Delft University of Technology
Master of Science Thesis in Embedded Systems

Firmware Updates Over The Air for LoRa using Random Linear Network Coding

David Zwart

TU Delft Delft University of Technology

Firmware Updates Over The Air for LoRa using Random Linear Network Coding

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

David Zwart
d.j.zwart@student.tudelft.nl
davidzwa@gmail.com

27th June 2022

Author
David Zwart (d.j.zwart@student.tudelft.nl)
(davidzwa@gmail.com)
Title
Firmware Updates Over The Air for LoRa using Random Linear Network Coding
MSc Presentation Date
27th June 2022

Graduation Committee

Dr. Prof. R.R. Venkatesha Prasad	Delft University of Technology
Dr. Prof. R.K. Bishnoi	Delft University of Technology
MSc. K. Kroep	Delft University of Technology

Abstract

LoRaWAN is a public Wireless Sensor network with excellent properties like being long-range, low-energy radios and resulting in long battery life. Devices are connected to this network through gateways, and they will run in that deployment for years without replacement. Therefore, bugs and security issues in such devices' firmware are present for a long time, unless firmware updates are applied to fix them. Firmware-Updates Over-The-Air (FUOTA) is a firmware update application framework for LoRaWAN, but packet loss is an unsolved issue for such firmware updating frameworks. State-of-the-art packet dissemination uses the Low-Density Parity Checks (LDPC), but this has decoding overhead. Also, it is inflexible due to its fixed-rate nature. The code can not be dynamically adjusted to adapt to temporary changes in channel conditions.

In this work, we provide critical analysis to justify replacing Low-Density Parity Checks code (LDPC) proposed in FUOTA with the famous code Random Linear Network Coding (RLNC). The benefit of RLNC shows when scaling to multicast networks using LoRaWAN FUOTA as fewer messages need to be exchanged to serve the complete network of firmware updates. An analysis is presented on how to configure the finite field size, generation size, and redundancy for generation-based RLNC. The parameters are optimized to cope with the worst-case packet-error rate. As such, RLNC can optimally counter every lost packet with redundancy with near-zero decoding overhead. Since devices sleep after decoding a set of fragments, or the generation, sending additional fragments does not decrease efficiency. Finally, the decoding probability is provided as an analytical tool to show that a specific amount of redundancy can deal with a specific worst-case packet-error rate.

To evaluate and test the RLNC code for LoRa, embedded firmware and terminal software have been developed to do indoor and outdoor measurements. This testbed has been developed with a custom control plane replacing all FUOTA MAC layer modules (Fragmentation, Multicast, Synchronization, Device Management, etc.). Therefore, it can be shaped into custom network configurations meant for evaluating the firmware updating process without having to set up infrastructures like ChirpStack or TheThingsStack. Our work's testbed can isolate the decoding process and evaluate it for artificial packet drops.

Results are presented, which show that systematic coding phase of LDPC will perform poorly compared to RLNC. This systematic phase is at least as large as the firmware update. Between 38 % to 88 % improvement in decoding success is found by using RLNC in case of varying network conditions due to burst loss.

Preface

Doing a thesis project in the middle of the COVID-19 pandemic slowed things down. My motivation was stunted, and even doing the simplest tasks was tough. Also, visiting the faculty was not an option. Some people work well in isolation, but everyone needs social interaction. Luckily, this changed entirely after the university opened for students. This is why I felt like my project started only at that moment. The lesson I learned is that having people around you leads to rhythm, happiness, enthusiasm, and many creative moments.

Many directions and ideas were considered in the search for novelty. Most of these ideas did not make it into this thesis. I now accept that self-found ideas take a long time to fully conceptualize, review for novelty, and implement. This makes the end result truly your own and is very risky. It is a scary road, but I'm glad I took it.

I want to express my respect and gratitude to MSc. Kees Kroep, my daily supervisor. You bring great excitement to doing research and solving Game Theory puzzles! Secondly, thanks to Dr. R.R. Venkatesha Prasad for providing me the opportunity and guidance to undertake this project. I would also like to thank my direct and indirect friends and colleagues in the ENS group for the unlimited amount of coffee, wild goose-chase brainstorming sessions, and feedback. Specific thanks go out to Anup, Niels, Layla, Sury, Naram, Josine, Tareq, Mike, Gabe, Eric, Vineet, and many others for helping in collecting and reshaping the many divergent ideas that a thesis project brings with it. I would like to thank you, Britt, for all your support, for being the lovely and ever-happy person you are, and for helping me get through this challenging yet memorable period. Finally, I would like to express my gratitude for my family and, specifically, my mother.

David J. Zwart

Delft, The Netherlands
27th June 2022

Contents

Acronyms

ACK Acknowledgement. 20, 25, 29, 67

ARQ Automatic Repeat Request. 20, 29

blob Binary Large Object. 23–25, 28, 35, 43

BW Bandwidth. 10–13, 44, 52, 53, 57

CDF Cumulative Distribution Function. 35

CR Coding Rate. 10, 19, 66

CRC Cyclic Redundancy Check. 10, 19, 33

CSS Chirp Spread Spectrum. 10

DC Duty Cycle. 12, 13

DR Data Rate. 10

FEC Forward Error Correction. 10, 19, 20

FOTA Firmware Over-The-Air. 19

FUOTA Firmware-Updates Over-The-Air. 6–8, 14–16, 19–21, 23, 27, 39, 40, 47–50, 66–68, 73, 75

IID Independent and Identically Distributed. 62

IoT Internet of Things. 5, 19, 28

LCG Linear Congruential Generator. 32

LDPC Low-Density Parity-Check. 7, 8, 19–21, 49, 50, 65–68

LFSR Linear Feedback Shift Register. 32

LoRa Long-range wireless area network. LoRa is a long-range low-energy proprietary radio technology brought forward by the chip manufacturer Semtech and the non-profit organisation LoRa Alliance. iii, 5–13, 16, 19–21, 23, 25, 37, 39–41, 43, 44, 46, 48–50, 66

LoRaWAN LoRaWAN is a public wide area network (WAN) providing connectivity and compatibility with many off-the-shelf LoRa chipsets. iii, 5–9, 12–17, 19–21, 25, 40, 47–50, 62, 65, 67, 68, 73, 74

LPWAN low power wide area network. 6

LRConf LoRaConfigurator. 40–43

LUT Look-up Table. 26, 27

MAC Medium Access Control. 7, 9, 12, 14–16, 21, 23, 39, 40, 47, 49, 62, 67, 73, 74

MC Multicast. 44

NC Network Coding. 20, 23, 25

NS Network Server. 16, 17, 48, 49, 67

OAP Over-The-Air Programming. 19, 28

OSI Open Systems Interconnection model. 9

OTA Over-The-Air. 19

PER Packet Error Rate. 28, 29, 40, 45–49, 52–54, 57–60, 62–64

PHY Physical. 9, 10, 15, 47

PMF Probability Mass Function. 35, 63

pRNG Pseudo-random number generator. 29–32, 43

RLNC Random Linear Network Coding. 7, 8, 19–21, 23, 25–36, 39, 40, 43–45, 47, 49–51, 57, 61, 62, 65–68

RREF Row-reduced Echelon Form. 26, 30, 33

RSSI Received Signal Strength Indicator. 41, 43, 45, 52–55

SF Spreading Factor. 10–12, 40, 44

SNR Signal-to-Noise Ratio. 41, 43, 45, 52–55

stream Abstraction for a sequence of messages transmitted through a computer system or network. 24, 25

TOA Time-On-Air. 11, 12, 14

UC Unicast. 44

WAN Wide Area Network. 13

WSN Wireless Sensor Network. 19

Symbols

BW Bandwidth. 11

CR Coding rate. 11

D Number of end devices. 60, 61

F Fragment size. 23, 24, 30, 34–36, 43

G Generation size. 28–30, 33–36, 43, 57, 58, 60, 61, 65

$N_{\mathbf{payload}}$ Packet payload symbol count. 11, 12

$N_{\mathbf{pre}}$ Packet preamble symbol count. 11

N_f Number of firmware fragments. 23, 24, 34–36, 43

N_g Number of generations. 33, 34, 36, 66

$P_{\mathbf{GD}}$ Network decoding success probability. 60

$P_{\mathbf{TX}}$ Transmission power. 10, 13, 52, 53, 57

$P_{\mathbf{fail}}$ Decoding failure probability. 35, 58, 60

R Redundancy-extended generation size. 30, 33–37, 48, 57, 58, 60, 61, 65, 66

SF Spreading factor. 11, 12

$T_{\mathbf{decode}}$ Decoding delay. 36

$T_{\mathbf{firmware}}$ Firmware update duration. 23, 24, 36

$T_{\mathbf{packet}}$ Packet duration. 11, 23, 24, 36, 37

$T_{\mathbf{payload}}$ Packet payload duration. 11

$T_{\mathbf{pre}}$ Packet preamble duration. 11

$T_{\mathbf{symb}}$ Chirp symbol duration. 11

$U_{\mathbf{firmware}}$ Firmware update size. 23, 24, 34–36

W PER filter window. 46, 52, 53, 55

δ Redundancy factor. 30, 33, 36, 37, 43, 48, 57, 61, 65, 66

ϵ Uniform packet-error rate. 35, 37, 57–62

π_B Ratio of time spent in burst state. 62–64

π_E Expected mean per. 62, 64

π_G Ratio of time spent in good (non-burst) state. 62

$\mathbf{E}$ Encoding submatrix. 29, 30, 33

$\mathbf{F}$ Uncoded fragment submatrix. 29, 30

h Burst packet error rate. 62

k Good (non-burst) packet error rate. 62

p Burst state entry probability. 62, 63

q Order of a finite field. 25, 26, 35, 36

r Burst state exit probability. 62, 63

x Good vs Burst frequency/duration coefficient. 63, 64, 66

M_D Decoding matrix. 28–30, 32, 33

M_E Encoding submatrix. 28–30

Φ Encoded fragment submatrix. 29, 30

Chapter 1

Introduction

The infrastructure of Internet of Things (IoT) is being enriched by many different free-to-use non-cellular radio technologies and LoRaWAN (Figure 1.1) is a very popular one. However, the wireless protocol LoRa, which drives LoRaWAN, is not a good fit for transmitting large payloads required for, for example, firmware updating.

Figure 1.1: **LoRa and LoRaWAN logos**

LoRa technology provides connectivity for a high amount of devices while maintaining reliable long-range communication and low energy consumption - a genuinely unique combination of capabilities. Applications like climate and crop monitoring, automation and control of irrigation, smart building/lighting control, and remote energy metering (gas, water, solar energy) are just a few of the many applications where LoRaWAN is beneficial. See Figure 1.2 for such a network.

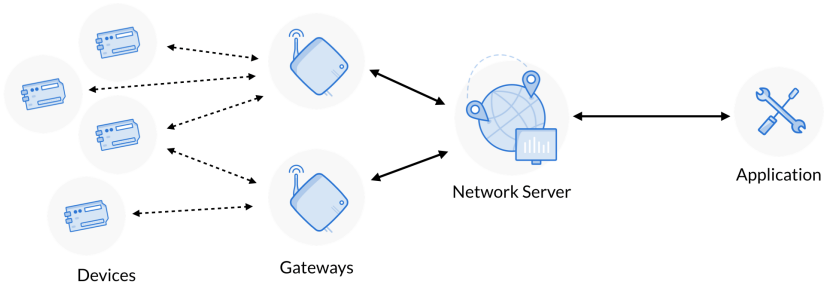


Figure 1.2: **Example LoRaWAN star network with two gateways providing connection for multiple end devices to a LoRaWAN backbone (Network Server and Application backends).**

LoRa is free-to-use for the public, and it is set up using inexpensive gateways and IoT end devices, each equipped with a standardized LoRa radio. The LoRa radio uses a unique wireless technique to receive messages with low signal

strength, allowing large areas to be covered. When a LoRa end device transmits back messages, it consumes similar amounts of energy compared to other low-power networks. However, the end device is put to low-power sleep when not transmitting or receiving, resulting in low energy consumption and years of battery life. The device can only send messages at relatively low speeds to ensure that long distances are covered. So, low energy consumption, long sleeping times, low data rates, long-range, and long battery lifetimes make LoRa an attractive network. It is for these reasons that the LoRaWAN network (see Figure 1.2) is estimated to be the leading non-cellular low power wide area network (LPWAN) technology by 2026 making it very interesting for research studies like those presented in this work[1].

So, what is the catch? The challenge of low-power long-range communication is the chance of lost messages due to interference with other devices transmitting at the same time. The result is packet loss, which can be detrimental. For LoRa, it is common to disable acknowledgments and retransmissions to keep the traffic simple and predictable. Consequently, packet loss is something that needs to be taken into account when designing a LoRa application and network configuration. Also, the device is inactive due to its long sleep time to save power, which means it is harder to communicate with it, and its availability is, therefore, low. Quite some engineering effort goes into being able to communicate with such a sleepy device.

1.1 Reducing device maintenance

Although deploying the network on remote locations is feasible with LoRa, it is challenging to reach the devices once maintenance is necessary. Consequently, the cost and difficulty for an engineer to go around are high. It, therefore, is cheaper and quicker if the firmware could be updated remotely and for all devices in a deployed network. Given that LoRa end devices face packet loss and that firmware updates require many packets to be sent in sequence, a new solution is needed to give applications running over this network the capability to be updated reliably. Similar to other wireless network types, in LoRaWAN the application firmware update mechanism is named Firmware-Updates Over-The-Air (FUOTA)[2]. This name resembles its remote aspect of updating devices using LoRa as the wireless physical layer and LoRaWAN as the medium access control layer.

Without remote or on-premises firmware updates, devices are prone to security vulnerabilities or reduced compatibility with the application. It is a fact that on-premises firmware updates are a necessity and requirement[3]. However, transmitting new firmware in fragments with LoRa is not trivial. The LoRa network presents a challenge due to the high chance of packet loss. Therefore, a firmware update might not arrive entirely or correctly. This thesis presents a combination of novelties to improve firmware updates for LoRa in the following problem statement:

Thesis Problem Statement Firmware updates specifications and example applications have been released into public domain[4, 5, 6, 7]. In other work, the security aspects of FUOTA, multicast scalability and energy consumption have been analyzed[8, 9, 10, 11]. Certainly, applications have already been realized,

which implement FUOTA for multiple LoRa devices. These applications use three proposals which extend the LoRaWAN Medium Access Control (MAC) layer with commands for block fragmentation, time synchronization and multicast communication for FUOTA[12, 13, 14]. Given the progress in this area, it is surprising that the evaluation of the fragment block code *Low-Density Parity-Check (LDPC)* used in FUOTA is missing. To the best of the author's knowledge, this choice has not been critically evaluated or improved with a better fragment code. Secondly, no concrete methods were developed to analyze the success rate of FUOTA, irrespective of the implemented fragment decoder. Such a method is essential, given that LoRaWAN networks have to deal with varying conditions due to packet loss and burst loss.

The thesis objective is as follows:
Implement Random Linear Network Coding (RLNC) for LoRa on provided STM32 F446RE embedded devices and SX1261 radio shields. Develop a robust testbed with measurement storage for evaluating packet erasure on devices placed remotely or nearby. Use increased network utilization to evaluate the RLNC decoder tolerance against real and artificial packet erasure due to uniform or burst loss. Provide analysis and observations of measurement results to suggest which code and coding configuration is most suitable for FUOTA under poor channel conditions.

1.2 Contributions

The main contributions described in this thesis are as follows:

- Developing a mathematical framework for evaluation of a RLNC decoder for performing high-duration firmware updates (Chapter 4). The generation decoding success rate is a metric for determining whether a specific encoding configuration is going to succeed.
- Develop a testbed using real hardware for measureming the RLNC decoding probability as well as LoRa network packet error rates. Different testbed configuration provide methods for evaluating RLNC with FUOTA applications in LoRaWAN (Chapter 5).
- Comparing LDPC currently used in FUOTA to RLNC (Chapter 5).
- Performing indoor and outdoor experiments to collect data for studying packet loss scenarioss.
- Evaluating performance of RLNC for burst loss using the Gilbert-Elliot burst loss model (Chapter 6).

1.3 Thesis structure

Chapter 2 provides background on LoRa and LoRaWAN. Chapter 3 presents related works and justifies the need for the research of this thesis using RLNC for FUOTA. Chapter 4 introduces the RLNC encoding/decoding scheme in detail and establishes relevant parameters, a mathematical analysis framework and

observations to be used for our testbed presented in Chapter 5. This chapter elaborates the hardware, software capabilities and custom LoRa control plane for doing evaluation of the RLNC end device decoder and provides a strategy on how to project the insights from Chapter 4 to a LoRa star-network and extends to the context of LoRaWAN class B and C in multicast mode. Following this, Chapter 6 presents the results and observations from experimental data and simulations to compare FUOTA using RLNC to FUOTA using LDPC and systematic coding. Finally, Chapter 7 concludes this work and provides leads for future work with interesting research targets to extend this work.

Chapter 2

Background

This chapter explains the three network layers LoRa and LoRaWAN consist of. Figure 2.1 shows that three media layers are covered compared to the *Open Systems Interconnection model (OSI)* model, which has four additional host layers. The Physical (PHY) and Data Link layer are treated as one layer in this chapter. Note that although this layer is configured differently per region, only the region Europe is assumed as the region of operation. Other regions are out of the scope of this work.

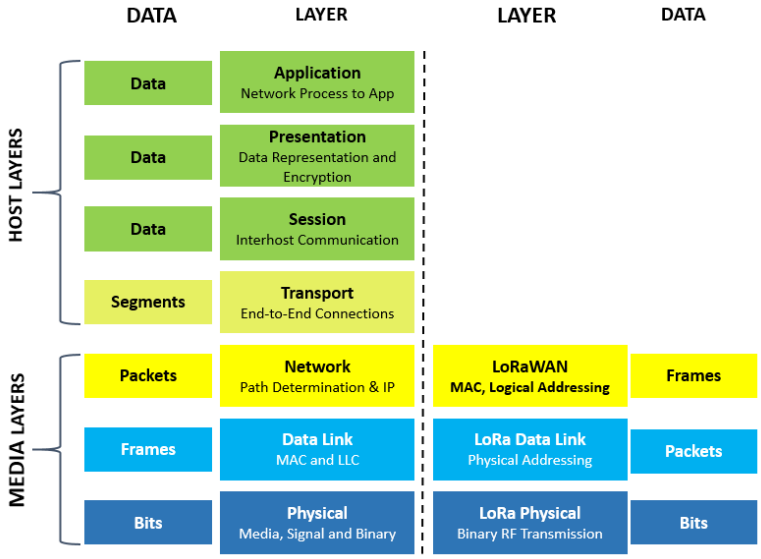


Figure 2.1: **OSI seven-layer vs LoRa three-layer network models (image from Semtech Developer Portal[15])**

Section 2.1 explains the control over the LoRa PHY Layer. Secondly, Section 2.2 presents the MAC Layer configurability and network commands in order to inter-operate with LoRaWAN. Finally, it is explained how an application layer needs to hook into the PHY and MAC layers in order to operate, which provides the basis for realising the testbed of Chapter 5.

2.1 LoRa physical layer

LoRa packets are sent using the Chirp Spread Spectrum (CSS) modulation technique. The symbols of a packet are transformed into up and down-chirp symbols when modulated by this technique. Each chirp starts on low frequency and ends on high frequency after which the next chirp symbol starts. The preamble consists of up-chirps which increase linearly in frequency, followed by two sync down-chirp symbols. Together the preamble and sync symbols are used for detecting a packet at a receiver. Finally, the data symbols are sent by applying an offset to the starting frequency of each chirp. This explains why the data symbols look like they are split into smaller pieces - the linear frequency increase causes the chirp to wrap to the lowest frequency if the maximum frequency is reached.

The LoRa protocol works very well even below the noise floor over long distance[16, 17]. This is achieved by spreading the symbols over time with the *Spreading Factor (SF)*, increasing the transmission power P_{TX} and increasing the signal *Bandwidth (BW)*. The SF parameter, of which its inverse is Data Rate (DR), stretches the symbols in time making it easier for a receiver to lock onto the signal. The downside is that this consumes more energy, so a balance between energy and signal quality needs to be found. The same holds for increasing the parameters P_{TX} and BW, although these are not specific to LoRa PHY only. The parameters BW, SF and P_{TX} define the packet duration, but these are not the only parameters to improve communication robustness.

Figure 2.2 shows the packet structure in symbol and binary representation. The preamble is 8 symbols long by default. This is followed by the optional header. This header, which contains its own *Cyclic Redundancy Check (CRC)* field, specifies the payload length, whether the 16 bits CRC is present at the end and what *Coding Rate (CR)* is applied. The CR provides Forward Error Correction (FEC), but will not help with lost packets(packet erasure)[1]. The CRC is a compact representation of the packet, also called a checksum, which can be calculated with the packet payload. If the reconstructed CRC value does not match the CRC in the packet, the packet is flagged as faulty and an error is sent to the radio LoRa receive error handler. Given that faulty packets are not valuable, throughout this thesis it is assumed the packets are erased as if they were never received at all.

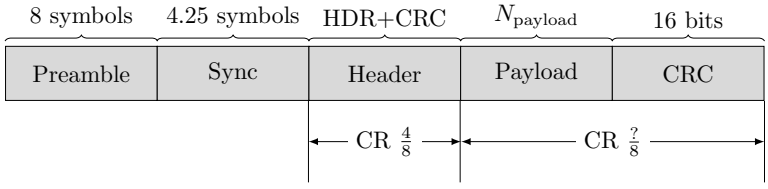


Figure 2.2: **LoRa PHY packet structure with preamble, optional header, payload and CRC fields. The payload and last CRC fields are coded with configurable CR.**

The packet is transmitted with a certain transmission power setting ran-

[1]The CR defines the amount of additional bits provided to correct a limited amount of packet bit errors. This FEC is part of the proprietary firmware of the LoRa radio frontend.

ging from −17 dBm to 14 dBm or 20 dBm (depending on choice of LoRa radio hardware). This thesis does not explore transmission power on received signal strength except for testing in preliminary lab experiments. The primary argument for this is that new LoRa radio hardware is released by manufacturers at a high pace, rendering measurements with any used radio chip outdated or even obsolete.

