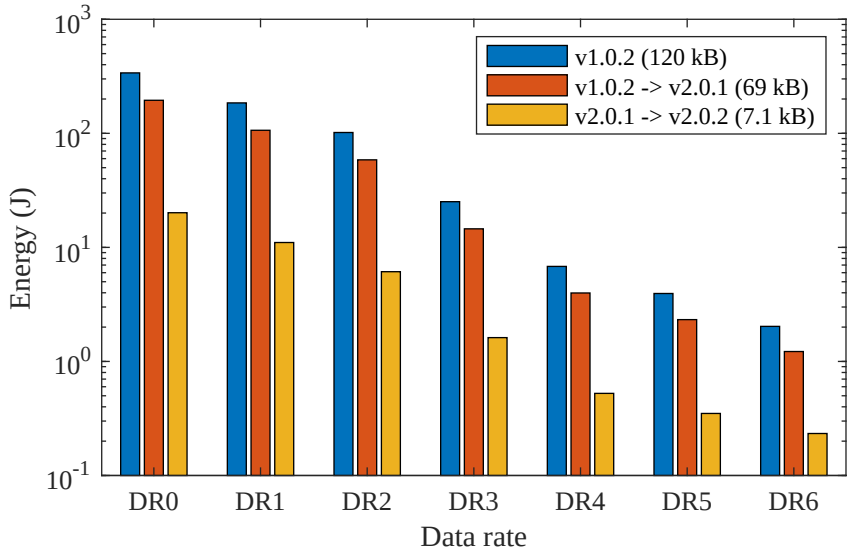


Figure 3.11: **Average FUOTA power consumption for three different firmware updates, based on a periodicity of** 4**, a code rate of** 5/4**, and a packet loss of** 10 %**. A logarithmic scale on the y axis is used.**

will not be updated suffer from downlink message blocking during FUOTA (2) the time until each end-device is switched to class B increases with the number of end-devices, resulting in higher power consumption. In Section 3.5 and Section 3.6 we recommended using a dedicated gateway to solve this problem. For the second problem we propose a modification to the *LoRaWAN Fragment Data Block Transport* package as described below.

As described in 2.2 the *LoRaWAN Fragment Data Block Transport* package manages the transmission of data fragments and the sending redundancy packets. To setup a data fragments session, the server send a *FragSessionSetupReq* to each end-device. This message specifies how many fragments will be sent during the session.

This *FragSessionSetupReq* message can be extended to solve the problem of the long Class B switching time (Beacon Acquisition Phase) when increasing the number of devices. The time of the Beacon Acquisition Phase is not necessarily the problem, the number of extra receive windows opened when switched to Class B mode are. Especially for the devices that switch to Class B early in the Beacon Acquisition Phase. We propose adding an extra session start time field to the *FragSessionSetupReq* message. This session start time indicates when multicast messages will be transmitted, until then, end-devices can go to sleep. No unnecessary receive windows will be opened before the multicast messages are transmitted.

For example, looking at the results in Section 3.4.3, it takes on average 16 extra beacon periods before it can start sending multicast firmware fragments. This means that an end-device that is switched to class B at the beginning of the Beacon Acquisition Phase, opens $32 \times 8 = 256$ ping-slot receive windows (when using periodicity 3) without actually receiving data. By telling the end-devices

it has to open these ping-slot receiving windows only after 16 beacon periods the power consumption can be reduced by 265 mJ. In comparison with a large firmware update in combination with a low data rate, the power saving does not seem much, however for smaller updates with a higher data rate it makes much more difference. For example consider the 7.1 kB firmware update with a data rate of 3, it only consumes 1.61 J of power. By using the extra session time, the power consumption is reduced by 16 %.

Chapter 4

Conclusion

In this thesis we evaluated the efficiency and scalability of transmitting firmware updates over LoRa. The two main questions were: *"What is a reliable model to estimate the energy consumption of a firmware update?"* and *"What is the increase in packet losses during a firmware update?"*. The answers are found by experimenting with a network setup with one gateway and up to 100 end-devices. For the gateway a Raspberry Pi [14] was used. The end-devices, consisting of the STM32F070CR microcontroller [31] and a SX1276 radio module [11], were made by the company TWTG [36].

We divided the FUOTA process in three different phases and measured the activities during these phases. The first phase is the Initial Phase, where all the FUOTA setup messages are sent. The second phase is called the Beacon Acquisition Phase, where each end-device listens for beacons sent by the gateway, once received the end-devices switches to class B. When each end-device is switched, the Multicast Phase starts. In this last phase the actual firmware data is sent using multicast downlink packets.

The main power consuming events were found to be receiving and transmitting data over LoRa. With a relatively simple power consumption model the power consumption during the different phases was calculated in the ideal situation (no packet losses, delays etc.). These calculations showed that even though sending one uplink message is more power consuming than receiving a message, the Multicast Phases was the most power consuming phase due to the large number of received multicast messages.