2.1.1 Packet duration

Long range wireless communication sacrifices some performance aspects compared to short range communication. Each packet takes time to transmit, called the *Time-On-Air (TOA)*. The longer each symbol transmission takes or the more symbols are transmitted for each packet, the more energy and time is required for each packet. Energy consumption is very important, but previous work has been done on this matter, and, therefore, the energy aspect is not in the scope of this thesis[11, 10, 18]. The aforementioned SF parameter is the most critical control parameter of the packet TOA.

The SF parameter defines the number of quantization steps per chirp or frequency step. Such a small frequency jump is called a *chip*. Each chirp increases in frequency by these quantized frequency jumps and wraps around at the highest frequency. A higher SF requires more chips to be traversed before the next chirp symbol starts, resulting in a quadratic increase in transmission duration. The BW defines the speed at which the chips are produced. Therefore, a higher BW results in a higher *chip rate*, which translated to a higher symbol rate and data rate. In other words, more data can be sent in the same chirp time frame. There are three possible values for BW: 125 kHz, 250 kHz and 500 kHz. However, 250 kHz is the most realistic bandwidth limit as is explained in Section 2.2.

Now the packet TOA equations are presented[19]. The choice of BW and SF define the maximum data rate and the data rate has great influence on the possible communication range. Equation (2.1) presents the inverse of the data rate, the symbol duration T_{symb}. Following, Equation (2.2) shows that the preamble duration is dependent on the symbol duration T_{symb} and preamble length N_{pre}. From Equation (2.3) and (2.4) it can be deduced that the payload duration is very configurable[2], whereas the preamble is quite static. Resulting, Equation (2.5) is mostly influenced by the payload settings.

$$T_{\text{symb}} = \frac{2^{SF}}{BW}, \tag{2.1}$$

$$T_{\text{pre}} = (N_{\text{pre}} + 4.25)T_{\text{symb}}, \tag{2.2}$$

$$N_{\text{payload}} = 8 + \max\left\{0, \left\lceil \frac{8N_{\text{payload}} - 4, SF + 28 + 16\text{CRC} - 20\text{H}}{4(SF - 2\text{DE})} \right\rceil (CR + 4)\right\}, \tag{2.3}$$

$$T_{\text{payload}} = N_{\text{payload}} \cdot T_{\text{symb}}, \tag{2.4}$$

$$T_{\text{packet}} = T_{\text{pre}} + T_{\text{payload}}. \tag{2.5}$$

[2]The payload symbol length has some intricate choices like optional header H, data rate optimization DE for SF 11 or 12 and some fixed parts like 16-bits CRC etc.

Equation (2.5) leads to the logarithmic TOA plots in Figure (2.3). This Figure shows that high SF like SF11 or SF12 result in packet duration in the order of hundreds of milliseconds up to multiple seconds. Therefore, great care must be taken to prevent excessive packet duration because the throughput of the network becomes very low. This might be desirable for extremely energy-optimized applications, but this thesis' premise depends on quite a high amount of packets to be transmitted and lower SF is more beneficial. In Chapter 6 this becomes clear when the firmware update success rate is evaluated.

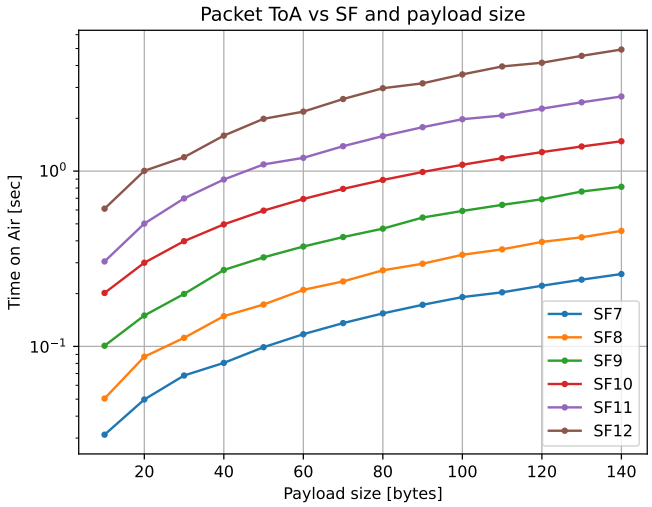


Figure 2.3: **Time on Air TOA at BW** 250 kHz **for different payload size** $N_{\mathbf{payload}}$ **and** SF**. High spreading factors like SF11 and SF12 show much higher packet duration** ($>$ 1 sec)**.**

The next section dives into the LoRaWAN MAC layer of LoRa, which introduces the fair use policies, constraints on available wireless frequencies, and the concept of the Duty Cycle (DC) limit is introduced. Furthermore, the three device classes A, B, and C are brought to light in the context of this thesis.

2.2 LoRaWAN layer

The LoRaWAN stack is an open standard maintained by the collective named the LoRa Alliance®. Continuous updates are released and for this reason, different versions exist with each specification manual. Practical MAC commands are added, but also security considerations are applied. Throughout this work, it is assumed that all information is based on version v1.1 (not specifications in the range of v1.0.x)[20].

LoRa frequencies are part of the ISM bands (Industrial, Scientific, and Medical radio bands). Gateways and/or end devices may use some of the same predefined (sub-)bands within the ISM bands given very tight fair use constraints. Such constraints ensure that high amounts of devices on LoRa or other network protocols can co-exist as much as possible. These limits are rarely enforced on a

radio hardware level, so it is often up to the application or Wide Area Network (WAN) backbone to enforce this. Table 2.1 presents the bands labeled *g*, *g1*, *g2*, *g3* and *g4*. It is due to the 10 % DC and 500 mW transmission power limits that *g3* is the most viable band to use for the firmware update application.

Name	Min Freq.	Max Freq.	DC	Max BW[a]	Max P_{TX}
g	863.0 MHz	868.00 MHz	<1 %	500 kHz	<25 mW
g1	868.0 MHz	868.60 MHz	<1%	500 kHz	<25 mW
g2	868.7 MHz	869.20 MHz	<0.1%	500 kHz	<25 mW
g3[b]	869.4 MHz	869.65 MHz	<10%	250 kHz	<500 mW
g4	869.7 MHz	870.00 MHz	<1%	250 kHz	<25 mW

[a]The bandwidth values here were capped off by configurable bandwidths. Offset sub-bands are available, but share the DC limit.

[b]The band *g3* is the best candidate for transmitting firmware updates due to its 10 % DC limit and 500 mw P_{TX}.

Table 2.1: **Frequency bands with duty cycle limits (DC), bandwidth (BW) and max transmission power (P_{TX}).**

Next, the three classes of LoRaWAN end device modes are explained to understand better what will practically work for the application at hand. Before this is explained, the concept of message flows needs to be defined. LoRa acts as a star network. This means messages can be sent from end devices to gateways and vice versa. The LoRaWAN network does not give incentive to implement peer-to-peer traffic between end devices as the DC limit is very low (1 % or 0.1 %). Therefore, the network type is classified as star network (see Section 2.2.3). It is common to define messages from the end device to the gateway as *uplink traffic*, which is also the most common traffic as will be shown later when class A is defined. Similarly, messages from the gateway to the end device are classified as *downlink traffic*. The downlink message flow direction is the most interesting in the scope of this thesis. Therefore, the next section will provide insight as to which LoRaWAN class supports downlink traffic capabilities best.

2.2.1 Class A, B, and C

Classes A, B, and C exist to provide runtime adjustments for devices to improve reachability by increasing energy consumption. This means devices in Class B and C will listen and possibly transmit more using their radio frontends, which is the main contributor to energy consumption. Class A is the default and in this class, it is impossible to transmit downlink messages asynchronously simply because the end device will be in sleep mode. This class defines that uplink messages are *device initiated* after which the network may schedule one or two downlink time slots. This is not a feasible class for sending many downlink messages. Secondly, the required downlink traffic for the firmware update application can benefit greatly from multicast downlink messages[3].

There is an easy shortcut to understand classes better:

- A: asynchronous (aloha uplink-oriented)

[3]Multicast messages can be received by multiple devices which are configured through the same multicast group using the GroupID identifier

- B: beacon (synchronous bidirectional)
- C: continuous (asynchronous bidirectional)

Each end device needs to upgrade by request or be upgraded dynamically to class B or C to function in either of these classes. It is assumed throughout this thesis that Class B or C is configured for a firmware update application that depends almost completely on downlink traffic. The reader is referred to work that explains the scalability of Class B for multicast traffic in great detail[9]. More recent work studies the implementation of firmware updates FUOTA for multicast traffic in Class B and analyzes bottlenecks in the existing MAC layer command structure required for this FUOTA application type[11].

Class B is time-synchronized using gateway beacons which are transmitted in a period of 128 sec by default. This sets up a strict time-division schedule between each beacon called *ping slots*. Each ping slot is a *contract* between the LoRaWAN network and the end device enforcing that the device listens for downlink transmission during that time slot. The number of ping slot subdivisions is determined by the *ping slot periodicity*. The resulting period between each possible slot is 1 sec up until 128 secs. From this, it is clear that the number of messages that can be sent in Class B can only be 128 messages per 128 seconds resulting in a maximum (optimistic) rate of *1 message per second*. Therefore, scheduling downlink messages is expensive, and transmitting messages with low information value is wasteful. Luckily, *multicast* messages can be sent using the same ping slot construct to reach more devices at once. The devices are (securely) registered into multicast groups first, which provides these devices with enough configuration to be able to listen to and decrypt the downlink messages from one or more gateways. This is an essential optimization, as now the same firmware update messages can be used for many devices at once.

The ping slots are randomly, yet deterministically, offset using *ping slot offset*. This value defines what offset the device and gateway will use for the ping slot periodicity. This mechanism exists to offer more downlink transmission opportunities for multiple different devices, or device groups. This ping slot offset is not studied as this thesis does not make assumptions about whether Class B or C is chosen.

It is fact that near the end of the beacon period, ping slots could cause the next beacon not to be scheduled for transmission[4]. Simply because a gateway cannot transmit more than one message at the same time. This is called *beacon blocking* and, although there exists a guard around the beacon transmission, it can cause some ping slots at the end of the cycle to be unuseful[21]. In effect, some ping slots at the end of the beacon period cannot be used for downlink multicast transmissions. This can be checked by calculating the TOA from Equation (2.5) and adding this to the absolute starting time of a ping slot. If the beacon guard window overlaps, it is best to either choose a shorter payload and thus shorter TOA. Alternatively, it is best to not send on those ping slots to ensure the Class B beacon can still be sent. Concluding, when taking into account such scheduling constraints Class B is a viable option for transmitting the intended multicast application fragments on a periodic time-synchronized sense[11].

[4]The beacons are transmitted with very tight timing error margins, so deferring transmission is not possible by design.

Class C dictates that the end device is continuously listening, which means the energy consumption is high. The benefit is of course that these devices can accept asynchronous downlink traffic compared to Class B and at possibly higher message rate as there is no ping slot construct. It is, however, not a given that Class C is viable for any network of devices, because this depends on the deployment. Therefore, throughout this thesis, *it is not defined whether either Class B or C is used.* Instead, the network throughput is optimized to be used as efficiently as possible[5]. This will be shown in Chapter 6 where channel conditions are taken into account. The next Section introduces the MAC layer packet structure, to show that the LoRaWAN packet has more overhead, which limits the size of the application payload to be sent.

2.2.2 LoRaWAN Medium Access Control

Figure 2.4 extends the PHY packet structure from Figure 2.2 and shows how three layers nest the application payload. The bottom layer contains the application payload (encrypted `FRMPayload`). The actual method of how the backbone network uses encryption to secure this payload is beyond this thesis' scope. Similarly, throughout this work it is assumed that the default configuration is used for the MAC, therefore, detailed explanation will be left out and this is presented for reference. Only the first Radio PHY layer is used in this work and the payload field in this layer is filled with custom control commands as explained in Chapter 5.

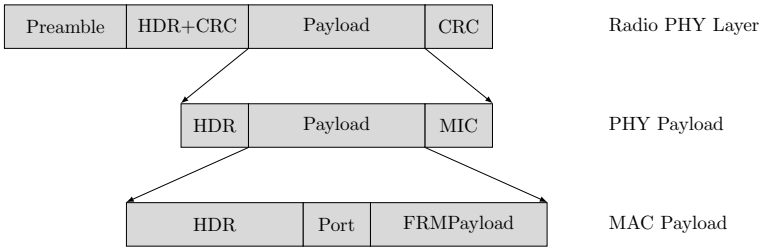


Figure 2.4: **Overview of LoRaWAN packet nesting structure for the Radio PHY layer, PHY payload and MAC payload layers. The bottom layer contains the encrypted frame payload `FRMPayload` for the application.**

For the FUOTA application there exist three extension specifications: clock synchronization[13], fragmentation[12] and multicast[14]. These specifications are summarized in one application process specification[22]. A short excerpt from the multicast specification is shown in Figure 2.5. It can be deduced from commands like `McClassBSessionReq` and `McClassCSessionReq` that the devices are commanded to join a multicast session for either Class B or C mode. The end device must have indicated their Class beforehand through an uplink message. How this specifically works for Class B differs from C. Class B has a bit field in the MAC payload header, whereas Class C has a custom MAC command `DeviceModeInd` for it (the header field must not be used for Class C).

[5]For all network coding tests using the testbed of this thesis, the radio is set to continuous listening (similar to class C) as energy consumption is not of concern.

CID	Command name	Transmitted by		Short Description
		End-device	server	
0x00	***PackageVersionReq***		x	Used by the AS to request the package version implemented by the end-device
0x00	***PackageVersionAns***	x		Conveys the answer to PackageVersionReq
0x01	***McGroupStatusReq***		x	Asks an end-device to list the multicast groups currently configured
0x01	***McGoupStatusAns***	x		Conveys answer to the McGroupStatus request
0x02	***McGroupSetupReq***		x	Provides an end-device will all necessary information to join a multicast group
0x02	***McGroupSetupAns***	x		
0x03	***McGroupDeleteReq***		x	Used to delete a multicast group from an end-device
0x03	***McGroupDeleteAns***	x		
0x04	***McClassCSessionReq***		x	Conveys information about the next classC multicast session the end-device shall join
0x04	***McClassCSessionAns***	x		
0x05	***McClassBSessionReq***		x	Creates a class B multicast session
0x05	***McClassBSessionAns***	x		

Figure 2.5: **Multicast commands extending the LoRaWAN MAC layer with Command ID (CID), transmitter origin and description. Image from LoRa Alliance[14].**

For brevity, most of the LoRaWAN MAC commands have been omitted here. For more information the reader is referred to Appendix A where relevant MAC commands for a FUOTA session are presented.

2.2.3 Network topology

The LoRaWAN network is a star network topology from the perspective of end devices(referring to Figure 1.2), which means that end devices are not in communication with each other. One or multiple gateways ensure end devices are connected to the backbone network, which has much more throughput as it is not built on top of LoRa but Ethernet, cellular, etc. The backbone consists of the *Network Server (NS)* which is a generic adapter serving traffic management from and to many gateways and end devices. Because multiple gateways can receive the same message from an end device, it is the responsibility of the NS to deduplicate the messages. The gateway does not process the internal message payloads, but instead, it only forwards the messages to the right NS.
The NS processes the message and forwards the resulting payload to a known and verified application server.

Figure 2.6 shows that multiple gateways can provide overlapping coverage for a subset of end devices. Therefore, the network topology from the perspective of NS is a two-hop network where the end devices and gateways form an incomplete bipartite graph. This abstraction of LoRaWAN is used as basis for projecting LoRaWAN on the testbed in Section 5.3. The next chapter will summarize relevant works used for this thesis, taking into account this network topology.

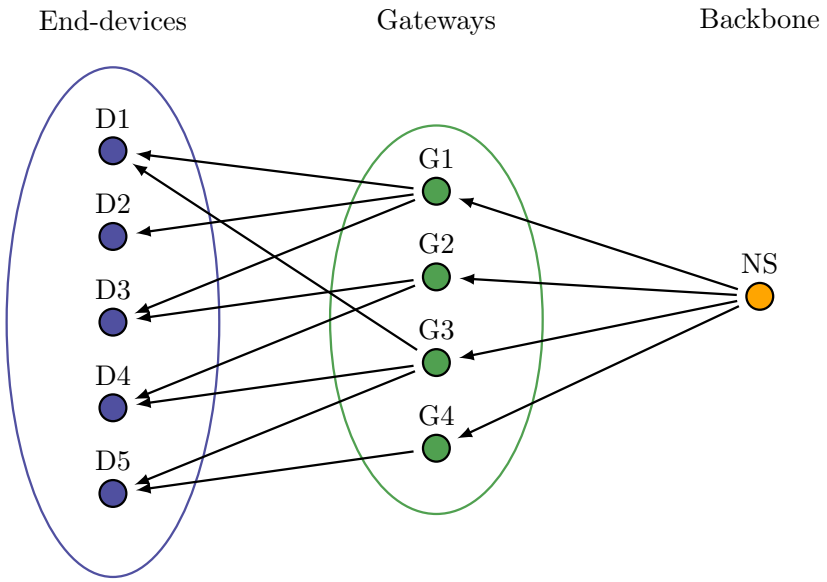


Figure 2.6: **The LoRaWAN network forms an incomplete bipartite graph connected to backbone infrastructure. End devices (blue) are serviced only by gateways (green). The Network Server (NS, orange) connects all gateways in a clique.**

Chapter 3

Related work

Firmware updates of embedded devices are done by directly connecting to a device. Therefore, after the devices are deployed in the field, a remote update mechanism is required to avoid having to visit each device separately. Exactly such firmware update protocols have been developed for almost all popular Wireless Sensor Network (WSN) protocols like Bluetooth[2], ZigBee[23], LoRaWAN[18] and IoT in general[24, 25]. These firmware updates protocols are often named Over-The-Air (OTA), Over-The-Air Programming (OAP), Firmware Over-The-Air (FOTA) or *Firmware Updates Over-The-Air FUOTA* application frameworks. Throughout this work we will refer to FUOTA applied to LoRa context - unless specified otherwise - due to this being the common term for it in the LoRaWAN community[26, 22, 6].
FUOTA has the responsibility of transmitting/receiving updates, correcting faults, and loading new firmware securely and reliably. As of now, a great number of studies have been done in the field of firmware updates for LoRaWAN. The main driving reason is that it is such a challenge to perform large-size data transfers over this error prone, low rate LoRaWAN networks[8, 11, 18, 27]. The primary challenges are packet errors and erasure. LoRa's proprietary physical layer provides FEC and CRC error detection on the hardware level, which can be configured on the software level. This becomes clear by reviewing the SX1261/2 datasheet, a commonly used LoRa radio shield and driver specification[19]. FUOTA also offers mechanisms to deal with both error and erasure correction using the block code *LDPC*[4, 12]. It is possible that this code was chosen, because of its popularity in 5G New Radio[28]. However, it is surprising that LDPC was selected as the *FEC capabilities* are already provided by the proprietary LoRa radio hardware. Therefore, LDPC is not optimal for LoRa devices. In the case packet errors are detected but the error count exceeds the maximum for the LoRa CR setting, the packet must be discarded. This means additional packet erasure. This fact has been previously fortified by the work of peers[29]. Although LDPC does provide limited capabilities for repairing erased packets, it has been shown that the statistical overhead due to non-ideal parity matrix rank can only be reduced by increasing the number of transmitted fragments to high numbers[12]. Using this reference, it is known that the overhead for only 20 fragments is 10 % and even for 100 fragments the overhead is on average 2 %. Later, it is shown that this can be improved: block-based generation RLNC has a neglibible decoding overhead.

3.1 Scalability

As packet erasure has been established as an issue, another challenge arises for FUOTA due to the number of downlink packets that must be sent to a possibly large group of devices. LoRaWAN supports multicast messages in its MAC layer to address this in Class B and C as was stated in Section 2.2. Previous works by peers present thorough analysis and simulations to address the scalability of using multicast for LoRa class B devices[9, 21]. From this analysis and by looking at popular LoRaWAN network stacks like TheThingsNetwork and ChirpStack, it has become clear that message confirmations using Acknowledgement (ACK) are not advised for LoRaWAN. To make this point even stronger, Class B or C do not support confirmations for multicast in the MAC layer. Therefore, it is up to the application layer to provide enough redundancy in the form of FEC or verification by aggregation or selective ACK using a form of Automatic Repeat Request (ARQ). This work focuses mainly on providing *feed-forward redundancy*, whereas ACK-confirmations are outside the scope of this work.

To the best of the author's knowledge FUOTA performance has not been analyzed yet in the context of non-ideal network conditions. Also, no average and burst erasure analysis has been applied to FUOTA framework using LDPC as of this moment. It's, for this reason, the well-studied encoding Network Coding (NC) protocol RLNC is proposed to deal with providing enough fragment redundancy, dealing with the scalability of a LoRa network, and, adapting to LoRa's multi-gateway star network.
RLNC has been developed for generic broadcast networks[30]. Recently, however, the erasure code was also evaluated for LoRa. The decoding latency has been studied through simulation, but no analysis or insights were presented on how the devices were modeled or what radio configurations lead to better throughput[31]. Also, this work assumes multi-hop network topology for LoRa, which is as of this moment not possible since LoRa must be set up in star network topology (single-hop).
Other recent studies, outside the field of LoRa, show that the code is a near optimal decoding protocol concerning decoding efficiency given the choice of the Galois field size (decoding complexity) is traded off with decoding latency and redundancy[32]. Recent work however also shows the decoding complexity of two RLNC encoders can be combined to reduce decoding complexity resulting in a Fulcrum or Adaptive Fulcrum code[33]. The authors extend the mathematical framework to trade-off decoding probability and computational complexity based on the original RLNC mathematical equations[30], which is also what has been done in this thesis report. However, concerning computational complexity it has been shown in work by peers that the energy consumption of the LoRa radio - not the primary micro-controller unit - is the greatest contributing factor to the total energy consumption of a LoRaWAN end device[11]. Given that a FUOTA session mostly requires the end device to receive messages over its radio, it makes sense to choose near-perfect decoding computational complexity if the total energy consumption is negligible and, more importantly, if a higher probability of decoding successfully is guaranteed. For this reason, this thesis focuses on providing analysis, real measurements, and validation using RLNC and not using newer Fulcrum codes.

3.2 State-of-the-art

It has been mentioned that FUOTA currently is provided with LDPC code, which has been presented in the LoRaWAN multicast MAC-layer specification[14]. Other crucial mechanisms like fragmentation[12], time synchronization[13] and process[22] MAC-layer extensions have also been published in specifications by the LoRa Alliance. Similarly, companies like ARM, Semtech and STMicroelectronics are developing their own take at FUOTA[4, 26], which is based on (or showed the need for) the specifications. The result is publicly available software and documentation resources [6, 7].

Other work shows how multiple gateways can simultaneously contribute to the FUOTA performance to counteract limits like the permitted duty-cycle on the selected ISM-band radio frequency[27, 34]. It is interesting to see from their simulation results that a higher amount of cooperating gateways reduces the total network energy consumption, and the update duration and increases the update efficiency. This thesis will not study combining multiple gateways as it's clear that the benefit of multiple gateways will still hold. Similarly, a lot of work was published concerning the cryptograpy and security aspects of firmware updates and the root-of-trust bootloader[6, 8]. This thesis will not try to improve this as it has been extensively reviewed.

As might be clear there is no work published that takes a critical look at the FUOTA erasure-correcting code from a network perspective. To gain insights on this level of abstraction, simulation is essential for evaluating scalability. One related work does this through a LoRaWAN multicast Class B implementation in the *ns-3* simulation framework[35]. These insights have been taken into account in this thesis. Another related work provides the simulator FuotaSim to analyze the performance of dissemination LoRaWAN under non-ideal channel conditions[36, 18]. This simulator implements the LoRaWAN MAC layer and shows the control plane commands. Surprisingly, the simulator does not incorporate the LDPC code of FUOTA although it is given that the code has imperfect decoding efficiency. Therefore, this thesis provides an alternative approach which does not simulate the LoRaWAN MAC layer. Recent work by peers studied and provided measurements of the MAC layer concerning energy and FUOTA session duration [11]. Instead, we provide a time-discrete simulation on the application level to provide a fair comparison between primitive uncoded dissemination, LDPC and RLNC. Real-life indoor and outdoor experimentation results using our custom LoRa testbed support our claims that RLNC is a very suitable code for FUOTA.

Chapter 4

Network Coding for Firmware Updates

Network Coding(NC) is a broad term for network-optimized packet coding, which improves network robustness and efficiency[37]. Transmitted packets are fragmented, transformed, and forwarded, so the output packets are more suitable against faulty network behavior. Multiple aspects define a network's behavior. For example, the number of end devices, whether data is multicast or the probability of packet loss, can bottleneck network performance. A firmware payload is too large for the LoRa network to process all at once. Such a firmware payload is named a Binary Large Object (blob).
This chapter explains the fundamentals of firmware dissemination in Section 4.1. Next, the principle of sending a generic blob using the NC scheme RLNC is presented in Section 4.1.3. This provides insights why specifically RLNC is a powerful NC scheme for firmware dissemination. Finally, Section 4.3.2 presents a mathematical framework for evaluating configurations of RLNC, which is applied in Chapter 5.

4.1 Firmware Dissemination

Firmware dissemination is the process of fragmenting a firmware payload into smaller fragments in sequence, transmitting these to all receivers, and recovering from imperfections like errors or erasures(explained in Section 4.1.1). The targeted devices must receive all original fragments. Otherwise, the new firmware can not be adequately reconstructed and can not be trusted to work as intended.
Fragmenting the firmware needs to be done while taking into account the limitations of the intended network like physical layer constraints, message scheduling bottlenecks, and MAC layer frame overhead, to mention some examples. The firmware of size U_{firmware} symbols is sent with fragments of size F symbols, as shown in Equation (4.1)), which results in N_f fragments. Following, the total session duration T_{firmware} defined in Equation (4.2) dependent on both N_f and the transmission period T_{packet} of each individual fragment. As will become clear in Section 4.3.3, the FUOTA session takes very long. Therefore, it is

crucial to keep T_{firmware} as low as possible (Equation (4.2)).

$$N_f = \left\lceil \frac{U_{\text{firmware}}}{F} \right\rceil, \tag{4.1}$$

$$T_{\text{firmware}} = N_f T_{\text{packet}}. \tag{4.2}$$

Techniques exist to reduce N_f before dissemination. So-called *difference algorithms* take advantage of changes in the target device's firmware because an update is a successor of existing firmware. By comparing an already installed firmware version with an updated firmware, redundant data is removed resulting in a *diff* with N_{diff} fragments where $N_{\text{diff}} \leq N_f$ holds. Other algorithms like RSync[38], RMTD[39] and R3Diff[40] can reduce the blob size even further by applying compression across the disseminated fragments. However, since these optimized algorithms have been developed to full maturity, it is outside this thesis' scope to reduce and compress the firmware size itself[25]. Therefore, the firmware with size U_{firmware} is considered a constant factor defined by the disseminated application firmware.

4.1.1 Encoding and decoding

An *encoder* accumulates fragments to generate a stream of coded fragments. The receivers must collect a certain number of these encoded fragments to be able to reconstruct the original payload. A *decoder* algorithm is implemented on each receiver. Such a decoder is designed to be tolerant against fragment faults caused by poor network conditions on the packet level. Two such faults are packet errors and packet erasure. These two faults can be prevented by appending *reparation symbols* to each packet or transmitting *redundant packets*. The decoder takes advantage of these two types of redundancy with the highest possible decoding efficiency. This efficiency is defined by the bits required to decode successfully compared to the transmitted bits. Decoding efficiency is independent of the number of transmitted packets over the network. For example, suppose two packets are required for decoding successfully, and four were transmitted because of two erased packets. In that case, the decoding efficiency will still be 100 % as only no additional information was necessary. Therefore, since packet erasure can still cause decoding failures for a perfect decoder, decoding efficiency is insufficient to determine decoder performance. The *decoding probability* is a complete statistic, because it incorporates a random packet error/erasure model, redundancy, and decoding efficiency[33, 29]. The decoding probability is the first step to evaluating the decoder for static or dynamic network conditions.

4.1.2 Fixed-rate and rateless coding

Relevant to the context of this work, two main classes of coding exist. *Fixed-rate coding* adds redundancy based on a fixed code rate. The stream of fragments is extended with redundant information, which constitutes a fixed ratio compared to the original unencoded stream. This ratio is determined beforehand and can not be changed without first exhausting or terminating the current stream. Another variant, named *rateless coding*, does not require determining and exchanging the code rate beforehand as the stream can be extended on-the-fly.

In other words, the maximum number of encoded packets is not limited by a code rate and, therefore, the stream can be extended without having to flush the decoder(reset). Concluding, rateless codes provide flexibility.

4.1.3 Network Coding

Network performance is primarily evaluated by throughput, the number of data packets it can exchange between transmitters and receivers for a certain duration. Data packets alone are not sufficient for the network to stay operational. A network must exchange control packets to ensure proper synchronization and dynamic adjustments to counteract network side-effects (refer to Section 2.2.2 for LoRaWAN control packets). These two packet classes are called the data plane and control plane.
A NC algorithm makes the network more robust by combining the *control plane* and *data plane* packets. Fewer control packets must be exchanged to disseminate a blob payload. Less ACK control packets are required to verify the fragments have arrived without faults. ACK control packets provide feedback from end devices to compensate for packet erasure. The transmitter can then compensate by transmitting redundant fragments. At first sight, it seems that this solves the issue of erasure, but sadly it does not scale properly with network size. Also, additional feedback communication lowers the bandwidth for transmitting relevant data - this is especially clear in LoRa. This effect is visible in a large network through the scaling phenomenon called *ACK-implosion*. More added devices in the network require more control frames to be sent, which 'implodes' or reduces the network throughput on the data plane.
However, the RLNC scheme is a significant improvement that reduces the amount of end device feedback in case of network faults.

4.2 Random Linear Network Coding

RLNC is a rateless erasure coding scheme sharing similarities with fountain codes[29]. Theoretically, a limitless number of encoding symbols can be generated to compensate for packet erasure (refer to Section 4.1.2). Secondly, it is a *linear code*, which means that linear operations are used on a specific mathematical field to encode and decode the fragment symbols. In the case of RLNC, the field is a finite field called Galois Field (GF)[30], which is defined by a specific degree and irreducible polynomial[1]. Thirdly, encoding helper symbols are selected in a pseudo-random fashion within this finite field, which is why RLNC is classified as a *random code*. Finally, it is a *network code* which shows itself in two ways. Firstly, in case of packet loss, receiver nodes only need to give feedback on how many packets were lost, not specifically which packets. Secondly, intermediate nodes *recode* fragments without knowing the original unencoded fragments using independently generated random encoding symbols. It depends on the network topology whether intermediate nodes are present. However, the LoRa network topology from Section 2.2.3 shows that the single-hop network

[1] Mathematical operations on the symbols in a finite field always results in outcomes within the field. No overflow or carry is required as the modulo operator reduces the outcome within the field by using the irreducible polynomial. For order $q = 2^k = 256$ (degree $k = 8$), or 256 elements in the finite field, the irreducible polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (hex: 0x11D) is used.

does not naturally have such intermediate nodes. For this reason, this work will focus on encoding and decoding only.
Next, the finite field is explained, a mathematical construct used by the RLNC encoding and decoding algorithms. Note that the decoder algorithm is of more importance as it is implemented on end devices with limited processing power.

4.2.1 Coding system overview

Figure 4.1 shows the generic approach to a RLNC encoder-decoder system between two hosts. A large payload is transmitted from the encoder system to a decoder. To achieve that the firmware is *pre-processed* using these transformations:

1. Fragmentation (Section 4.1.3)
2. Finite-field mapping (Section 4.2.2)
3. Generation segmentation (Section 4.2.3)

After pre-processing, each generation is completely precoded using a *RLNC generation-based encoder*:

1. Randomized encoding vectors selection (Section 4.2.4)
2. Generation fragments and redundancy precoding (Section 4.2.4)
3. Encoded fragment transmission

Finally, the transmitted encoded fragments are put into the decoder one by one. For each new fragment decoding can be done to progress further.

1. Perform fragment decoding by matrix row-reduction to Row-reduced Echelon Form (RREF)
2. Verify decoding success and store generation in non-volatile storage

As fragmentation was previously covered (it is not unique to RLNC), finite fields and generation segmentation are covered next.

4.2.2 Finite field mapping

A Finite Field (Galois Field) represents a limited numerical system like a digital system. The advantage of finite fields is that overflow does not occur. All encoding and fragment symbols used for RLNC are represented within the Galois finite field. Each symbol is of the size as the field order q. Relevant options for q are 2^0, 2^1, 2^2, 2^4, 2^8 and 2^{16}. This symbol size can, therefore, be represented in bits (binary numbers). As is shown in Section 4.3.2 that $q = 256$, noted as GF(2^8), is accurate enough for decoding with RLNC. This accuracy is practical because one or more 8-bit symbols can be nicely packed into a byte, two bytes, or 4 bytes. Also, the addition, subtraction, multiplication, and division field operators can be optimized for $q = 2^8$ through the use of Look-up Tables. Beyond 16-bits, the time complexity of operations defined on the Galois field becomes quite large. For $q = 2^{16}$ optimization using LUTs is no longer feasible

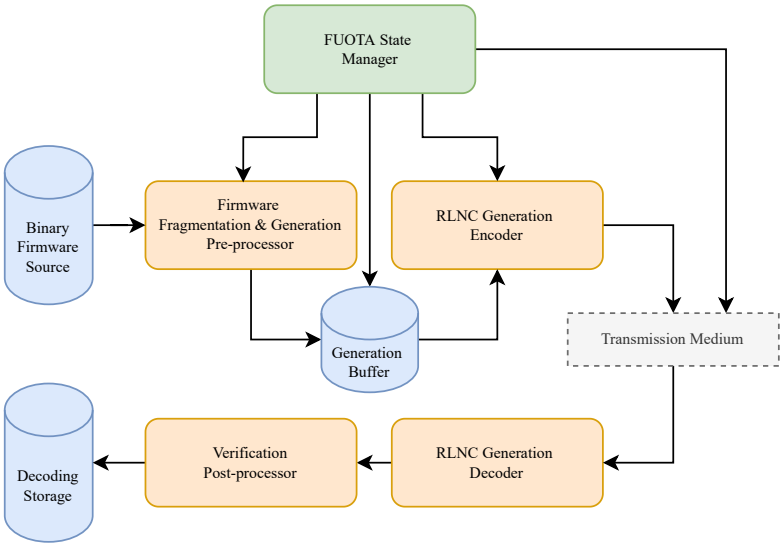


Figure 4.1: **RLNC system overview for FUOTA with fragmentation preprocessor, generation buffer, and generation encoder controlled by FUOTA State Manager. The dashed transmission medium represents an arbitrary network topology to one or more receivers. Transmitted fragments are decoded by a RLNC decoder and verified before being stored.**

Operation	Time Complexity	Function
Addition	$O(n)$	$g(x) + h(x)$
Subtraction	$O(n)$	$g(x) - h(x)$
Multiplication	$O(n^2)$	$g(x) \cdot h(x)$
Inversion	$O(n^2)$	$g(x)^{-1}$

Table 4.1: **Trivial (un-optimized) time complexity for finite fields without use of multiplication or division LUTs, where $n = \lceil log^2(q) \rceil$ or the degree of the field[41].**

as the memory required to store this is too excessive. Table 4.1 presents the time complexity of operators used in the RLNC code.

Finite fields and their operators are extensively used in cryptographic (RSA), hashing (SHA), and parity check (CRC) functions. For this reason, it is outside the scope of this thesis to discuss optimizations for the Galois Field through LUT. The RLNC algorithm works correctly for $q = 2^8$ by calculating intermediate results in online fashion (without help of LUT).

Concluding, the definition of a symbol in the Galois Field depends on the degree of the field. Symbol mapping is necessary to convert from a binary system of a certain degree (in bits). For example, a symbol or word on a 32-bit architecture system must be split into four GF(2^8) symbols. Another better approach would be to work from with bytes (8-bit) directly instead, so the mapping becomes trivial as both a byte and GF(2^8) can represent the same symbols. This approach is assumed in this thesis.

4.2.3 Generation-based encoding

Section 4.1 introduced the concept of fragmenting the firmware payload and Section 4.2.2 showed symbol conversion to GF(2^8). Generation-based RLNC goes one step further by introducing segmentation called a generation[2]. After fragmentation completes, the fragments are grouped into generations of G fragments in size. The RLNC encoder iterates each generation separately. Uncoded fragments are put in the rows of an encoding matrix M_E. Similarly, encoded fragments end up in a decoding matrix M_D after reception. Both M_E and M_D are augmented matrices with on the left side encoding vectors and the right the original fragment rows. The only difference between M_E and M_D is that they are maintained by the encoder and the decoder, respectively.

There are clear benefits to using generations:

- The decoder needs G linearly independent generation fragments to be able to decode successfully.
- The decoder allocates much less memory for the decoding matrix M_D.
- Decoding the generation takes few computations, and the workload is split into small pieces.
- After decoding, the end device can save power until the next generation starts.

When a generation is being transmitted over the network, selecting a certain number of redundant fragments is possible to cope with packet erasure (Packet Error Rate (PER)). Since packet erasure can only be estimated or predicted, this is a *feed-forward* instead of a *feedback* approach. If one or more receivers could not decode the generation, the estimate of the network packet erasure was incorrect, and some form of feedback is required to at least indicate this to the transmitter (encoder). Multiple situations are possible:

1. The redundancy was not enough to cope with uniform PER.
2. The redundancy was exactly right (or too high), and the whole network has completed successfully.
3. Temporary packet erasure (burst loss) caused multiple packets to be erased, causing generation decoding to fail, but these bursts do not occur often nor uniformly.

Case 1 is solved by statically increasing redundancy beyond the threshold of average packet erasure. This requires a small number of feedback packets from the receivers in the network at the beginning or some form of network throughput estimation. Such a configuration is named *block-based RLNC*[42].
Case 2 seems to be what is desired, but it is not likely that this kind of uniform packet loss occurs in reality. Also, if excessive redundancy is applied, much time could be spent on packet transmission without extra information.
Case 3 is called *burst loss* and this is much more complex. It can be solved in

[2]The term generation is often referred to as a block or page in more generic OAP and IoT works for blob payload dissemination[25].

two different ways. By increasing the redundancy threshold beyond the worst-case PER, a lot of redundant packets will counteract the burst loss. However, when no burst loss is present, the redundancy is overestimated and wasteful. Therefore, a second solution to *Case 3* is adjusting redundancy dynamically with feedback from receivers. Consequently, redundancy for moments without burst will not waste as many packet transmissions. Also, this feedback can be used to *extend, restart or terminate* generations if needed to be able to either guarantee successful completion or to gain knowledge and statistics about what caused the generation failure(s). However, great care is needed to avoid *ACK-implosion* because of too much feedback communication, especially for a large network.
It can be deduced from these three disjoint cases that an evaluation method is needed to estimate and correct the performance of one or more RLNC decoders present on the network. The decoders synchronously decode the same generation but will experience different packet losses due to independent channel characteristics. Consequently, the redundancy must be adjusted based on performing this evaluation statically or dynamically. One solution for this is *sliding-window RLNC*. This variant has been developed to adjust generation (or redundancy) sizes dynamically. *Sliding-window RLNC* has been shown to outperform *block-based RLNC* and classical feedback mechanisms similar to ARQ[42]. This RLNC extension requires extra control plane packets compared to block-based RLNC. The core working principle is the same. For that reason, block-based RLNC encoding algorithm is introduced first. The sliding-window extension is treated in Section 5.2.

As previously mentioned, encoding is performed on matrix M_E, which is an *augmented matrix* with encoding vectors on the left side. The following section explains how these vectors are generated using a specific choice of Pseudo-random number generator (pRNG) to ensure linearly independent vectors.

4.2.4 Randomized encoding

The RLNC encoding algorithm is explained now. First, the uncoded state is presented, which is the desired state after decoding. Also, it is explained how to evaluate whether decoding will fail. A precise mathematical basis is provided for understanding the encoded state. The essential arithmetic to achieve this encoded state is explained.
Beforehand, a generation is assembled and buffered as explained in Section 4.2.3 into a matrix $\mathbf{F}$ with symbols $\mathbf{F}[i,j] = f_{ij}$ in GF(2^k). In this notation i is the fragment index (row) and j the fragment vector index (column) as shown in the *uncoded or decoding matrix* M_D from Figure 4.2. It is observable that the left submatrix I is an identity and, therefore, a linearly independent matrix. Note that this is also the desired state after decoding has been completed. If, after receiving sufficient linearly dependent fragments, the encoding matrix does not reduce to an identity matrix (unity symbols on the diagonal), the matrix has become corrupted. This topic is revisited in Section 4.2.6. Alternatively, if too many fragments were linearly dependent, the matrix rank is smaller than G. This method is used to determine decoding success.

Equation 4.3 is the encoding function which produces the *encoding matrix* M_E with (left) submatrix $\mathbf{E}$ and (right) submatrix Φ in Figure 4.3. Each row of $\mathbf{E}$ is called the *encoding vector* of the associated encoded fragment row in

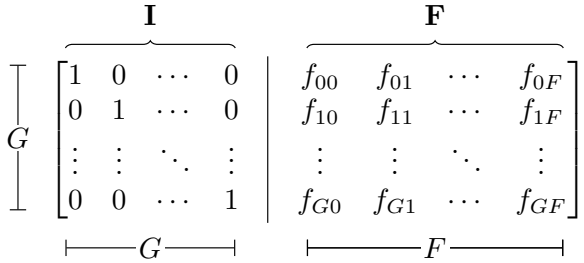


Figure 4.2: **RLNC *uncoded/decoding* matrix M_D. Note the submatrix I is an $G \times G$ identity matrix (RREF) and the fragment submatrix F is a rectangular $G \times F$ matrix.**

Φ. Elements in submatrix **E** will be randomized and the end result is used to transform submatrix **F** into submatrix Φ. Note that redundant rows are generated which results in R encoded rows. This is possible, because the coordinates $\Phi(i, j)$ are essentially linear combinations of symbols f_{ij} and, therefore, more rows can be generated than provided as input to **F**.