In terms of scalability, we performed four experiments with different number of end-devices in the network: 15, 30, 60 and 100 end-devices. We showed that the average number of uplink messages in the Initial Phase did not increase much when scaling up the number of end-devices. However, there were a few end-devices which had to send three times more uplink messages when using 100 end-devices. The poor random implementation of the deviation between each message might be the cause of this problem. We also showed that the Beacon According Time significantly increases when using more end-devices. More end-devices in the network require more downlink messages by the gateway, however the gateway has a duty cycle restriction of 1 % resulting in long wait times for some end-devices. During the Beacon Acquisition Phase, end-devices switch to class B. Once in class B they open a receive window periodically. However, the Multicast Phase only starts when all the end-devices are switched to class B.

All the end-devices in class B during the Beacon Acquisition Phase are opening receive windows for nothing and therefore consuming extra power.

Based on these results, we proposed a small change to the existing communication protocol. We propose to add a session start time to *FragSessionSetupReq* message. This session start time tells the end-device when to expect the multicast messages, until then it can go to sleep. This reduces the number of unnecessary opened receive windows during the Beacon Acquisition Phase and safe up to 265 mJ when using a periodicity of three.

To evaluate what the impact is on non-FUOTA devices (devices that are part of the network but not updated), we performed experiments with 10 FUOTA end-devices and one non-FUOTA end-device. By changing the ping-slot periodicity and firmware fragment size, we concluded that the higher the multicast firmware transmission duty cycle the more downlink messages meant for the non-FUOTA devices were lost. When using a duty cycle of 10 %, almost 20 % of the downlink messages were lost. The duty cycle restriction was not the problem because the firmware fragments were sent at a different sub-frequency band. The problem was that the gateway can only send one message at the time, resulting in lots of conflicts between the firmware downlink messages and the non-FUOTA downlink messages. Because of these conflicts we recommend to use a dedicated FUOTA gateway that is responsible for sending the firmware fragments to the end-devices. In this way other devices o can continue without any conflicts.

Lastly we evaluated the feasibility of FUOTA over LoRa by using firmware image developed by TWTG for their wireless Valve sensors [35] as reference. Three different updated sizes (120 kB, 69 kB and 7.1 kB) were evaluated when sending with different data rates. The power consumption increases quadratically with the data rate (lower data rate requires more power). When sending 120 kB with DR0 the total additional power consumption of the firmware update is only 0.73 % of the battery, considering a battery of 12.9 W h.

Bibliography

[1] Khaled Abdelfadeel, Tom Farrell, David McDonald, and Dirk Pesch. How to Make Firmware Updates over LoRaWAN Possible. In *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 16–25, Cork, Ireland, August 2020. IEEE.

[2] LoRa alliance. LoRaWAN 1.0.3 specification. `https://lora-alliance.org/wp-content/uploads/2020/11/lorawan1.0.3.pdf`, 2018. Lase accessed: Apr 28 2021.

[3] LoRa alliance. FUOTA process summary technical recommendation. `https://lora-alliance.org/wp-content/uploads/2020/11/tr002-fuota_process_summary-v1.0.0.pdf`, 2019. Lase accessed: Jan 26 2021.

[4] Martin Bor and Utz Roedig. LoRa Transmission Parameter Selection. In *2017 13th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 27–34, Ottawa, ON, June 2017. IEEE.

[5] Orne Brocaar. Chirpstack open-source application server. `https://www.chirpstack.io/application-server/`. Lase accessed: Apr 28 2021.

[6] Orne Brocaar. Chirpstack open-source network server. `https://www.chirpstack.io/network-server/`. Lase accessed: Apr 28 2021.

[7] Lluís Casals, Bernat Mir, Rafael Vidal, and Carles Gomez. Modeling the Energy Performance of LoRaWAN. *Sensors*, 17(10):2364, October 2017.

[8] LoRa Alliance Technical committee. LoRaWAN 1.0.2 regional parameters. `https://lora-alliance.org/wp-content/uploads/2020/11/lorawan_regional_parameters_v1.0.2_final_1944_1.pdf`, 2017.

[9] Semtech Corporation. What are LoRa and LoRaWAN? `https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/`. Last accessed: Nov. 09, 2020.

[10] Semtech Corporation. LoRa modulation basics. `https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf`, 2015. Last accessed: Nov. 09, 2020.

[11] Semtech Corporation. SX1276/77/78/79 - 137 MHz to 1020 MHz low power long range transceiver. `https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiAbqcpqh9qSjE`, May 2020. Last accessed: Jul, 13, 2021.