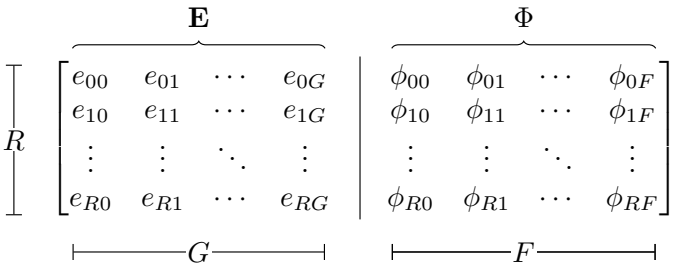


Figure 4.3: **RLNC *encoding* matrix M_E. Note the encoding submatrix E is an $R \times G$ matrix and the coded fragment submatrix Φ is a $R \times F$ rectangular matrix.**

$$\begin{aligned}\Phi(i,j) &= \mathbf{E}(i,0)\mathbf{F}(0,i) + \mathbf{E}(i,1)\mathbf{F}(1,i) + \cdots + \mathbf{E}(i,G)\mathbf{F}(G,i) \\ &= e_{i0} \cdot f_{0i} + e_{i1} \cdot f_{1i} + \cdots + e_{iG} \cdot f_{Gi} = \phi_{ij}, \\ &\text{where } 0 \leq i < R \text{ and } 0 \leq j < G.\end{aligned} \tag{4.3}$$

The encoded generation size becomes R defined in Equation 4.4. The redundancy coefficient δ is the ratio of added packets to the generation. Therefore, R always is greater or equal to the original generation size G.

$$R = (1 + \delta)G \geq G \tag{4.4}$$

The encoded fragments found on the rows of Φ must be transmitted over the network. The encoding vectors on the rows **E** could be appended to each fragment packet as well, which is called *explicit transmission*[37]. Alternatively, the encoding vectors can be randomized using a deterministic type of pRNG (pseudo-random number generator). The pRNG seed belonging to each fragment is appended to each fragment to synchronize the decoders, which is

called *implicit transmission*[37]. The implicit type of randomization requires the decoder software to implement the same pRNG functionality as the encoder software. Also, the decoder must be able to reset the internal state of said random-number generator to the state provided with each fragment (a seed value). This reduces overhead as fewer data symbols need to be sent. This optimization becomes especially apparent when the encoding vectors are nearly the same size as the encoded fragments.
Only a subset of pRNG types is suitable for this code. The requirements and choice of pRNG are explained next.

4.2.5 Pseudo-random number generator

Multiple properties classify a pRNG, which are prioritized from high priority (top) to low priority (bottom) for RLNC as follows:

1. Cyclic length or period (and internal state size)
2. Repeating/non-repeating outcomes
3. Includes/excludes symbol 0
4. Memory/code footprint
5. Computational complexity
6. Uniform randomness
7. Cryptographic security

The decoding phase of RLNC is not of a cryptographic kind whatsoever, and it should not be used that way (**7**). Therefore, statistical unpredictability[3] of the pRNG is not of importance, and it need not be perfectly uniformly random (**6**). The memory space and code footprint complexities should be matched with the used hardware and firmware, but because the intended application has a low sampling rate, it is not of great importance (**5**). The same argument extends to computational complexity (**4**). The random symbol source could include the symbol 0 in its possible outcomes to create more unique encoding vectors, which means that more linearly independent encoding vectors can be generated. In general lower probability of linearly dependent encoding vectors is very desirable, so the 0-symbol should generally be included (**3**). Furthermore, the pRNG should be able to repeat the same elements in the Galois Field for the same reason of desiring independent encoding vectors (**2**). This aspect is discussed further in Section 4.3.2 where the decoding failure probability due to linear encoding vectors is treated. Finally, the simplest and most important property of the pRNG is the so-called cycle period, which, if it is too small, is important because of possible faulty pRNG behavior (**1**).
All pRNG have a bounded cycle. When the number of samples drawn from the generator exceeds the cycle period, theoretically, the pRNG design can be reverse-engineered. This is not important for the RLNC coding scheme (refer to property **7** about importance of cryptographic strength). However, note that the RLNC encoder draws many symbols from the generator simultaneously while

[3] An unpredictable pseudo-random number generator is still deterministic.

encoding a generation. With a short period, a pRNG can only generate a limited number of unique encoding vectors. Therefore, the cycle should be great enough to guarantee a sufficient number of unique sequences of symbols, rather than just individually random symbols. In conclusion, the design and thus inner state size of the pRNG must be considered when selecting the generation size of the RLNC.

Three types of pRNG were considered: Linear Feedback Shift Register (LFSR), Linear Congruential Generator (LCG) and XoShiRo (xor-shift-rotate). These are quickly compared.
LFSR is simple to implement with XOR-arithmetic[4], computationally highly efficient and can be scaled up to desired period. This type of generator only generates values in its cycle once and excludes the symbol 0. Therefore, property **2** and **3** do not meet the requirements set before. Finally, the cycle period is equal to $2^n - 1$ with n-bits state elements, which means that the 32-bit state will result in a high period.
LCG is simple to implement using modulo, multiplier, and increment operators. This efficient design can generate the same symbols, including the symbol 0, so all requirements are met. However, it is considered a very poor pRNG implementation based on property **7** (statistical strength)[43]. The next option is much stronger.
XoShiRo is a novel, very robust and efficient scrambler inspired by filtered LFSR[44]. It uses XOR and rotation arithmetic. Many variants can be picked to meet a specific set of requirements. Each XoShiRo type of pRNG has been proven to be statistically robust with high cycle period length[44].
For referential purposes, the XoShiRo32** (XoShiRo, 32-bit state, 8-bit output) was picked for the RLNC encoder and decoder, which is not a standard variant. Four states of 8-bit are combined to a total of 32-bit internal state with a high period ($2^{32} - 1$). The 8-bit output symbols perfectly matches the finite field GF(2^8) (Section 4.2.2).

Concluding, the choice of pRNG is crucial for determining the quality of the RLNC encoding vectors. The probability of linearly dependent encoding vectors can be reduced by high pRNG period and allowing for repeated symbols, including the symbol 0. The next topic is the inner working of decoding and decoding matrix rank determination to test for completion.

4.2.6 Decoding algorithm

The decoding matrix M_D is periodically filled with newly received generation fragments. Each time a new RLNC fragment packet is received, the firmware unpacks the packet. The packet contains the generation index, the *implicit randomization seed* for the encoding vector, and the fragment symbols. Multiple actions are performed by the RLNC decoder to update the M_D:

1. Check that the fragment generation index is equal to the decoder generation index; otherwise, reset and flush the decoder.

[4]XOR is one of the fundamental types of digital arithmetic. It performs an exclusive-OR operation on two inputs.

2. Check that the decoding matrix has full rank and the identity matrix is not corrupted. If successful, flip the success state[5].

3. Use the fragment random seed to generate G encoding symbols (the encoding vector).

4. Insert the reproduced encoding vector and encoded fragment into M_D

5. Perform row-reduction by row-swapping and pivotting using GF(2^8) finite field arithmetic.

Decoding has four possible states after each fragment has been processed: `Completed`, `NoProgress`, `Progress` and `Corrupted`. The `Completed` state is determined by validating the encoding submatrix $\mathbf{E}$ in the decoding matrix M_D. As the decoder has previously executed row-reduction, it is guaranteed that the matrix is in RREF. Therefore, completion is checked by checking the diagonal of $\mathbf{E}$ for ones and zeroes on all other places in the submatrix. The matrix is not full rank if zeroes are detected on the diagonal. Consequently, decoding cannot complete. If the decoder rank did not increase compared to the previous round, the state switches to `NoProgress`. The encoding vector was linearly dependent and did not carry enough new information to progress. However, if the rank increased by one, the state `Progress` is flagged. Finally, if the decoder processes a corrupt fragment or seed, the state is changed to `Corrupted`. The decoder will have to reset its state and lose its progress because parts of the decoding matrix have become corrupt. Because this is such a costly event, a mechanism is needed to prevent fragment corruption. This mechanism is provided by payload CRC as an integrity check.

Summarizing, RLNC generation decoding should succeed when G fragments are received, unless linearly dependent encoding vectors were selected or packet erasure occurred. To compensate for this redundancy factor δ is applied on G resulting in R fragments per generation. The next section provides mathematical equations to get expected performance using different settings of such parameters.

4.3 Decoding performance

This section provides mathematical tools to evaluate the required memory for a decoding matrix, the decoding latency, the session duration, the decoding probability, and the network recovery probability. These tools are essential for evaluating the RLNC decoder robustness against packet erasure. The equations applied in the evaluation are presented in Chapter 6.

4.3.1 Decoding matrix size

The encoding vectors in sub-matrix $\mathbf{E}$ have a length equal to N_g and there are R encoding vectors in total. Therefore, the encoding matrix dimensions are ${N_g}^2$.

[5]The testbed decoder will continue decoding after completion. In reality, the decoder skips new packets after decoding successfully, and the system switches to energy-saving deep sleep until next-generation fragments are transmitted.

Each fragment is of size F and there are R fragments in each generation, but after each fragment is received row-reduction is performed. When more than G fragments are received, the decoding matrix is full rank and new rows will not result in new information. Therefore, the decoding matrix needs only be allocated with a capacity of G rows and $G + F$ columns (see Figure 4.2).

$$S_{\mathrm{D}} = G \cdot G + G \cdot F = G(G + F) \tag{4.5}$$

Now, the generation size is related to the configured generation count N_g:

$$N_g = \left\lceil \frac{N_f}{G} \right\rceil, \tag{4.6}$$

and the total exchanged amount of data will exceed the firmware update size (albeit only slightly). The following equation shows how generation size, fragment size and generation count relates to the firmware size:

$$U_{\mathrm{firmware}} \leq F \cdot G \cdot N_g. \tag{4.7}$$

It seems from Equation (4.5) that the generation size N_g has quadratic influence on the size of the decoding matrix. Note, however, that changing the generation size will inversely change the fragment size. This effect is visible in plots of Figure 4.4, which shows the linear relation of decoding matrix size to fragment size and generation size.

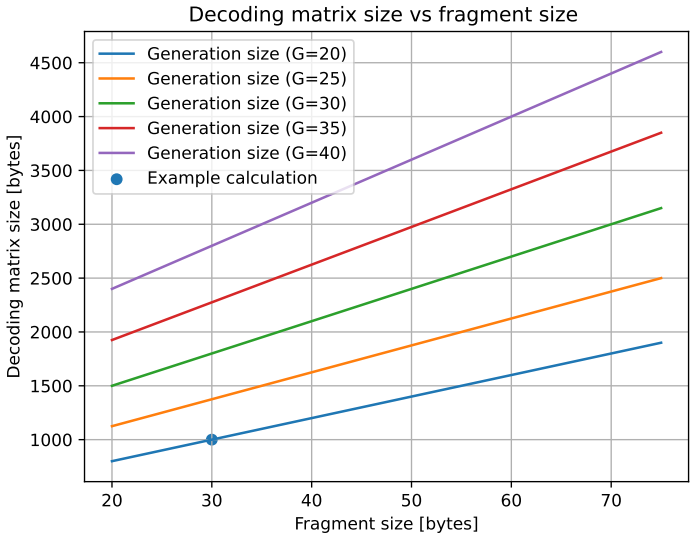


Figure 4.4: **Allocated RLNC decoding matrix size for different configurations of generation size G and fragment size F.**

If practical, computational, hardware and radio limitations are ignored, the parameters N_f and G are free to choose. The next example calculation shows how this works.

Calculation example A firmware blob $U_{\text{firmware}} = 60\,\text{kB}$ is transmitted using $N_f = 2000$ fragments (F=30 bytes per fragment). Assuming $G = 20$ the decoding matrix will have a size of $S_D = 1000$ symbols or 1 kB in case each symbol is a byte. This approach is useful for validating if the memory usage will not exceed the available dynamic RAM on a target embedded device. Figure 4.4 shows this calculation result and generalizes it for fragments between 20 and 75 symbols in size and generations between 20 and 40 in size.

4.3.2 Decoding probability

The RLNC decoding failure probability is dependent on two factors: packet erasure rate ϵ and probability of linearly independent encoding vectors. This probability is defined as $P(m, n, k)$ in Equation (4.8)) where m is the transmitted fragment count, n the generation size and k the field degree ($k = \log^2(q)$ [32, 30].

$$P(m, n, k) = \prod_{i=0}^{n-1}(1 - \frac{1}{k^{m-i}}) \tag{4.8}$$

In other words, Equation (4.8) shows the probability of decoding success due to linearly independent encoding vectors. This function is 0 for all $m < n$, which makes sense because decoding can only succeed when n frames are received with a probability higher than 0. One could state from Equation (4.8) that the probability of decoding increases monotonically with increasing n. However, this means the threshold of each generation is increased and, therefore, m must be increased as well. Concluding, the decoding probability due to non-linear packets does not become lower for larger generation size n, and only the influence of k must be further evaluated.

Equation (4.9) extends this decoding probability by including decoding failure due to packet erasure. This packet erasure is modeled here with Bernouilli probability ϵ which results in a binomial Probability Mass Function (PMF)[32]. This PMF is cumulatively summed to acquire the Cumulative Distribution Function (CDF) using the masking feature of Equation (4.8).

$$P_{\text{fail}}(G, R, \epsilon, k) = 1 - \sum_{m=G}^{R} \binom{R}{m} \epsilon^{R-m}(1-\epsilon)^m P(m, G, k) \tag{4.9}$$

Figure 4.5 shows multiple plots using Equations (4.8) and (4.9). One of the plots is the perfect finite field decoder, which shows decoding failure probability based purely on packet erasure. This is achieved by forcing the probability $P(m, G, k) = 1$ of Equation (4.8) for $m \geq G$. Compared to such a decoder, a GF(2^8) decoder has nearly no additional probability of failing due to non-linear packets. Decoders with GF(2^4) deviate only slightly, but GF(2^2) and GF(2^1) show undesirable additional decoding failures. Therefore, $q = 2^k = 2^8$ or higher should be preferred from the perspective of maintaining high decoding probability.

4.3.3 Decoding latency

The average decoding latency defined as the time step required to encode a blob, transmit the encoded fragments to one or more receivers and decode them.

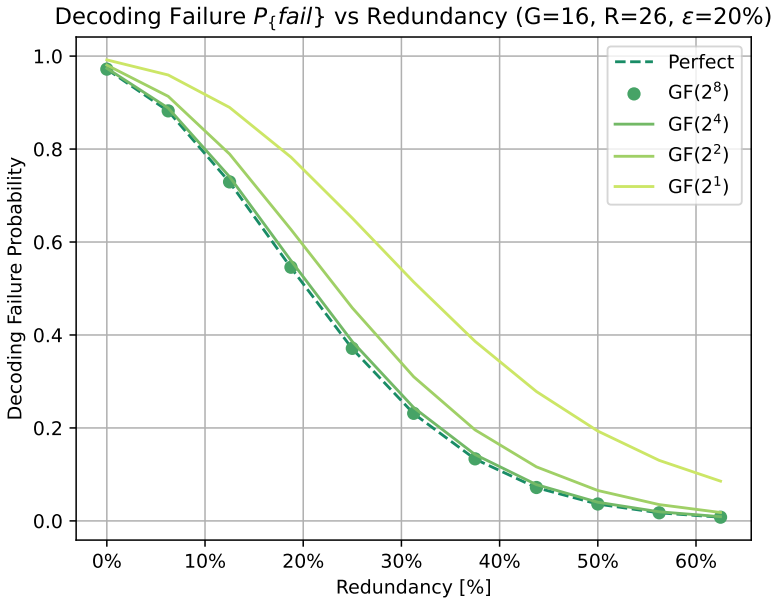


Figure 4.5: **RLNC decoding failure for different Finite-Field degrees (1, 2, 4 and 8). GF(2^8) is nearly perfect.**

Equation 4.1 previously defined N_f, but now it is refined with finite field symbol mapping of Section 4.2.4 in mind as shown in Equation (4.10). Note that with $q = 2^8$ or GF(2^8), the equation simplifies to Equation (4.1).

$$N_f = \left\lceil \frac{U_{\text{firmware}}}{F} \cdot \frac{2^8}{q} \right\rceil, \tag{4.10}$$

R in Equation (4.4) can then be redefined:

$$R = \left\lceil \frac{N_f}{N_g} \right\rceil (1 + \delta). \tag{4.11}$$

This leads to the following bounds for the generation *decoding latency*, which is defined for decoding success only

$$N_g G \leq T_{\text{decode}} \leq N_g R, \tag{4.12}$$

and complete firmware update duration is then bounded as follows:

$$T_{\text{packet}} \cdot N_g G \leq T_{\text{firmware}} \leq T_{\text{packet}} \cdot N_g R. \tag{4.13}$$

Calculation example A firmware blob 60 kB of fragments of each 30 bytes each in GF(2^8) would result in 2000 fragments according to Equation (4.10). Assuming generation size $G = 20$ we require $N_g = 100$ generations from Equation (4.6) without redundancy.

The packet erasure ϵ must be taken into account. Referring to Figure 4.5, more than 63 % fragment redundancy δ is required to guarantee successful decoding for a channel with $\epsilon = 20\,\%$ (uniform distribution). Using δ, each generation sends $R = 32$ fragments. The whole generation will take between 20 and 32 fragments. If one fragment is transmitted every $T_{\text{packet}} = 1\,\text{second}$, the decoding latency lies between 20 sec and 32 sec (Equation (4.12)).
With a total fragment count of 3200 , the whole decoding session will roughly take between 34 minutes and 54 minutes. This is an optimistic example, indicating that firmware update sessions for LoRa take long to complete.

Basic tools for the analysis of decoding a single generation have been provided. Multiple generations must succeed, and often multiple end devices will perform this decoding process simultaneously. Section 6.2.3 presents results the probability distribution analysis in *a network of independent end devices*. A testbed has been developed, which verifies this distribution, explained in the next chapter.

Chapter 5

Testbed Design

The testbed described in this chapter has been developed to validate and gain insights about the RLNC FUOTA application layer for LoRa. As will become clear, the capabilities of the testbed are a superset of collecting decoder performance statistics. Static and dynamic external factors impact coding performance on each device in LoRa. Channel conditions, time synchronization, and MAC dynamically change the probability of decoding the generations in our RLNC system. Critical factors (f.e. encoding parameters or LoRa, software, and hardware limitations) are evaluated with this testbed presented in this chapter. The software for the LoRaConfigurator terminal host is published online[45][1]. Also the end device firmware, plots and measurements are available online[46][2]. This leads to the results presented in Chapter 6.

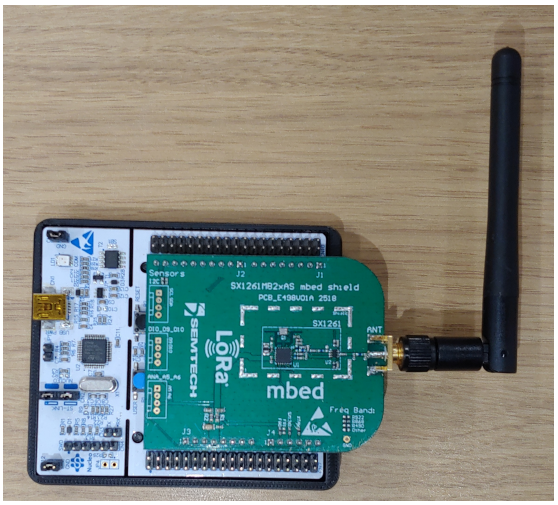


Figure 5.1: **SMT32 Nucleo64 F446RE end device with 3D-printed protective case, SX1261MD2BAS LoRa shield, and** 868 MHz **antenna.**

Figure 5.1 shows the STM32 Nucleo64 F446RE device equipped with SX1261MD2BAS LoRa shield and 868 MHz antenna[47, 19]. This device has been flashed with

[1] `https://github.com/davidzwa/LoRaConfigurator`
[2] `https://github.com/davidzwa/F446_ProtoBufDevice`

firmware capable of acting as both *gateway* and *end device*. However, it is common in LoRa networks to equip a gateway with a collector radio. A LoRa collector can operate at multiple SF settings simultaneously, which the SX1261 radio shield cannot achieve. However, using the same device as the end device and gateway pays off in its simplicity and, most importantly, in the ability to assign multiple roles without having to change network deployment. Therefore, only single-channel devices are used for this testbed. Figure 5.2 illustrates two such end devices in a symmetrical setup.

The two largest flash memory pages on the F446RE chip of 128 kB are not used by application firmware. Therefore, these pages are available for other purposes. This chapter assumes that the *first page* is used for storing experimental measurements and the *second page* for providing encoded RLNC configuration and fragments compiled into a small blob less than 128 kB in size.

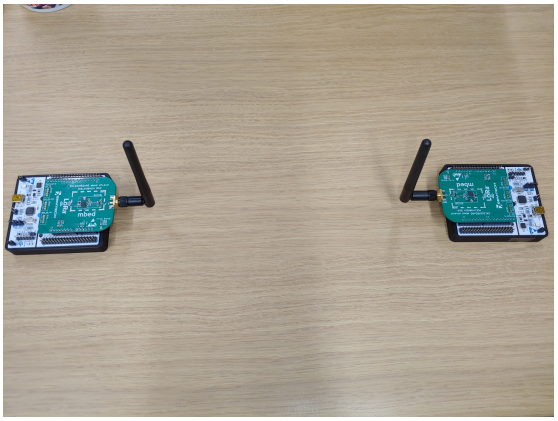


Figure 5.2: **An example of two F446 end devices placed close together.**

Furthermore, this chapter explains the possible testbed configurations which are required to validate the system components. A PC application has been developed named LoRaConfigurator (LRConf), which can trigger experiments and store associated measurements. The wireless LoRa and wired UART interfaces between LRConf and the end device are explained in Section 5.1. Secondly, the message flow of an experiment testing the RLNC coding application is presented. Thirdly, Section 5.2 shows how measurements are processed to get estimates for the PER in real LoRa experiments. Finally, Section 5.3 provides an overview of constraints, practical experimental observations, and their inferences to integrate RLNC with LoRa and LoRaWAN. This provides the basis for the next Chapter 6, where the performance of this novel FUOTA application is evaluated.

5.1 Wireless and wired interfaces

Two end devices communicate over the wireless LoRa interface without any additional commands required to synchronize their exchanges. This physical link between two devices is the most primitive possible method of exchanging frames. Because the LoRaWAN MAC-layer adds quite some message traffic to set up, gain and defer channel access (Appendix A), the testbed will avoid this

additional control plane[11]. Instead, the testbed has its minimal control plane with capabilities specifically designed for the intended tests. The following list specifies the capability for transmission (TX) or reception (RX) handlers of each end device's firmware:

1. (TX) Identify receiver using Unique Hardware Identifier (UHI)
2. (TX) Mark message as unicast or multicast
3. (TX) Add sequence number for message ordering
4. (RX) Append Signal-to-Noise Ratio (SNR) and Received Signal Strength Indicator (RSSI) provided by the radio to each message
5. (RX) Drop received unicast messages meant for another device
6. (RX) Store sequence number, SNR and RSSI in Flash measurement storage
7. (RX) Drop packets using a uniform or burst erasure model (pseudo-random, see Chapter 6)

Next, the basic building blocks *source* and *sink* are presented which build up different experiments using the capabilities mentioned above.

5.1.1 Configurations with Source and Sinks

One *source* (transmitter) sends messages to one or more *sinks* (receivers). Each received message results in a data point, and all points form an experimental data set. Both sinks and sources can be combined into different *Configurations (Configs)* shown in Figure 5.3. Each Config shines a different light onto the network it consists of. Often there are multiple parameters associated with the evaluations possible with each configuration. Therefore, experiments iterate relevant parameters, which are integrated into the experimental control plane. Each control message sends the specific set of parameters to the sink either directly using a wired link or indirectly using a *proxy end device*. It is interesting to note that the configurations have been designed by adding more and more influencing factors to the network. For example, the most straightforward configuration, Config **C**, excludes wireless communication by using a loopback message handler setting, which results in the ability to evaluate every feature except for the LoRa interface. This method of progressively evaluating additional influencing factors while excluding others gives the testbed a compelling characteristic.

Figure 5.3 shows practical setup configurations with LRConf, the sinks, and sources. Most options can easily be changed from a unicast to a multicast mode or vice versa. The configurations are presented as primitive building blocks to work with. Interesting capabilities are visible in configurations **B**, **C**, **E** and **F**, which are explained in more detail next.

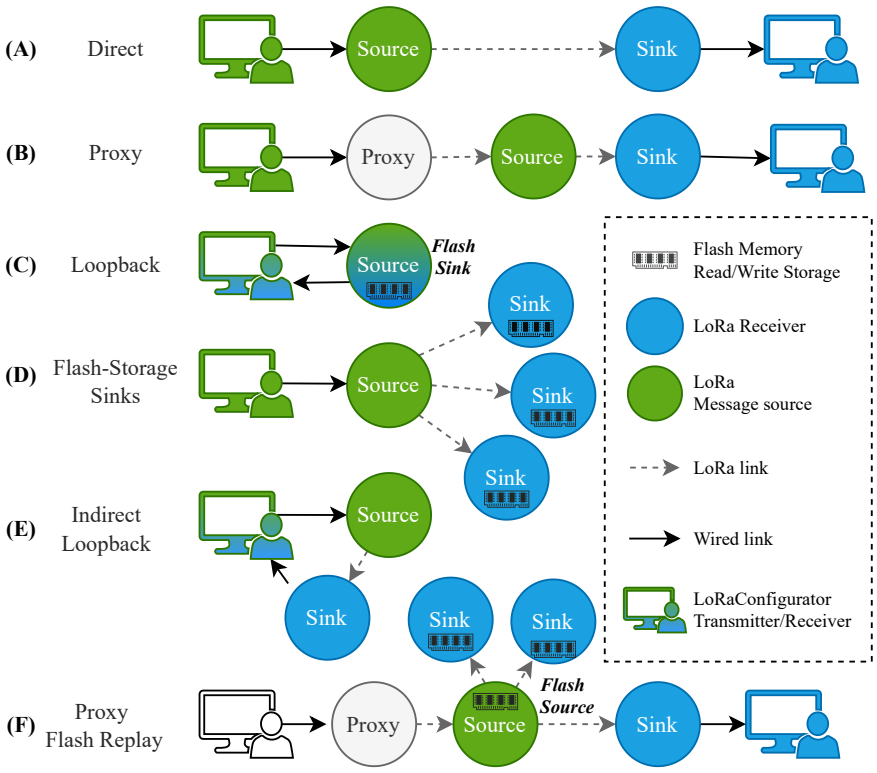


Figure 5.3: **LRConf basic setup possibilities.**
Config A Direct source-sink with data storage on another LoRaConfigurator.
Config B Proxy variant of A.
Config C Single end device with self-testing loopback.
Config D Multicast source with measurements stored in sink Flash storage.
Config E Simplification of Config A and C.
Config F Remote replay from flash storage transmitted in multicast mode.

Config B: It provides the capability for the end device to relay messages not intended for itself. This is achieved by verifying the intended receiver ID in each message.
Config C: This is more complicated than it seems at face value because the end device can be instructed to immediately process a certain payload as if a radio callback passed it - a loopback interface[3]. This is vital functionality when testing all variables except for the radio and wireless link. The terminal software also provides source messages, sink data storage, and tracing/logging.
Config E: This shows that the LRConf terminal handles multiple wired connections at the same, making this config quite similar to Config **A, B, and C**. This configuration offers the ability to evaluate a single wireless link between a sink device and a source device.
Config F: This is the most exciting and complex configuration. Flash storage is used for replaying messages through a source device. One or multiple sinks receive this replayed data and store it in flash (i.e., Config **D**) or at the LRConf host.

Note, the output from Config **F** provides data for plots and observations of Chapter 6. This configuration tests the complete RLNC coding scheme from Chapter 4 in a repeatable and, therefore, predictable manner. It is the testbed's primary and most important purpose to ensure that each component in the network performs as it should. Stepping through each Config, more complexity is added to the network progressively. Finally, Config **F** ends up as the setup most similar to a real LoRa network running a RLNC decoding session.

5.1.2 Flash replay RLNC session

As was briefly stated in Section 5.1.1 the source device in Config **F** can send a stream of predefined messages from its flash storage. This data is programmed separately from the application firmware with a compiled blob, so it can be rewritten more conveniently[4]. The layout of this flash memory blob, the validation/loading functions, and the execution of commands stored in the blob are available for review online[45, 46].

Figure 5.4 shows the flow of messages of a LoRa RLNC session as well as the wired commands sent over USB. Note that the transmitter is not directly connected to the host PC on purpose. This shows the independence of said transmitter node and its capabilities to stream multicast RLNC init and start commands, as well as RLNC fragments, to other nodes without requiring a PC. This has one small downside and one major benefit. The downside is that almost all coding parameters like generation size G, fragment count N_f, fragment size F, and redundancy factor δ are kept constant beforehand. Also, the pRNG seed for the encoding vectors is constant (see Section 4.2.4). The benefit of this choice is that the coding experiment is highly repeatable. To compensate for the static nature of this setup, many different configurations are run to generate rich experimental results. The host PC needs to synchronize each configuration to ensure it can track the session properly. It is, therefore, explained how the chain of messages looks to run decoding experiments.

[3] In the loopback construct it is not possible to define a SNR and RSSI, so default and impossible values are inserted.

[4] Since the data blob is a 128 kB binary file it is convenient to store this with the resulting measurements for administration purposes.

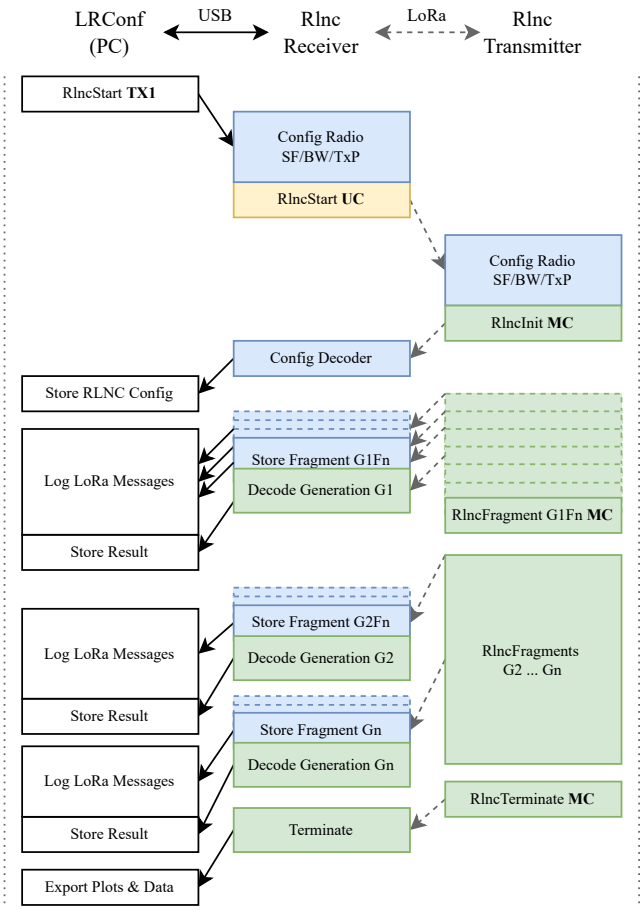


Figure 5.4: **Network message exchange for a RLNC setup. Transmitted Multicast (MC) commands are green and Unicast (UC) in yellow. Blue tiles represent command reception and an executed action. White tiles represent actions on the host PC.**

The first phase is performed by sending the `RlncStart` command, which contains dynamic LoRa parameters like SF, BW, transmission power, and more. Once this start command is received by the RLNC transmitter, it will autonomously start the session. First off, it responds with the `RlncInit` command with its flash configuration on the condition the flash storage content is in the correct state. The receiver end device then resets its decoding state and forwards the received RLNC configuration to the host PC for administration.

The second phase starts after a specific timeout on the transmitter. It will automatically switch to loading and transmitting coded `RlncFragment` messages which the receiver adds to its decoding matrix. The receiver performs row-reduction each time it receives a new fragment, even if it has not received enough fragments to be able to decode completely. This is because the matrix rank of the decoder matrix shows how the far decoding session has progressed. Therefore, the received fragment messages and the updated decoding state are

subsequently sent to the host PC to be logged. Once enough fragments have arrived, the receiver's decoding state will change from failure to success. Even after the generation decoding is complete new fragments are still added to the decoding matrix by overwriting the last row for testing purposes. The second phase is restarted for each new generation. The receiver unpacks the generation index value prefixed to each fragment and, therefore, can determine whether the decoder needs to be flushed for a new generation to start.
The final phase is the termination of the decoding session to command the receiver to flush all its state. This is done by sending the `RlncTerminate` command. Once the host PC receives this forwarded command, it closes the session. The process is iterated from initiation until termination in different configurations. For example, the spreading factor could be changed to find a time scheduling bottleneck, or the PER could be swept to see how robust the decoding is.

Finally, the data is post-processed to estimate the channel conditions, the decoder successes/failures, and how much redundancy was required to compensate for packet loss. This post-processing results in plots presented in Chapter 6. The following section explains how the decoding state updates are post-processed to get the decoding success rate for multiple redundancy thresholds and specific channel conditions, as well as other metrics like PER for specific configurations previously introduced in Section 5.1.1.

5.2 Decoder tests

Currently, two methods of collecting measurements were introduced: *remote* and *direct sampling*. These methods are briefly revisited. In the first *remote* method, the sink end device is not connected to a host PC, so it needs a way to buffer the measurements in non-volatile storage. Therefore, it can store roughly 16000 measurement tuples (SNR, RSSI, and sequence number), after which any new measurement must be discarded. Section 5.2.1 elaborates how the sequence numbers are retrieved from the end device and processed to get the PER metric. The second direct method immediately forwards measurements to a host PC. This is more suitable for high data-rate measurements like decoding updates and debugging the decoding process. Also, the resulting data does not have to be separately fetched from the end device flash, and the flash memory does not constrain the maximum measurement count. Therefore, this situation is more favorable for running the RLNC coding scheme with the testbed. However, one downside is connecting the source and sink end device directly to the host. The distance between the transmitter and receiver antennas is very short, and the channel conditions of the environment are almost perfect. To resolve this Chapter 6.2.1 explains how packets are artificially dropped by software using a uniform probability model or temporal burst model[5]. Now it is established how temporal packet erasure is estimated using the PER metric.

[5]The parameters for the uniform/burst loss filter are also included in the `RlncStart` command

5.2.1 Packet error rate

The received array of measurements contains a sequence number for each received message. A missing (erased) message can only be determined by analyzing the gaps in the series of these sequence numbers. Now, to determine the rate at which packets were lost for a certain moment in time, it is desirable not to include all packets to determine the temporal change in PER. This is solved by masking the sequence numbers using Equation (5.1). The temporal PER is estimated using a low-pass averaging filter. This is implemented using a sliding window of a certain width W. This width determines how quickly the PER can change; therefore, different values of W provide a different insight into how bursty the packet loss was in time. The window starts at packet j and ends at packet k, so the following holds: $W = k - j$.

$$p_i = \begin{cases} 1, & \text{if packet } i \text{ erased,} \\ 0, & \text{if packet } i \text{ received} \end{cases} \tag{5.1}$$

$$\text{PER} = \frac{1}{k-j} \sum_{i=j}^{k} p_i, \qquad \text{for } k > j. \tag{5.2}$$

The PER presented in Equation (5.2) is a reliable estimate of channel performance for a certain time duration W from packet j to k. To determine this metric correctly, received packets from unrelated external sources should not be included in the estimation. Otherwise, the estimated PER could exceed 1. Therefore, functionality is added to force the end device to accept only unicast messages of a specific type targeted to itself. Other messages are discarded and will not be added to the data set and PER estimation.
The method described above provides the ability to evaluate what packets were dropped given measurement data collected using *remote sink setups* like Config **A, B, D, or F** (Section 5.1.1). This leads to PER analysis of experiments done with the testbed of this thesis, shown in Chapter 6.
To test the decoder for different PER levels, packets can be dropped artificially for more pragmatic experimentation of the testbed[6].

5.2.2 Artificial packet loss

The coding scheme is tested using artificial loss. It is not practical to experiment with varying radio channel conditions to look for specific packet loss. As will become apparent with results presented in Section 6.1, burst or temporary loss is a common cause for packet erasures. Reproducing burst losses in realistic environments is challenging, inefficient, and hardly repeatable. Therefore, it becomes intractable to accurately and sufficiently reproduce all levels of burst loss. As such, it is justified to artificially introduce uniform or burst loss into the system, so the firmware and decoder can still be thoroughly tested. Also, real LoRa networks are spread over long-range outdoor line-of-sight locations, and gateways are located at high elevations. An indoor lab situation can not reproduce such environments well.

[6]The result of the remote measurements is useful as input for this artificial packet loss functionality.

Inference 5.1 *Measurements from devices located in short range of each other are reproducible.*

Inference 5.1 significantly impacts the testbed's design because devices can be placed close together, and artificially introduced packet loss makes experiments *reproducible*. As a nice side-effect, the coding scheme can be fully evaluated without being hampered by unwanted faults in the control plane as (experimental) control packets will never be dropped. Using the concept of artificial loss, the following section elaborates how the RLNC performance is evaluated by defining the *generation success threshold*.

5.2.3 Generation success threshold

As explained in Section 4.2.3 the RLNC session is split into progressive generations. It has been shown that RLNC can be extended to *sliding-window RLNC* to adjust to dynamic changes[37, 42]. However, it cannot be realistically assumed that end devices can send back information about decoding progress and estimated network performance[7]. More analysis is required to prove that the network as a whole can guarantee not losing feedback packets as well as RLNC fragment packets. Secondly, it is not a given that the MAC layer can provide enough opportunities in LoRaWAN Class B and C to let the end devices respond. The opportunities for LoRaWAN Class B devices are limited by the ping slot configuration (refer to Section 2.2.2). Although class C devices are offered more liberty through asynchronous transmission scheduling, the duty cycle limit of the PHY layer (0.1% or 1%) limits class C as well as B equally (refer to Section 2.1). For these reasons, it is assumed that dynamic adjustments are disabled. Therefore, each generation must be equal in size and have a sufficient amount of redundancy to cope with packet erasure.

Inference 5.2 *Without feedback from receiver end devices during FUOTA, the on-the-fly adjustment of decoding parameters like generation size and redundancy is unavailable.*

From Inference 5.2 is deduced that sliding-window RLNC is not an option as that method dynamically increases generation size and redundancy in case of burst loss or temporary outage. It would require a control plane with feedback commands which are not available for LoRaWAN.

After a generation of fragments (including redundancy), the receiver should have received enough fragments to decode the generation successfully. This point of success marks the *generation success threshold*. Similarly, if not enough fragments were received, the redundancy was not optimally matched to compensate. This results in generation failure, and the success threshold is not defined. The penalty of generation failure is high if the FUOTA scheme does not have an end device feedback mechanism to restart or extend such a failing generation. Therefore, the redundancy must compensate for peak (expected) PER due to burst loss if failure is intolerable.

Inference 5.3 *Statically configured RLNC generations require equally sized generations with enough redundancy to restore lost fragments.*

[7] Sending feedback with decoding updates and network performance is only realistic for testing environments.

The experimental target is to collect threshold values for enough generations of the same configuration (size and redundancy). However, iterating different redundancies and generation sizes against different non-uniform channel conditions like burst loss rapidly increases the number of iterations of an experiment. Therefore, the sweep parameter range should be selected with caution. However, since all generations in one experiment iteration are equally sized (Inference 5.3), and generation success is defined by reaching the generation success threshold for several transmitted fragments, an optimization exists. One generation can be used to sample many redundancy levels (data points), and the data points of each generation can be combined into a histogram. After normalizing, the bins in the histograms will more accurately approach a distribution given that a high number of generations is sampled in the experiment.

Inference 5.4 *Taking the redundancy level capable of counteracting worst-case PER is sufficient to get a complete success rate distribution.*

Inference 5.4 leads to a method for estimating the optimality of a certain redundancy factor δ or R. The reasoning from Inference 5.4 is useful for evaluation of how well a FUOTA session performs. The target is to reduce the unnecessary amount of redundancy while still keeping enough to have robustness against worst-case packet loss.

5.3 Projecting onto LoRaWAN

Although FUOTA for LoRaWAN is still in its infancy, many proposals and specifications provide a foundation for companies to develop applications to be deployed with a firmware update strategy in mind. One such example comes from STMicroelectronics who have developed the STM32WL series of system-on-chip development boards for LoRa accompanied by complete example projects to develop a FUOTA-based application[48, 4]. This board is equipped with two separate CPUs, one meant for an updatable application firmware and another for a fixed FUOTA management LoRaWAN firmware[48]. This setup shows that FUOTA requires strategic planning starting from the hardware design to the software stack on the end device. The next section shows the software blocks currently present in example FUOTA application projects provided by STMicroelectronics and Semtech.

5.3.1 FUOTA firmware layers

The company Semtech also brings forward specifications for FUOTA layers presented in Figure 5.5.

The NS (network server) on the right part of the image was briefly introduced in Section 2.2.3. Common NS server stacks are discussed in the next section. The App Server and Device Management block has been completely built from the ground up for this thesis for complete experimental control. High-level features such as security with public-private key encryption, over-the-air activation (OTAA), or activation by personalization (ABP) have been replaced by simple and insecure device hardware identifier exchange. This thesis's experimental setting for firmware updates does not need device activation. The LoRaWAN

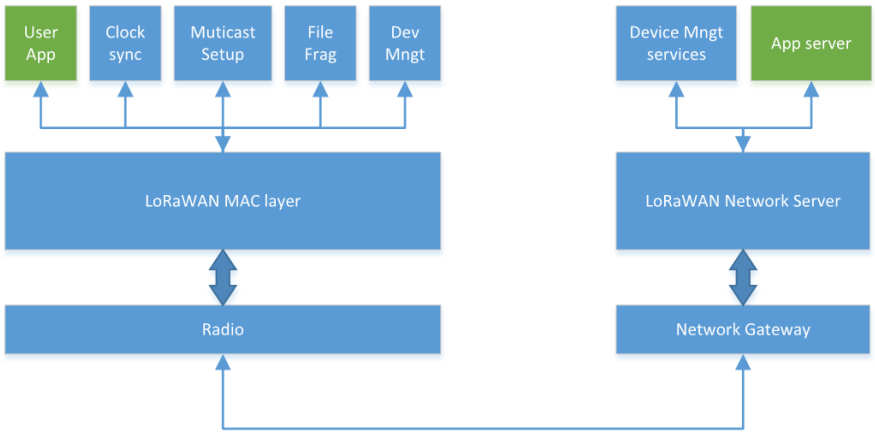


Figure 5.5: **Semtech FUOTA layers for LoRaWAN. The NS device management and application backend are on the right. The end device system layers are on the left. Image source: Semtech Developer Portal[26].**

MAC layer (on the left) has been completely replaced by the control plane introduced in Section 5.1. Since Class A, B, or C were not implemented, the *Clock Sync* block is not required. No accurate timing is required to improve measurements to estimate PER or decoder failure rates.
The *Multicast Setup* block has been replaced by a simple control plane using device hardware identifiers to filter LoRa packets that are unicast and mark multicast packets with a separate flag. In other words, the *Multicast Setup* block was left out of the testbed.
The *Dev Mngt* (device management) block provides bootloader features for secure firmware updates. The bootloader is responsible for loading firmware at boot, hence the name. The bootloader is not relevant for evaluation of RLNC for FUOTA.
The *File Frag* has been replaced by the RLNC decoder. Validation functionality is added to evaluate the RLNC decoder output aimed at making sure no decoder matrix corruption happened during experiments (refer to Section 5.2) and to track decoder success ratio. To achieve this different functionality was required than what was provided by the fragmentation decoder of FUOTA. A more in-depth comparison between the File Frag (also known as Frag Decoder) based on LDPC (Low-density Parity Checks) and RLNC is provided in Section 5.3.3.

5.3.2 Comparison TheThingsNetwork and ChirpStack

Two well-known LoRaWAN server stacks exist, at the moment of writing.