[12] LTD. Dragino Technology Co. 10 channels - LoRaWAN GPS concentrator for Raspberry Pi. https://www.dragino.com/products/lora/item/149-lora-gps-hat.html. Last accessed: Jul, 13, 2021.

[13] Fhessel. fhessel dragino pi gateway fwd Github repo. https://github.com/fhessel/dragino_pi_gateway_fwd, January 2020. Last accessed: Apr, 10, 2021.

[14] Raspberry Pi Foundation. Raspberry Pi 3 model b+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. Last accessed: Jul, 13, 2021.

[15] The PostgreSQL Global Development Group. PostgreSQL: The world's most advanced open source relational database. https://www.postgresql.org/. Last accessed: Jul, 13, 2021.

[16] Cian Guinee and Jonathan Dukes. Efficient firmware update transmission for LoRa low power wide area technology. https://www.scss.tcd.ie/publications/theses/diss/2019/TCD-SCSS-DISSERTATION-2019-013.pdf, 2019.

[17] ECS Inc International. Ecs txo 2520 datasheet. https://ecsxtal.com/store/pdf/ECS-TXO-2520.pdf. Last accessed: 1 Aug 2021.

[18] Jan Jongboom. Towards firmware updates over LoRa: cryptography and delta updates. https://os.mbed.com/blog/entry/towards-fota-lora-crypto-delta-updates/, October 2017. Last accessed: Jul. 26, 2021.

[19] Jan Jongboom. Armmbed/mbed-os-example-lorawan-fuota Github page. https://github.com/armmbed/mbed-os-example-lorawan-fuota, May 2019.

[20] J.Catalano (Kerlink), J-P.Coupigny (STMicroelectronics), N.Sornin (Semtech), and J.Stokking(The Things Network Foundation). LoRaWAN fragmented data block transport specification. https://lora-alliance.org/sites/default/files/2018-09/fragmented_data_block_transport_v1.0.0.pdf, 2018. Last accessed: 13 Jul 2021.

[21] J.Catalano (Kerlink), J-P.Coupigny (STMicroelectronics), J.Delclef (STMicroelectronics), A.Grigore (Flashnet), J.Schlarb (Comcast), N.Sornin (Semtech), J.Stokking (The Things Network Foundation), and A.Yegin (Actility). LoRaWAN remote multicast setup specification v1.0.0. https://lora-alliance.org/resource_hub/lorawan-remote-multicast-setup-specification-v1-0-0/, September 2018. Last accessed: 13 Jul 2021.

[22] Arm Limited. Mbed OS. https://os.mbed.com/mbed-os/. Last accessed: Aug. 4, 2021.

[23] QUITECH. Otii by QOITECH. https://www.qoitech.com/. Last accessed: Jul, 13, 2021.

[24] Peter Ruckebusch, Eli De Poorter, Carolina Fortuna, and Ingrid Moerman. GITAR: Generic extension for Internet-of-Things ARchitectures enabling dynamic updates of network and application modules. *Ad Hoc Networks*, 36:127–151, January 2016.

[25] Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, Jeroen Hoebeke, and Eli De Poorter. Modelling the energy consumption for over-the-air software updates in LPWAN networks: SigFox, LoRa and IEEE 802.15.4g. *Internet of Things*, 3-4:104–119, 10 2018.

[26] Saft Group SA. Ls, lsh, lsp — saft batteries. `https://www.saftbatteries.com/products-solutions/products/ls-lsh-lsp`. Last accessed: Jul, 21, 2021.

[27] Semtech. Semtech website. `https://www.semtech.com/`. Last accessed: Aug, 25, 2021.

[28] Semtech. Lora-net/loramac-node github page. `https://github.com/Lora-net/LoRaMac-node`, July 2021. Last accessed: Jul, 13, 2021.

[29] Yonatan Shiferaw, Apoorva Arora, and Fernando Kuipers. LoRaWAN Class B Multicast Scalability. In *2020 IFIP Networking Conference (Networking)*, pages 609–613, June 2020.

[30] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, page 188, Baltimore, MD, USA, 2004. ACM Press.

[31] STMicroelectronics. Mainstream Arm Cortex-M0 value line MCU with 128 Kbytes of flash memory, 48 MHz CPU, USB. `https://www.st.com/en/microcontrollers-microprocessors/stm32f070cb.html`.

[32] Johan Stokking. Firmware updates over low-power wide area networks. `https://www.thethingsnetwork.org/article/firmware-updates-over-low-power-wide-area-networks`, July 2017. Last accessed: Jan, 19, 2021.

[33] Jean-Jacques Chaillout Randa Jaouadi Taoufik Bouguera, Jean-François Diouris and Guillaume Andrieux. Energy consumption model for sensor nodes based on lora and lorawan. *Sensors 2018*, May 2018.