- ChirpStack (self-hosted, open-source model)
- TheThingsStack (publicly/self-hosted, commercial and open-source models)

The ChirpStack is a network server compatible with LoRaWAN 1.0 and 1.1[20]. It is an open-source server that can only be self-hosted to form private networks. Together with ChirpStack FUOTA Server, it is an exemplary implementation to set up a private network for LoRaWAN[49].
TheThingsStack is a stack that is publicly accessible through TheThingsNetwork. In specific regions around the world, public LoRaWAN gateways provide coverage to connect private LoRaWAN to the internet. Although this is ideal for testing LoRa device for connectivity, the stack is not compatible with FUOTA fragmentation and multicast capabilities and it is not clear if this is going to be implemented[12, 14, 5].

The Fragmentation Decoder (Frag Decoder) is studied by taking a look at the embedded example application provided by Semtech's LoRaWAN firmware for STM32 devices[7]. This FUOTA example implements the LDPC code.

5.3.3 Low-Density Parity Checks

The FUOTA specification has been publicly provided by Semtech, which proposes a fragmentation block decoder based on LDPC[22]. This is a fixed-rate block decoder (refer to Section 4.1.2).
It is fact that LDPC has a limited decoding efficiency. Each generation requires between roughly 2% up for large generations (large blocks) to 10% for small generations (small blocks) of extra fragments in order to decode succesfully[22]. Based on the decoding probability for RLNC provided in Section 4.3.2, it is clear that RLNC requires almost 0% extra fragments if GF(2^8) is used, which is an overall improvement. This is not the only property for which RLNC is more favorable.
The LDPC code rate must be determined beforehand. No dynamic adjustments are possible, so options similar to sliding-window RLNC are not available. More importantly, the proposed code starts with *systematic transmission* of all block packets. Such a systematic coding phase is also available for RLNC simply because it is a transmission of uncoded fragments. Only after this phase will LDPC transmit encoded fragments. Surprisingly, when it is known that the network has packet erasure, the code is not orchestrated to encode packets immediately. Section 6.1 presents simulation data proving that randomly scrambled systematic coding performs very poorly against uniform packet loss and burst loss. Systematic coding is not an optimal setting for a FUOTA under poor network conditions.

RLNC is useful in multi-hop networks using the re-encoding feature. Transmitted fragments could be recoded or remixed by intermediate nodes (like intermediate nodes or gateways in LoRa) to generate innovative packets. This feature is not very useful for the current network topology of LoRaWAN. Still, the possibility that this is an option for custom private LoRa networks is a benefit of RLNC over LDPC.

Chapter 6

Evaluation

This chapter presents evaluations of the testbed using measurements in real situations. Section 6.1 shows indoor and outdoor experiments to create an understanding of the testbed system. Secondly, it shows the robustness of the measurement storage system and complete testbed. Finally, an analysis of experiments classifies multiple types of channel conditions. Following, Section 6.2 uses uniform packet loss to present an analysis of the robustness of a block-based RLNC decoder and provides a method of analytically establishing the worst-case performance of a network of devices. The result is the generation success threshold, which extends to the worst-case network success threshold. Continuing, Section 6.3 provides time-discrete simulation results for testing a single decoder against different kinds of burst loss. Finally, scrambling methods similar to systematic coding are compared to RLNC. These methods provide insights into why RLNC is such a strong decoding method in case of burst loss, which concludes this chapter.

6.1 Experimental validation

Multiple outdoor experiments led to the following results, of which two specific locations: indoor (faculty) and outdoor (park). Although the experimental setup is detailed, the methodology used did not include reproducibility. Instead, it serves different purposes.

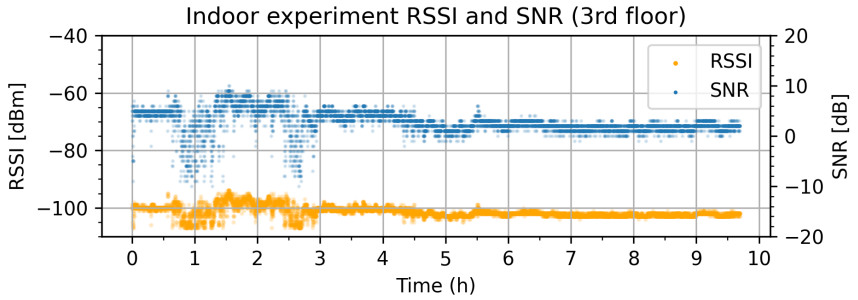
6.1.1 Indoor test results

The indoor experiments serve four goals set beforehand:

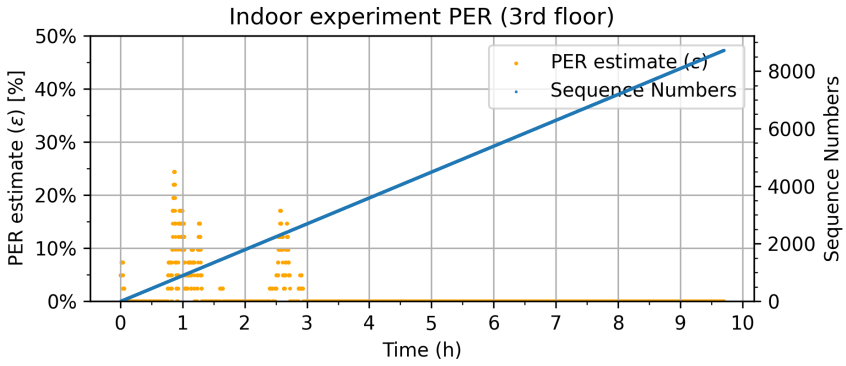
1. Prove the aspects of Configs **C** and **D** (one host and flash storage) work.
2. Show that the testbed firmware is able to run for long duration without crashing.
3. Show only monotonic increase in received sequence numbers (no packet symbol errors).
4. Prove flash storage is able to retain data.

5. Prove no unknown packets are included in the data set.

Experiments were performed, which ran for almost 10 hours, proving goal **1**. These confirm that the firmware is stable and the main functionality is bug-free (goal **2**).
Figure 6.1 and 6.2 show two measurement sets for two different receivers on the 3rd floor and 4th floor of our faculty. Although more data sets were collected on other floors, these two sets, specifically, were selected as they provide sufficient insights.
Both receivers sample the same messages transmitted by one transmitter on the 2nd floor of the building. The sample data points contain three measurements: SNR, RSSI and sequence number. The latter is post-processed to estimate the PER from Equation (5.2) using a window $W = 41$ messages[1].



(a) **3rd floor RSSI and SNR samples.**

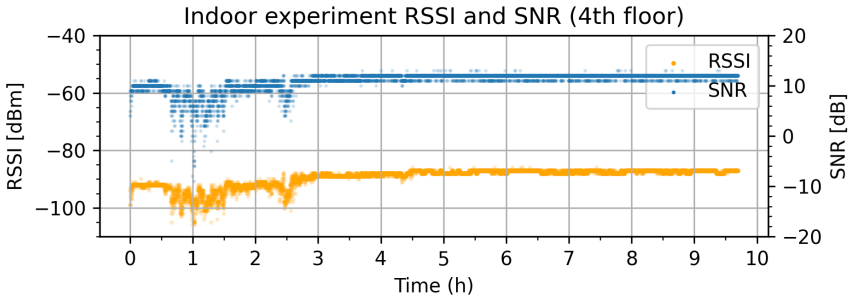


(b) **3rd floor PER estimation ($W = 41$).**

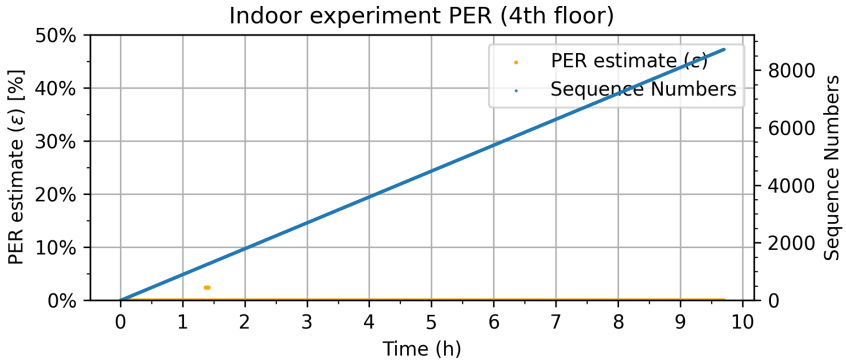
Figure 6.1: 10 hour **long indoor session from 7PM to 5AM on the 3rd and 4th floor at the university faculty. The transmitter was sending 4 dummy messages per second from the 2nd floor. Left side: (Yellow: RSSI), (Blue: SNR). Right side: (Yellow: PER), (Blue: Sequence number). SF7, $BW = 500$ kHz, $P_{\mathbf{TX}} = 14$ dBm, 872 MHz.**

Figure 6.1b and 6.2b show that the received sequence numbers never decreased. Consequently, both plots indicate that *goal 2* was met, which has also

[1]Window size W needs to be odd for the underlying mathematical toolbox to work, which has no impact on the result.



(a) **4th floor RSSI and SNR samples.**



(b) **4th floor PER estimation** ($W = 41$)**.**

Figure 6.2: 10 hour **long indoor session from 7PM to 5AM on the 4th floor at the university faculty. Left side: (Yellow: RSSI), (Blue: SNR). Right side: (Yellow: PER), (Blue: Sequence number). SF7,** $BW = 500\,\text{kHz}$**,** $P_{\mathbf{TX}} = 14\,\text{dBm}$**,** $872\,\text{MHz}$**.**

been verified on the data points directly. Therefore, *goal 3* was met. Finally, *goal 4* was verified by turning off the transmitter. No alien packets were accepted, therefore *goal 4* was met. This is possible because of the packet filter for the control plane introduced in Section 5.1.

Observations: Figure 6.1a and 6.2a show reduced RSSI and SNR in the first 3 hours, which stopped afterwards. Figure 6.2a shows lower PER, SNR, and RSSI even thought this receiver was located one floor higher with respect to the transmitter. but this is not seen in the plot. The PER plot in Figure 6.2b shows that almost no packet loss occurred for this receiver. Consequently, the 4th floor data set is an experimental control group and the 3rd floor represents the actual experiment. The PER plot in Figure 6.1b shows increased PER in the first three hours of the experiment, after which no packet loss was measured. Without context, it is unclear from the four plots what happened at the start as some packet loss occurred. This loss went away once the faculty closed late in the night. One apparent cause became clear. During the evening, students moved around the transmitter in the faculty and, more importantly, in the same room. After closing time, these people left the building.

Inference 6.1 *Channel conditions can deteriorate with the presence and move-*

ment of the human body. People moving around the transmitter can cause disruptions resulting in increased PER.

Inference 6.2 *Indoor measurements sampled during night-time result in little fluctuation.*

6.1.2 Outdoor test results

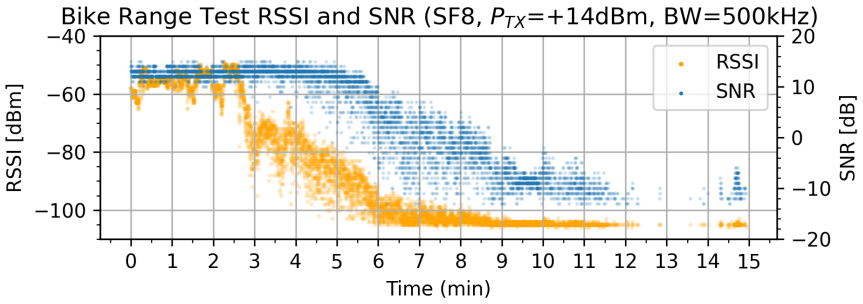
The following goals describe the purpose of the outdoor experiments:

1. Prove synchronous sampling using Config **A** and **B** (two independent PC hosts).
2. Prove the radio and antenna configuration are able to set up a $> 1\,\text{km}$ wireless link.
3. Characterize causes of packet loss.
4. Characterize PER associated with burst loss.

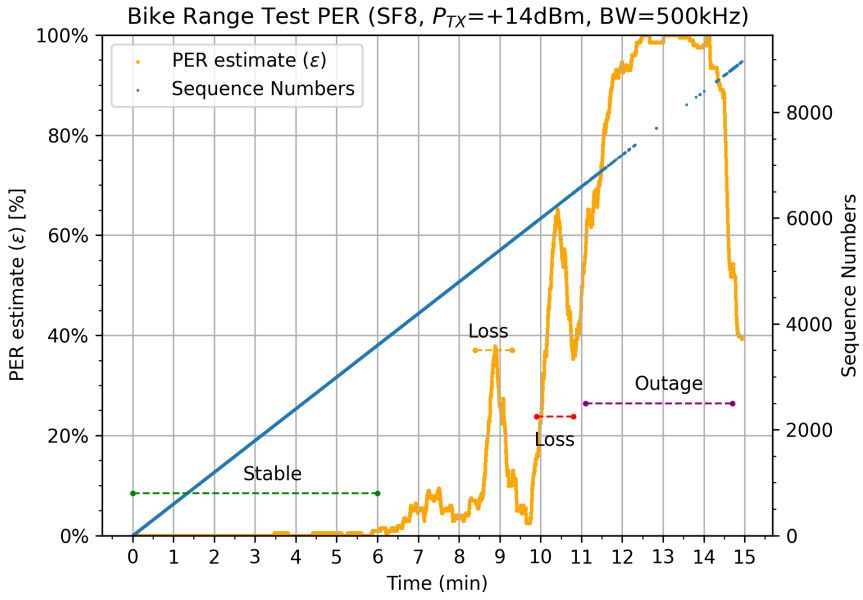
Figure 6.3b shows similar measurements stored by one receiver mounted on the rear of a bicycle. An independent host controls the transmitter(*goal 1*). The whole experiment iterated multiple spreading factors. Specifically, SF8 was selected as it resulted in interesting details highlighted in Figure 6.3b. o The experiment was a dynamic range test from 0 m to 1.2 m (*goal 2*). While riding the bicycle with a receiver attached to the rear end, the distance was steadily increased while measurements were stored. The cycled route was quite straight, visible on the travel route in Figure 6.4. Note that after 800 m a concrete bridge was passed, which is visible on the map. The *Start* position is synchronous with the start of the plot. The *End* marking, however, was reached around the 14 min timestamp.
Observations: Figure 6.3a shows that the RSSI started to decrease roughly at time 3 min and the SNR 4.5 min. At this time, more trees and other cyclists started to occlude the direct line of sight to the gateway. The figure also shows that, between time 11 min and 15 min, many packets did not arrive. The PER in Figure 6.3b confirms this in more detail. Secondly, it is clear from this plot that after time 6 min the channel conditions deteriorated slightly. At time 8.5 min high packet loss was measured, which shortly disappeared after, followed by higher packet loss. Finally, complete outage as the PER approached 100 %. This outage comes from a partial line of sight occlusion due to the concrete bridge suddenly being between the transmitter and receiver (*goal 4*).

Inference 6.3 *Three typical types of channel conditions are classified: stable, lossy and bursty (outage).*

Inference 6.4 *During outage the PER approaches* 100 % *(goal 4).*



(a) **Experiment RSSI and SNR data set.**



(b) **Experiment PER estimation** ($W = 201$).

Figure 6.3: **Experiment on bike with a statically mounted end device (source) at** $2\,\mathrm{m}$ **elevation using a tripod. Three types of packet loss are marked with the respective labels *Stable*, *Loss* and *Outage*.**

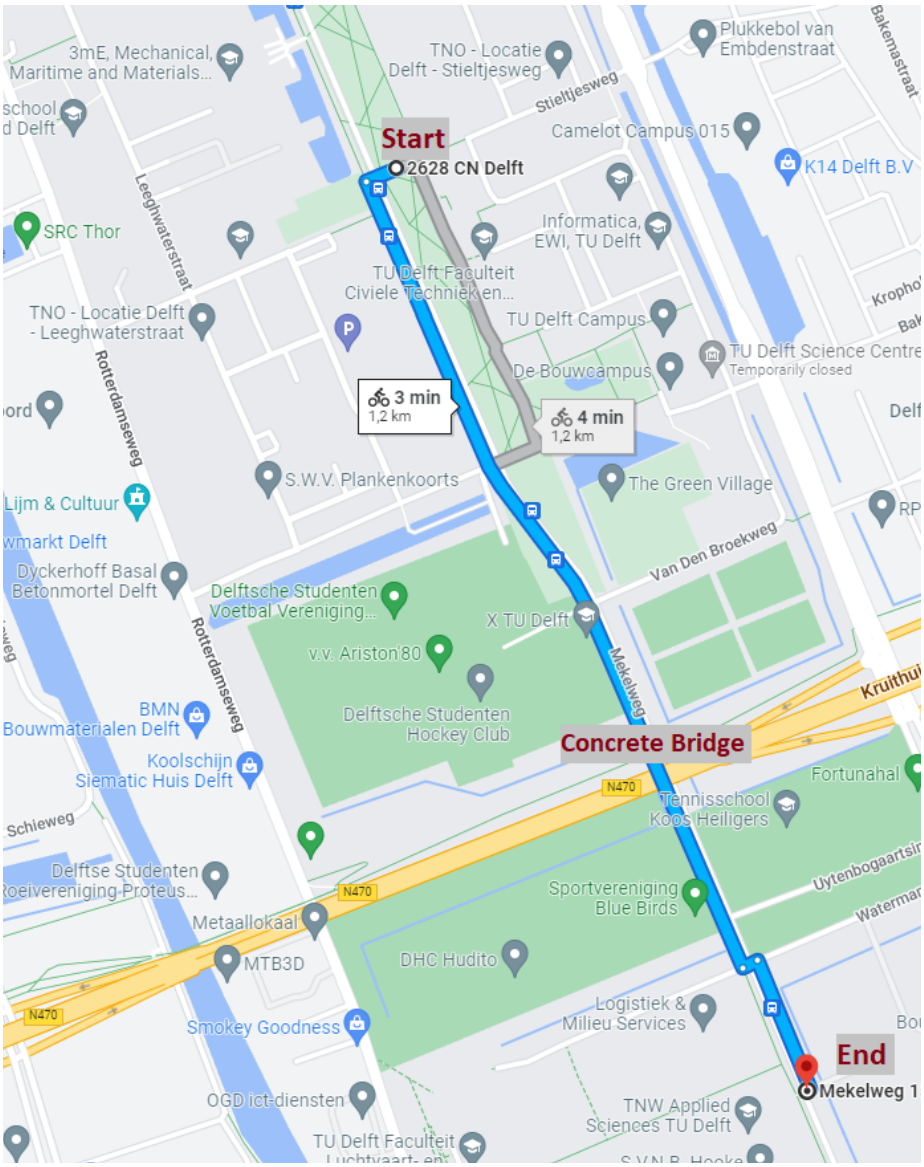


Figure 6.4: **Cycling route near the Technical University of Delft covering a distance of** 0 km **to** 1.2 km **from a source transmitter end device. The starting point and end point are marked. Between** 11 min **and** 14.5 min **the concrete bridge was passed, which caused outage due to lack of line-of-sight.**

6.2 Decoder model validation

It has been established that normal packet loss and burst loss can occur when conditions like range and line-of-sight change. With this in mind, the RLNC decoder has been tested using artificial packet loss. This packet loss is configured with the ϵ-parameter. Although it can be adjusted dynamically during a RLNC decoding session, the ϵ-parameter will initially be kept constant for clear validation. In Section 6.2.1, the packet loss distribution is presented[2]. This distribution verifies that the receiver performs the artificial packet loss filter properly. This filter is applied in Section 6.2.1, proving that the decoding probability of one decoder approximates the provided model of Section 4.3.2 closely. Finally, the worst-case decoding probability is studied for a network of many devices (3000) and constant ϵ to give insight into what faulty conditions RLNC can repair packet loss.

6.2.1 Packet error rate

Figure 6.5) shows PER boxplots. This information aims to verify that the artificial packet loss filter works: it is random and behaves similarly to the expected probability distribution. Each plot represents 30 generations of size $G = 20$. The redundancy is $\delta = 3$ ($R = 80$), or in other words 400 % of G. The PER is estimated over each generation with maximum redundancy, which is a constant of 80 fragments.

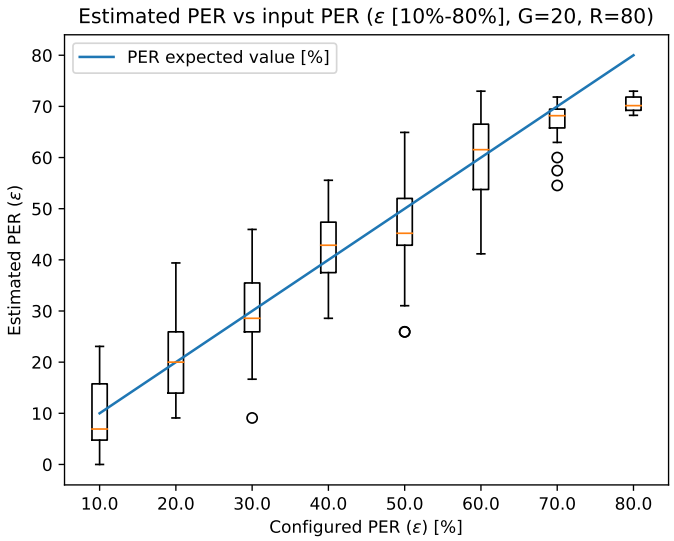


Figure 6.5: **The configured PER resulting in a Bernouilli PER distribution. It is visible from** 10 % **to** 80 % **that the mean PER approaches the expected value.**

Observations: The PER boxplots are evenly distributed around the PER

[2]The following transmission settings can be assumed SF7, $BW = 500\,\text{kHz}$, $P_{\text{TX}} = 14\,\text{dBm}$ unless specified otherwise.

filter probability ϵ. Also, the mean PER is generally near ϵ. The mean and distribution for 80 % is on the low side. However, it is shown in Section 6.2.2 that this PER still results in much poorer decoding success rate overall.

Inference 6.5 *2400 fragments are sufficient to approximate an Bernouilli ϵ-distribution.*

6.2.2 Single decoder

The effect of artifical packet loss on a single decoder is visualized using the expression $P_{\text{success}} = 1 - P_{\text{fail}}$ from Equation (4.9) in Section 4.3.2. Figure 6.6 shows the modeled probability for different amounts of redundancy (0 % up to 400 %). The PER is low, which means that only 65 % redundancy is needed to decode a generation. Therefore, only up to 150 % redundancy is plotted.
Note, the underlying histogram has been plotted as well for completeness.

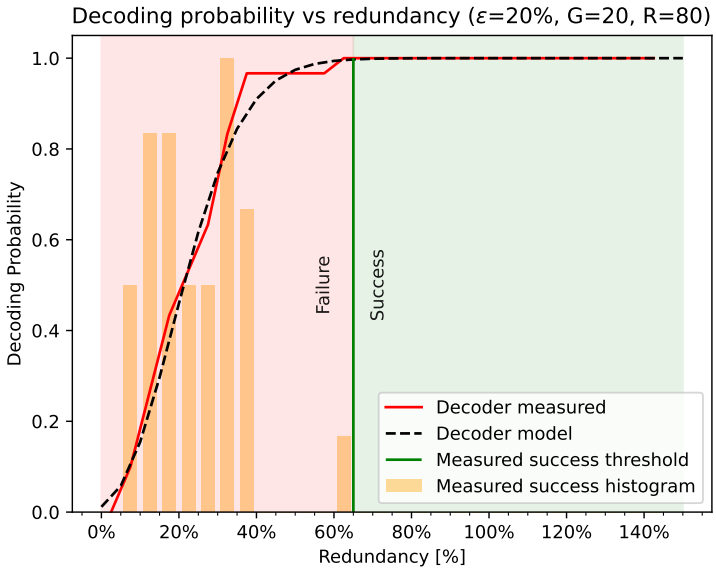


Figure 6.6: **Histogram (yellow). Modeled (black, dotted) and measured (red) decoder probability for $G = 20$, $R = 80$, and artificial packet loss $\epsilon = 20\,\%$. The resulting generation success threshold is at 65 % redundancy, beyond which decoding success is likely.**

Observations: This experiment closely follows the theoretical model. The success threshold, determined by finding the model 0.995 decoding probability intersection point, lies at 65 %. This fact is similarly observed in the experimental data with artificial packet loss. If, for example, 80 % redundancy is added to the generation size, or 36 fragments in total, the generation will be decoded with high probability.

Inference 6.6 *The redundancy threshold depends on the $P_{success}$ decoding probability model. Consequently, exceeding this threshold compensates for packet loss accurately.*

Multiple PER values have been configured for the real decoder. The data sets and model have been plotted in Figure6.7. **Observations:** The plot shows that $\epsilon = 70\%$ and $\epsilon = 80\%$ could not complete successfully with the maximum redundancy of 300 % added to the generation. These conclusions are supported by the model, which succeeds and fails for the same PER values. Also, the redundancy threshold does not increase linearly with PER. This is supported by plots in Figure 6.8. That plot is the Probability Density Function (PDF) of the model. It shows that for increasing PER, the required number of extra fragments increases quickly.

Inference 6.7 *The decoding probability based on Bernouilli packet loss converges to a normal distribution.*

Inference 6.8 *For higher PER the decoding PDF increases in standard deviation; compensating this with redundancy is very expensive.*

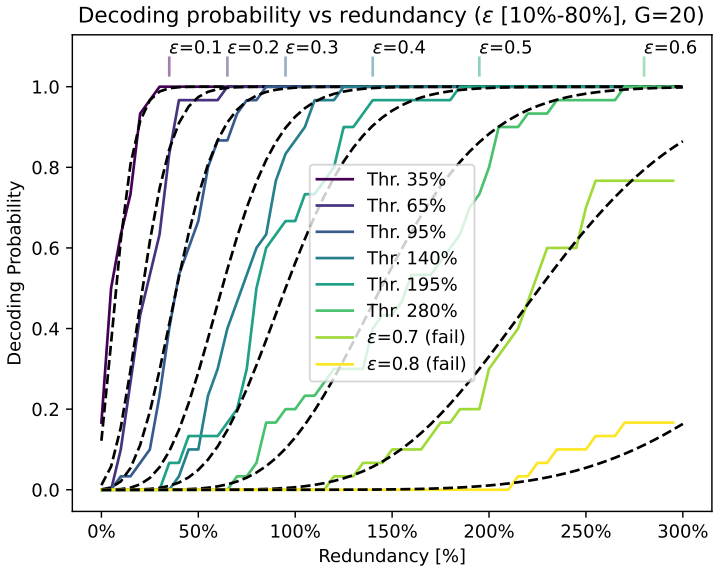


Figure 6.7: **Measured average generation decoding probability for different channel ϵ conditions. Each data point is represented by 30 independently decoded generations. The redundancy threshold for succesful iterations is shown in the legend as *Thr*. Decoding failed for ϵ=70 % and ϵ=80 %.**

6.2.3 Network decoding

By itself, the decoding probability is not sufficient for determining if a firmware update session completes successfully. All generations must be received and decoded fully for the firmware to be of any value. As such, one failed generation results in complete failure, represented by Equation (6.1). Note that this

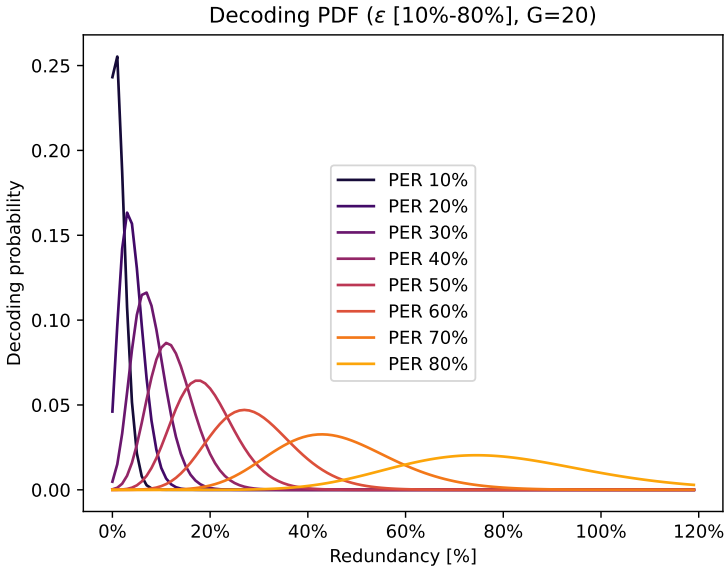


Figure 6.8: **Model PDF distributions for decoding probability for different PER. The deviation of each distribution increases for higher PER.**

equation is based on P_{fail} defined with Equation (4.9). It expresses the requirement that not only one generation must be decoded successfully, but all G of them. Secondly, the probability of several devices in a network to succeed is incorporated using parameter D. A different approach would be required to get the statistical decoding distribution for a small number of devices (single device distribution). By taking a high device count, f.e. $D = 3000$, the individual device distribution is completely represented with $((P_{\text{fail}})^G)^D$.

$$P_{\text{GD}}(G, R, \epsilon, k, D) = (1 - P_{\text{fail}}(G, R, \epsilon, k))^{GD} \tag{6.1}$$

Figure 6.9 presents the decoding probability of an arbitrary network of 3000 devices presented with the channel condition of ϵ=40 %. Similarly, Figure 6.10 introduces ϵ=50 % to the network.

Observations: The decoding probability of all generations is lower than one generation. In other words, the amount of redundancy is greater, and the all-generation success threshold is shifted to the right. Also, the all-device success threshold is shifted to the right, where the green region is drawn. This region indicates the redundancy configurations which will complete decoding. This region starts at $R \geq 72$ fragments or 260 % redundancy for ϵ=40 %. However, for ϵ=50 % the decoding does not succeed. The 10 % additional packet loss had a ripple effect, and the network required much more redundancy.

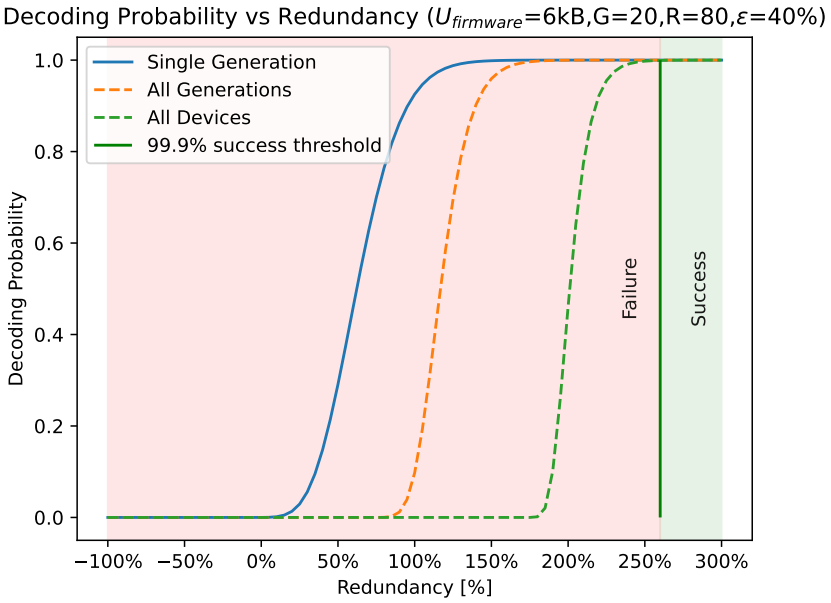


Figure 6.9: **RLNC probability distribution for D=3000 devices decoding a 6 kB firmware image with ϵ=40 % on all channels. The decoding session is successful with more than 99.9% chance, given that more than 260% redundancy (δ=2.6, R=52+G=72) is added.**

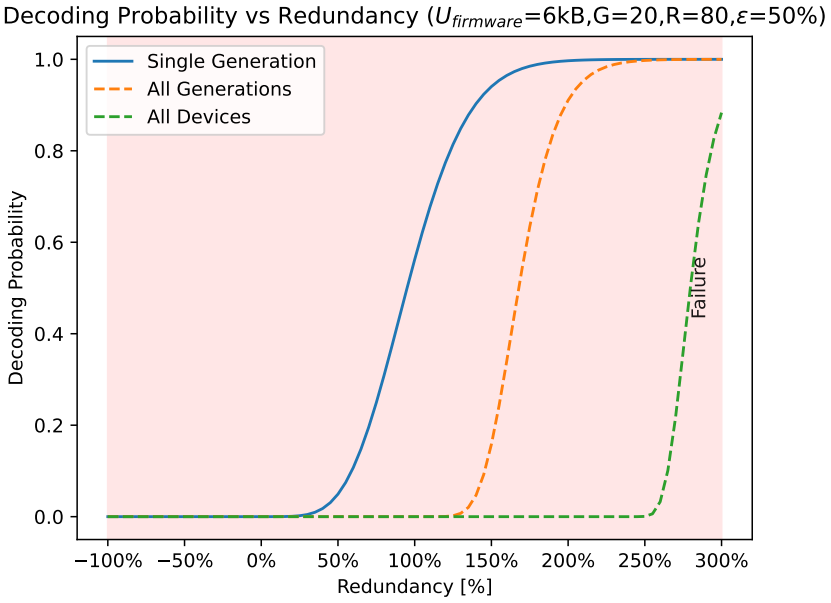


Figure 6.10: ***Failure* RLNC probability distribution for D=3000 devices decoding a 6 kB firmware image with ϵ=50 % on all channels.**

6.3 Burst loss

The effect of multiple packets erased in a sequence has been identified as burst loss. Burst loss shows itself in a sudden increase in PER for a particular duration. Statistically, the burst loss random variable is correlated in time instead of being Independent and Identically Distributed (IID). One example of such an occurrence is a temporary change in channel conditions, which occurred in the data sets presented in Section 6.1. To give a better overview of the probability of failure of RLNC, modeling the burst loss is essential. For example, if burst loss occurs during a vital part of transmitting a generation, the generation is probably lost altogether. Many generations after still succeed when the burst loss has disappeared. So, the mean PER might still be low, and it does not represent the channel conditions or the burst loss well.
The Gilbert-Elliott burst model introduces a temporal correlation for sudden packet erasure, which increases PER and causes multiple packets to be dropped[50, 51]. The model is based on a 2-state Markov chain presented in Figure 6.11, so two exit probabilities p and r are configured to model the chance of leaving the state of burst loss or good packet reception. The model is quite simplistic and should only be used for testing robustness against specific burst loss. For example, the model cannot simultaneously mimic minor changes of PER in that configuration - at least not without dynamically changing the Markov model.

$$\pi_G = \frac{r}{p+r} \tag{6.2}$$

$$\pi_B = \frac{p}{p+r} \tag{6.3}$$

$$\pi_E = (1-k)\pi_G + (1-h)\pi_B = \pi_B \tag{6.4}$$

The burst will be modeled with $\epsilon = 1$ packet erasure for ratio π_B (Equation (6.3)) and $\epsilon = 0$ for ratio π_G (Equation (6.2)). The good/gap state G and bad/burst state B represent toggling between stable and outage channel conditions. Equation (6.4) is the expected PER due to this toggling effect.
The parameter k is the PER in good state and the parameter h is the PER of burst state. Often, k is set to 1 and h is set to 0 in this equation. Therefore, Equation (6.4) reduces to π_B, the ratio of time spent in the burst state related to the time spent in the good state. Because the burst and gap states either represent packet loss or perfect reception, π_B is equal to the expected value of the packet error rate.

In order to change the burst duration, but not the mean PER, π_E should be kept constant. This is briefly discussed in the context of LoRaWAN MAC layer. Referring to Section 2.2.1, it is known that LoRaWAN Class B and C are characterized by low data rate and high inter-arrival time of transmitted frames. Therefore, in the case of temporary outage, the channel conditions have changed deteriorated for a certain duration.

Inference 6.9 *$k = 1$ and $h = 0$ are justifyable for deteriorated channel conditions in LoRaWAN Class B and C.*

Therefore, the ideal model configuration for burst loss must provide ability to sweep over varying burst time durations to study variable burst *intensity.*

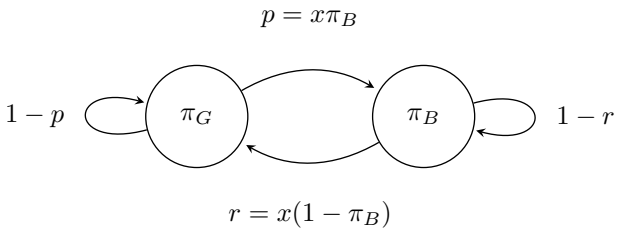


Figure 6.11: **Gilbert-Elliot 2-state Markov chain for burst loss simulation.**

So, in order to keep the mean PER the same while sweeping over different burst behaviour, the exit probabilities p and r are related with a coefficient x as presented in Equation (6.5) and 6.6:

$$p = x\pi_B \tag{6.5}$$

$$r = x(1 - \pi_B) \tag{6.6}$$

$$P_{\text{duration}}[i] = r(1 - r)^i \tag{6.7}$$

Equation (6.7) provides the PMF given i burst time-steps. Figure 6.12 shows the histogram of pseudo-randomized simulation using the Gilbert-Elliot Markov chain.

Observations: The histogram tracks the model PMF very closely. Therefore, the simulation and model are suitable for analyzing a network with burst loss for the same expected PER. Also, higher values of coefficient x result in less spread of the burst duration (time steps). Similarly, a lower value of x will increase the expected burst duration while decreasing the frequency as the distribution becomes flat.

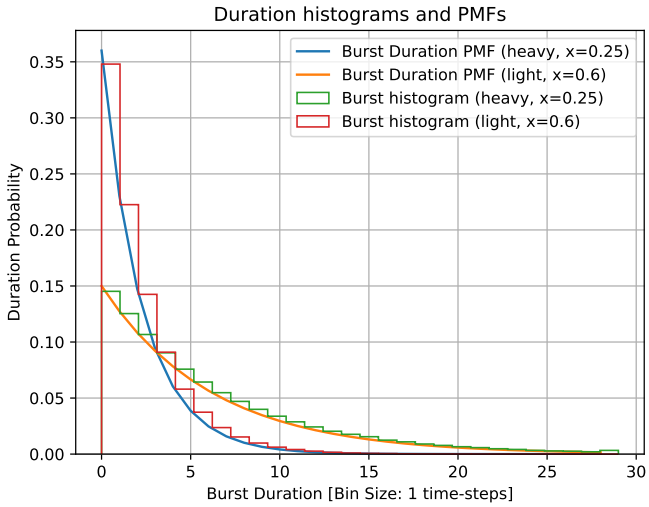


Figure 6.12: **Histogram and PMF for 5 million messages in good ($x = 0.6$) and bad ($x = 0.25$) burst mode.**

Figure 6.13 shows a simulation of 5 million fragments for values of $0 < x \leq 1$ and mean PER $\pi_E = \pi_B = 0.6$. Although the full range was simulated, only $x > 0.25$ is relevant. Below this value, bursts become extremely long and infrequent. This is not apparent from the plots in Figure 6.13. The plots contain temporal PER, the decoding success rate, and *needed redundancy* (min, mean, and max). The needed redundancy is the redundancy required to succeed in decoding a generation. Note, the *redundancy Used (max)* plot is bounded by a redundancy limit of 200 % to keep the simulation tractable. Secondly, note that failed generations will not count for the used redundancy as this made the data harder to interpret.

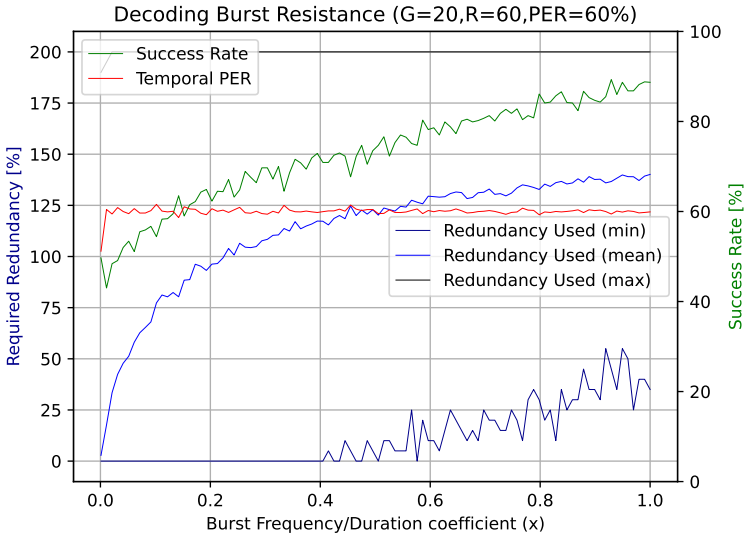


Figure 6.13: **Burst success rate simulation with the required redundancy. 5 million fragments are transmitted to one decoder under different burst loss conditions. Heavy burst (left) requires less redundancy overall, but also shows much lower success rate.**

Observations: The PER is roughly constant at 60 % as was enforced by π_E from Equations (6.5), (6.6) and (6.4). The maximum redundancy used is always nearly at the maximum available redundancy of 200 %. However, the success rate is lowered when lowering x. Similarly, the min and mean used redundancy lowered as fewer generations succeeded. In other words, the redundancy did not play an effective role in providing a higher success rate when the bursts are increased in duration.

Inference 6.10 *The Gilbert-Elliot model correctly maintains constant mean PER with Equation* (6.4) *for variable burst frequency/duration ratio x.*

Inference 6.11 *Heavier burst loss (for example, $x < 0.5$) results in long and infrequent outages. This results in 1) lower decoding success rate and 2) ineffective use of redundancy.*

6.4 Comparison RLNC and systematic coding

The following three generation-based dissemination methods are compared using time-discrete simulation:

- Random Linear Network Coding, block-based (Rlnc)
- Randomly Mixed (Rngmixed)
- Semi-Randomly Mixed (Semi-Rngmixed)

Systematic coding effectively disables the mixing feature of the chosen encoding mechanism. Fragments are transmitted with the rows of the identity matrix instead of a randomized encoding vector (see Section 4.2.4). Such a dissemination method is not a robust solution for packet loss or burst loss. The purpose of system coding is to transmit packets and to provide re-encoding capabilities for intermediate nodes in a multi-hop network[37]. Therefore, it is not intended to be robust against varying channel conditions, nor is it suitable for a single-hop star network such as LoRaWAN (Section 2.2.3). To make comparison between systematic RLNC and randomized RLNC (or LDPC for that matter) fair, two methods have been developed: *Rngmixed* and *Semi-Rngmixed*.
Instead of mixing fragment symbols, the first systematic approach *Rngmixed* randomly scrambles the order of a generation's fragments, storing the result in a generation buffer B_G. The redundancy is generated by copying the same generation fragments multiple times. Consequently, the *Rngmixed* method reorders duplicate fragments. It is likely that two fragments with the same content are put next to each other. When burst loss occurs during the transmission of such equivalent fragments, the symbols in those fragments are lost altogether. Therefore, that generation will fail decoding. The hypothesis is that this is a primitive approach. It was chosen to serve as a comparative method concerning burst loss robustness.
Another variant, *Semi-Rngmixed*, has been developed for this comparative study. The *Semi-Rngmixed* approach extends and improves *Rngmixed* by ensuring that any two fragments are separated by at least the half the generation size in fragment count. The following steps are performed by the method:

1. The method allocates an empty generation array (buffer B_G).
2. The original generation of size G is split into two sets G_1 and G_2.
3. The scrambling method will be repeated $2(1 + \delta)$ times (δ being the generation redundancy factor)
4. The method will keep toggling between G_1 and G_2, scrambling each set $(1 + \delta)$ times.
5. Each step randomly permutes either G_1 and G_2 and adds the resulting fragment symbols to the B_G.