[34] J.CATALANO (Kerlink)-N.SORNIN (Semtech) J.CATALANO (Kerlink) J.CATALANO (Kerlink) J-P.COUPIGNY (STMicroelectronics) M.KUYPER (TrackNet) 56 N.SORNIN (Semtech) A.YEGIN (Actility) T.KRAMP (Semtech), A.YEGIN (Actility). LoRaWAN remote multicast setup specification v1.0.0. `https://lora-alliance.org/wp-content/uploads/2020/11/tr002-fuota_process_summary-v1.0.0.pdf`, January 2019. Last accessed: 18 Jul 2021.

[35] TWTG. NEON valve sensor QT. `https://www.twtg.io/products/neon-valve-sensor-qt/`. Last accessed: Jul, 20, 2021.

[36] TWTG. TWTG company website. https://twtg.io. Last accessed: Jul, 20, 2021.

Appendix A

LoRaWAN 1.0.3 Commands

Quick reference to the used LoRaWAN 1.0.3 commands. **This is a quick reference and most parts are directly copied from the LoRaWAN 1.0.3 Specification [2] and the LoRaWAN Remote Multicast Setup Specification [21].**

A.1 DeviceTime command (DeviceTimeReq, DeviceTimeAns)

The device timing commands can be used to synchronized the internal clock of an end-device with the gateway.

DeviceTimeReq

Request seconds since epoch[1]. This message does not contain any payload.

DeviceTimeAns

Answer to the *DeviceTimeReq*, the payload is defined as follows:

Byte#	4:1	0
Payload	32 bit unsigned interger: seconds since epoch[1]	8 bit unsigned interger: fractional second in $\frac{1}{2}^8$ seconds step

Table A.1: **DeviceTimeAns payload format**

[1] seconds since Sunday January the 6th 1980 at 00:00:00 UTC

A.2 PingSlotInfo commands (PingSlotInfoReq, PingSlotInfoAns)

PingSlotInfoReq

Send by an end-device to inform the server of its unicast Class B periodicity. The payload is defined as follows:

Bit#	7:3	2:0
Payload	RFU	3 bits: Periodicity

Table A.2: **PingSlotInfoReq payload format. RFU means reserved for future use.**

PingSlotInfoAns

Acknowledge to the *PingSlotInfoReq*, this message does not have a payload.

A.3 Remote Multicast Setup Commands (McGroupSetupReq, McGroupSetupAns, McClassBSessionReq, McClassBSessionAns)

These commands are used to remotely setup a multicast session.

McGroupSetupReq

This command is used to create or modify the parameters of a multicast group. The payload is defined as follows:

Byte#	28	27:24	23:8	7:4	3:0
Payload	McGroupIDHeader	McAddr	McKey_encrypted	minMcFCount	maxMcFCount

Table A.3: **McGroupSetupReq payload format**

- *McGroupIDHeader* contains the *McGroupID* which is the multicast group ID of the multicast context.
- *McAddr* is the multicast group network address.
- *McKey_encrypted* is the encrypted multicast group key from which McAppSKey and McNetSKey will be derived. These keys are used for encryption of the multicast message.
- The *minMcFCount* field is the next frame counter value of the multicast downlink packet to be sent by the server for this group.
- *maxMcFCount* specifies the life time of this multicast group expressed as a maximum number of frames.

McGroupSetupAns

The end-device sends a acknowledgement using the *McGroupSetupAns*. The payload is defined as follows:

Bit#	7:3	2	1:0
Payload	RFU	IDError	McGroupID

Table A.4: **McGroupSetupAns payload format**

When set, the *IDerror* bit indicates that the end-device does not support the multicast context indexed by the *McGroupID* requested by the server.

McClassBSessionReq

This message is sent by the server and used to setup a temporary Class B session. The payload is defined as follows:

Byte#	9	8:5	4	3:1	0
Payload	McGroupIDHeader	SessionTime	TimeOutPeriodicity	DLFreq	DR

Table A.5: **McClassBSessionReq payload format**

- *McGroupIDHeader* contains the *McGroupID* which is the multicast group ID of the multicast context.
- *SessionTime* contains the Class B session start time in seconds since epoch.
- *TimeOutPeriodicity* encodes the maximal session time in beacond periods (128 seconds)
- *DLFreq* encodes the frequency used for the multicast.
- *DR* encodes the data rate used for the multicast.

McClassBSessionAns

The end-device acknowledges the session setup message by sending a *McClassB-SessionAns*. The payload format is defined as follows:

Byte#	93	2:0
Payload	Setup&McGroupID	TimeToStart

Table A.6: **McClassBSessionAns payload format**

- *Setup&McGroupID* encodes the multicast group ID and the status of the received *McClassBSessionReq*. The status contains status bits to indicate if the frequency, data rate and multicast group are correctly setup.
- *TimeToStart* sends back the number of seconds until the session starts.