The method is evaluated against burst loss applied to buffer B_G. Each generation succeeds decoding if at least each original fragment is received after packet loss is applied. Figure 6.14 shows how resilient the three methods are against burst loss. The parameter $R = 60$ was kept below generation threshold to reduce simulation duration. The resulting insights remain unaltered. Instead,

a high generation count $N_g = 1000$ was chosen, ensuring enough burst loss opportunities.

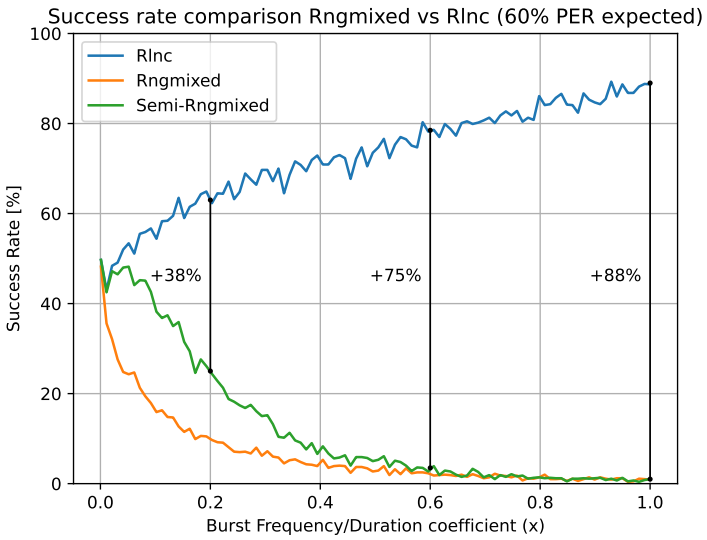


Figure 6.14: **Burst success rate for *RLNC*, *Rngmixed* and *Semi-Rngmixed* using $N_g = 1000$ generations per sample. Each generation is simulated with $\delta = 2$, $R = 60$ fragments. Note, only roughly $x > 0.2$ is relevant.**

Observations: The plot shows that the three methods never succeed with 100 % success rate, which was chosen on purpose. RLNC outmatches Rngmixed and Semi-Rngmixed for all kinds of burst loss. As stated before, only the domain $x > 0.2$ is relevant, because lower ratio models irrealistic burst loss duration (see distribution of Figure 6.12). Secondly, Semi-Rngmixed clearly outperforms Rngmixed for $0.2 < x < 0.4$. Thirdly, it is visible that burst loss has both a negative impact on RLNC success rate as well as a slightly positive impact on scrambled approaches. Concluding, it is visible that for $0.2 < x < 0.4$ RLNC performs between 38 % and 88 % better than systematic coding, even if non-primitive scrambling with minimum separation distance of equal fragments if applied.

Inference 6.12 *Systematic (scrambled) coding performs badly in network conditions with burst loss and should be avoided at all cost for LoRa.*

We know that LDPC has roughly 2 %-10 % decoding overhead dependent on the generation size (large to small size respectively). Since it has been shown that RLNC is able to tolerate burst loss and it has near negligible decoding overhead, this method should be preferred.
More importantly, LDPC for FUOTA transmits all fragments with systematic coding first. This means that, for example, with an overhead of 100 % or coding rate $CR = \frac{1}{2}$, 50 % of the fragments are systematically coded. If the coding rate is reduced, the portion of systematically coded fragments becomes less. Still, as this portion of fragments is large, RLNC will outperform during this stage.

Chapter 7

Conclusions and Future Work

7.1 Future work

Firmware updates are pretty disruptive operations for LoRaWAN networks. Results have shown RLNC to work even with high packet loss. Following work needs to explore and standardize this.

- **Large network simulation:** simulating RLNC and LDPC in simulators like *FUOTASim* or *ns-3* will lead to a complete overview of FUOTA for large LoRaWAN networks with multiple gateways and channel conditions. Realistic effects like multipath, Fresnel (gateway), capture effect, power effect could be implemented in either of these simulators. One interesting aspect to study could be to measure the decoding success distribution.
- **MAC layer testbed:** Implementing the complete MAC layer will lead to realistic network. An end device and gateway which connects the ChirpStack and the ChirpStack FUOTA server must completely implement the MAC layer. This could be studied to see how much RLNC reduces the FUOTA session duration and, therefore, how much energy could be saved for the end device.
- **Closed-loop FUOTA control plane:** It can be assumed that end devices in a network experience different channel conditions. Implementing a feedback mechanism that avoids ACK-implosion and which is designed for scalability could provide a solution to optimize decoding parameters like generation size and redundancy dynamically. This could lead to a more efficient redundancy factor for specific groups of devices in the network. One approach could be to cluster devices in multiple multicast groups and adjust the decoding parameters for said multicast groups in isolation from the NS.
- **Differentation algorithm for update compression:** Firmware updates are large payloads that must be split into fragments. That is why this work was quite elaborate on adding redundancy. This way, it is ensured that all fragments can be reconstructed into an update payload. It

has been previously discussed to use algorithms like RSync or RMTD to reduce this large payload beforehand. Alternatively, utilities like *JojoDiff* and *JojoPatch* could be implemented for comparison. This could lead to novel work for an incremental firmware update strategy.

7.2 Conclusions

Due to low network throughput and high packet loss, firmware updates have been identified as a challenge for LoRaWAN devices. This work presents a novel approach to FUOTA sessions for LoRaWAN using Random Linear Network Coding instead of Low-Density Parity Checks. Using rateless generation-based coding like RLNC means that coding configuration is adjustable on-the-fly, which results in feedback and coding adjustments as new possibilities. In our work, the finite field size, generation size, and redundancy coding parameters of RLNC are evaluated concerning the generation decoding success distribution. This results in the knowledge that GF(2^8) has practically nonexistent decoding overhead, where LDPC has 2 % to 10 % overhead dependent on generation size. Also, it leads to the definition of the generation success threshold, a redundancy region where decoding can deal with worst-case packet loss.

To the best of the author's knowledge, available research does not critically evaluate the choice of code in poor network conditions. For that reason, we have developed a testbed that replaces the essential modules of LoRaWAN. This testbed is able to store received measurements, replay encoded fragments remotely, and introduce artificial packet loss to test an application with a fragment coder. This way, we evaluated the influence of poor channel conditions resulting in packet erasure to show how RLNC needs to be configured to restore a certain number of lost packets for a predefined amount of redundancy.

The Gilbert-Elliot model has been applied to analyze the effect is of burst loss (time-correlated packet loss) on RLNC performance. The parameters of this model are normalized (constrained) for the constant mean packet-error rate. Consequently, the performance results of different burst loss conditions have become comparable. Conclusions of these results are that systematic coding will show between 38 % and 88 % lower decoding success rate, depending on burst loss intensity. Since systematic coding is part of the first phase of fragments coded with LDPC in FUOTA, we conclude that RLNC is overall more suitable in any kind of poor network conditions.

Bibliography

[1] A. Krishnan. Lorawan multi-ran architecture connecting the next billion iot devices. Technical report, ABI Research, December 2020.

[2] I. Mavromatis, A. Stanoev, A. J. Portelli, C. Lockie, M. Ammann, Y. Jin, and M. Sooriyabandara. Reliable iot firmware updates: A large-scale mesh network performance investigation. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 108–113, 2022.

[3] PubNub. A new approach to iot security. Technical report, PubNub Inc., 2015.

[4] STMicroelectronics NV. *Application Note AN5554 LoRaWAN® firmware update over the air with STM32CubeWL*, Feb 2022. Rev. 3.

[5] TheThingsNetwork. The things stack fuota process reference. `https://www.thethingsindustries.com/docs/reference/fuota/`. (last visited on 06/22/2022).

[6] J. Jongboom. Firmware-updates enabled lorawan example application. `https://github.com/ARMmbed/mbed-os-example-lorawan-fuota`, 2019. (last visited on 06/20/2022).

[7] M. Luis. Lorawan end-device stack implementation and example projects. `https://github.com/Lora-net/LoRaMac-node`, 2021. (last visited 06/26/2022).

[8] K. Zandberg, K. Schleiser, F. Acosta, H. Tschofenig, and E. Baccelli. Secure firmware updates for constrained iot devices using open standards: A reality check. *IEEE Access*, 7:71907–71920, 2019.

[9] Y. W. Shiferaw. Lorawan class b multicast: Scalability. MSc thesis, Technical University of Delft, September 2019.

[10] D. Heeger, M. Garigan, E. Eleni Tsiropoulou, and J. Plusquellic. Secure lora firmware update with adaptive data rate techniques. *Sensors*, 21(7), 2021.

[11] S. van Nieuwamerongen. Energy consumption and scalability of transmitting firmware updates over lora. MSc thesis, Technical University of Delft, 08 2021.

[12] FUOTA Working Group of the LoRa Alliance Technical Committee. LoRaWAN Fragmented Data Block Transport Specification v1.0.0. Standard, LoRa Alliance, Fremont, CA 94538, USA, September 2018.

[13] FUOTA Working Group of the LoRa Alliance Technical Committee. LoRaWAN Application Layer Clock Synchronization Specification v1.0.0. Standard, LoRa Alliance, Fremont, CA 94538, USA, September 2018.

[14] FUOTA Working Group of the LoRa Alliance Technical Committee. LoRaWAN Remote Multicast Setup Specification v1.0.0. Standard, LoRa Alliance, Fremont, CA 94538, USA, September 2018.

[15] Semtech. Lora® and lorawan®. `https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/`. (Last visited on 06/08/2022).

[16] L. Angrisani, P. Arpaia, F. Bonavolontà, M. Conti, and A. Liccardo. Lora protocol performance assessment in critical noise conditions. In *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, pages 1–5, 2017.

[17] J. Petajajarvi, K. Mikhaylov, M. Pettissalo, J. Janhunen, and J. Iinatti. Performance of a low-power wide-area network based on lora technology: Doppler robustness, scalability, and coverage. *International Journal of Distributed Sensor Networks*, Vol. 13:1–16, 03 2017.

[18] K. Abdelfadeel, T. Farrell, D. McDonald, and D. Pesch. How to make firmware updates over lorawan possible. In *IEEE WOWMOM 2020*, pages 16–25, 2020.

[19] SemTech. *SX1261/2 DataSheet DS.SX1261-2.W.APP*, Dec 2021. Rev. 2.1.

[20] LoRa Alliance Technical Committee. LoRaWAN 1.1 Specification. Standard, LoRa Alliance, Inc, Beaverton, OR 97003, USA, October 2017.

[21] Y. W. Shiferaw, A. Arora, and F. Kuipers. Lorawan class b multicast scalability. In *2020 IFIP Networking Conference (Networking)*, pages 609–613, 2020.

[22] FUOTA Working Group of the LoRa Alliance Technical Committee. FUOTA Process Summary Technical Recommendation TR002 v1.0.0. Standard, LoRa Alliance, Fremont, CA 94538, USA, January 2019.

[23] T. Feng and K. Yang. A nimble decompression algorithm for zigbee firmware update in smart home environment. In *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 630–631, 2017.

[24] A. Hagedorn, D. Starobinski, and A. Trachtenberg. Rateless deluge: Over-the-air programming of wireless sensor networks using random linear codes. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*, pages 457–466, 2008.

[25] K. Arakadakis, P. Charalampidis, A. Makrogiannakis, and A. G. Fragkiadakis. Firmware over-the-air programming techniques for iot networks - A survey. *CoRR*, abs/2009.02260, 2020.

[26] N. Sornin. LoRaWAN:Firmware Updates Over-the-Air. Standard, Semtech Corporation, Camarillo, CA 93012, USA, April 2020.

[27] C. Charilaou, S. Lavdas, A. Khalifeh, V. Vassiliou, and Z. Zinonos. Firmware update using multiple gateways in lorawan networks. *Sensors*, 21(19), 2021.

[28] F. Li, C. Zhang, K. Peng, A. E. Krylov, A. A. Katyushnyj, A. V. Rashich, D. A. Tkachenko, S. B. Makarov, and J. Song. Review on 5g nr ldpc code: Recommendations for dttb system. *IEEE Access*, 9:155413–155424, 2021.

[29] P. Marcelis, V. Rao, and V. Prasad. Dare: Data recovery through application layer coding for lorawan. In *IoTDI 2017: Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, pages 97–108, 04 2017.

[30] T. Ho, M. Medard, R. Koetter, D.R. Karger, M. Effros, J. Shi, and B. Leong. A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, 52(10):4413–4430, 2006.

[31] Y. Rivera, I. Gutiérrez, J. Márquez, R. Porto, and S. Castaño. Performance dynamic coding rlnc lora on smart cities. In *2021 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pages 1–7, 2021.

[32] I. Chatzigeorgiou and A. Tassi. Decoding delay performance of random linear network coding for broadcast. *IEEE Transactions on Vehicular Technology*, 66(8):7050–7060, 2017.

[33] V. Nguyen, J. A. Cabrera, G. T. Nguyen, F. Gabriel, C. Lehmann, S. Mudriievskyi, and F. H. P. Fitzek. Adaptive decoding for fulcrum codes. In *2018 IEEE 9th Annual Information Technology, ElectFronics and Mobile Communication Conference (IEMCON)*, pages 133–139, 2018.

[34] S. Abboud, N. el Rachkidy, A. Guitton, and H. Safa. Gateway selection for downlink communication in lorawan. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2019.

[35] Y. Shiferaw. Lorawan ns-3 module. `https://github.com/yoniwt/lorawan-private`, 2021. (last visited on 05/29/22).

[36] K. Q. Abdelfadeel. Fuotasim. `https://github.com/kqorany/FUOTASim`, 2020. (last visited on 06/10/2022).

[37] J. Heide. Random Linear Network Coding (RLNC)-Based Symbol Representation. Standard, Steinwurf Aps, September 2019.

[38] J. Jeong and D. Culler. Incremental network programming for wireless sensors. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 25–33, 2004.

[39] J. Hu, C. J. Xue, Y. He, and E. H.-M. Sha. Reprogramming with minimal transferred data on wireless sensor network. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, pages 160–167, 2009.

[40] W. Dong, B. Mo, C. Huang, Y. Liu, and C. Chen. R3: Optimizing relocatable code for efficient reprogramming in networked embedded systems. In *2013 Proceedings IEEE INFOCOM*, pages 315–319, 2013.

[41] P. C. van Oorschot A. J. Menezes and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Dec. 1996.

[42] E. Tasdemir, J. A. Cabrera, F. Gabriel, D. You, and F. H. P. Fitzek. Sliding window rlnc on multi-hop communication for low latency. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–6, 2021.

[43] R. S. Katti and R. G. Kavasseri. Secure pseudo-random bit sequence generation using coupled linear congruential generators. In *2008 IEEE International Symposium on Circuits and Systems*, pages 2929–2932, 2008.

[44] D. Blackman and S. Vigna. Scrambled linear pseudorandom number generators. *ACM Trans. Math. Softw.*, 47(4), sep 2021.

[45] D. Zwart. Local uart gateway proxy and configurator for stm32 lora devices. `https://github.com/davidzwa/LoRaConfigurator`, Jun 2022.

[46] D. Zwart. Lora stm32 firmware with embeddedproto interface. `https://github.com/davidzwa/F446_ProtoBufDevice`, Jun 2022.

[47] *Datasheet - STM32F446xC/E - Arm® Cortex®-M4 32-bit MCU*, Jan 2021. DS10693 Rev 10.

[48] STMicroelectronics. *Datasheet - STM32WLE5xx STM32WLE4xx - Multiprotocol LPWAN 32-bit Arm® Cortex®-M4 MCUs*, Mar 2022. DS13105 Rev 10.

[49] O. Brocaar. Chirpstack fuota server. `https://github.com/brocaar/chirpstack-fuota-server`, Jul 2021. (last visited on 06/22/2022).

[50] E. N. Gilbert. Capacity of a burst-noise channel. *The Bell System Technical Journal*, 39(5):1253–1265, 1960.

[51] E. O. Elliott. Estimates of error rates for codes on burst-noise channels. *The Bell System Technical Journal*, 42(5):1977–1997, 1963.

Appendix A

LoRaWAN FUOTA extension

MAC commands are defined for LoRaWAN presented in Figure A.1 for the application to join and use a network with the proper configuration. Figure A.2 shows an extension for this MAC layer for illustration. Finally Figure A.3 presents how a Class C device would join a multicast, time-synchronized, fragmentation session. There are multiple exchanges of messages required to start a FUOTA session.

CID	Command	Transmitted by		Short Description
		End-device	Gateway	
0x02	***LinkCheckReq***	x		Used by an end-device to validate its connectivity to a network.
0x02	***LinkCheckAns***		x	Answer to LinkCheckReq command. Contains the received signal power estimation indicating to the end-device the quality of reception (link margin).
0x03	***LinkADRReq***		x	Requests the end-device to change data rate, transmit power, repetition rate or channel.
0x03	***LinkADRAns***	x		Acknowledges the LinkRateReq.
0x04	***DutyCycleReq***		x	Sets the maximum aggregated transmit duty-cycle of a device
0x04	***DutyCycleAns***	x		Acknowledges a DutyCycleReq command
0x05	***RXParamSetupReq***		x	Sets the reception slots parameters
0x05	***RXParamSetupAns***	x		Acknowledges a RXSetupReq command
0x06	***DevStatusReq***		x	Requests the status of the end-device
0x06	***DevStatusAns***	x		Returns the status of the end-device, namely its battery level and its demodulation margin
0x07	***NewChannelReq***		x	Creates or modifies the definition of a radio channel
0x07	***NewChannelAns***	x		Acknowledges a NewChannelReq command
0x08	***RXTimingSetupReq***		x	Sets the timing of the of the reception slots
0x08	***RXTimingSetupAns***	x		Acknowledges RXTimingSetupReq command
0x09	***TxParamSetupReq***		x	Used by the network server to set the maximum allowed dwell time and Max EIRP of end-device, based on local regulations
0x09	***TxParamSetupAns***	x		Acknowledges TxParamSetupReq command
0x0A	***DlChannelReq***		x	Modifies the definition of a downlink RX1 radio channel by shifting the downlink frequency from the uplink frequencies (i.e. creating an asymmetric channel)
0x0A	***DlChannelAns***	x		Acknowledges DlChannelReq command
0x0B to 0x0C	***RFU***			
0x0D	***DeviceTimeReq***	x		Used by an end-device to request the current date and time
0x0D	***DeviceTimeAns***		x	Sent by the network, answer to the DeviceTimeReq request
0X0E to 0x7F	RFU			
0x80 to 0xFF	Proprietary	x	x	Reserved for proprietary network command extensions

Figure A.1: **LoRaWAN core MAC layer commands. Image source: Semtech[20]**

.

CID	Command name	Transmitted by		Multicast (M) / Unicast (U)	Short Description
		End-device	server		
0x00	***PackageVersionReq***		x	U	Used by the AS to request the package version implemented by the end-device
0x00	***PackageVersionAns***	x		U	Conveys the answer to PackageVersionReq
0x01	***FragStatusReq***		x	U/M	Asks an end-device or a group of end-devices to send the status of a fragmentation session
0x01	***FragStatusAns***	x		U	Conveys answer to the FragSessionStatus request
0x02	***FragSessionSetupReq***		x	U	Defines a fragmentation session
0x02	***FragSessionSetupAns***	x		U	
0x03	***FragSessionDeleteReq***		x	U	Used to delete a fragmentation session
0x03	***FragSessionDeleteAns***	x		U	
0x08	***DataFragment***		x	U/M	Carries a fragment of a data block

Figure A.2: **LoRaWAN Fragmentation MAC layer extension. Image source: Semtech[12]**

.

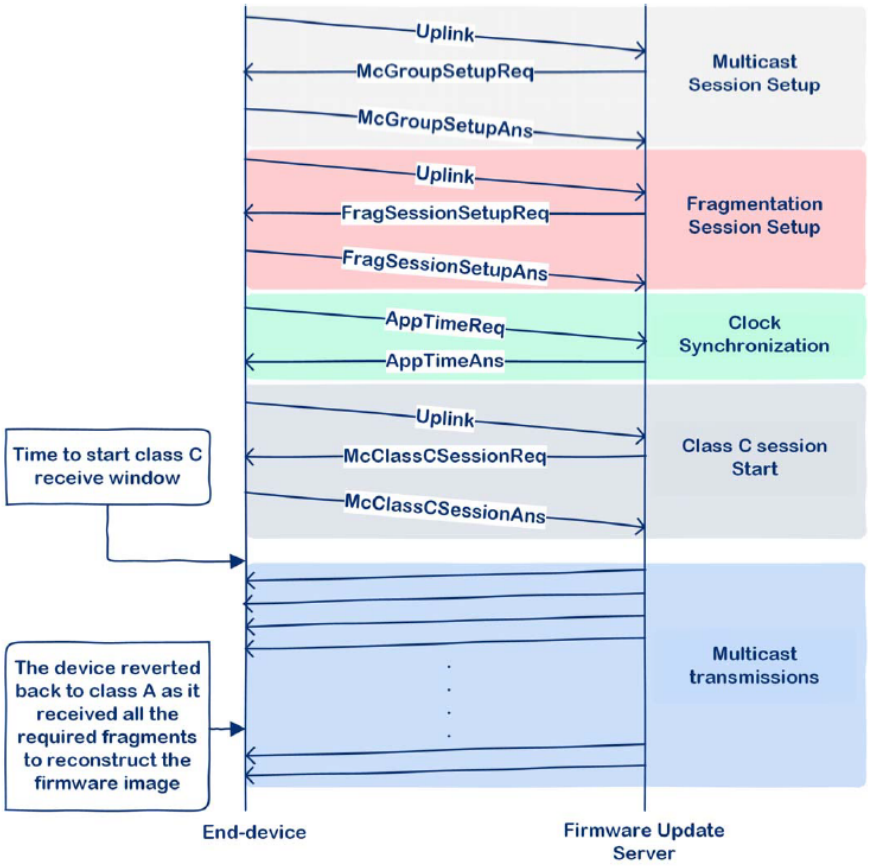


Figure A.3: **FUOTA message exchange overview[18].**