Decoding techniques for cancelling collisions in LoRa

Weixuan Xiao

Clermont Auvergne INP
Université Clermont Auvergne

École Doctorale
Sciences Pour l'Ingénieur

Thèse

Présentée par

Weixuan XIAO

pour obtenir le grade de

Docteur d'Université

Spécialité : Informatique

Techniques de décodage pour annuler les collisions dans LoRa

Thèse prévue pour être soutenue le mardi 20 juin, 2023, devant le jury composé de :

Prof. Martin HEUSSE	ENSIMAG, Grenoble	Rapporteur
Prof. Congduc PHAM	Université de Pau	Rapporteur
Dr. Oana IOVA	INSA Lyon	Examinateur
Dr. Baozhu NING	Semtech	Examinateur
Prof. Thomas NOËL	Université de Strasbourg	Examinateur (Président de jury)
Dr. Gil DE SOUSA	INRAE, Clermont-Ferrand	Encadrant
Dr. Nancy EL RACHKIDY	Clermont Auvergne INP Université Clermont Auvergne	Encadrante
Prof. Alexandre GUITTON	Clermont Auvergne INP Université Clermont Auvergne	Directeur de thèse

Acknowledgements

Words cannot adequately express my deep gratitude to my thesis director, Prof. Alexandre Guitton, and the chair of my committee, Prof. Thomas Noël, for their invaluable patience and constructive feedback throughout this journey.

I extend my sincere appreciation to the members of the jury, Prof. Martin Heusse, Prof. Congduc Pham, Dr. Oana Iova, and Dr. Baozhu Ning, for their keen interest in my study, unwavering support in participating in my defense, and reviewing my manuscript. My heartfelt thanks go to my two supervisors, Dr. Gil De Sousa and Dr. Nancy El Rachkidy, whose unwavering support and valuable guidance have been instrumental in shaping my three-year thesis.

I am grateful for the support received from the Agence Nationale de la Recherche (ANR) of the French government through the program "Investissements d'Avenir" (16-IDEX-0001 CAP 20-25). Additionally, I acknowledge the funding support from the Conseil Régional Auvergne-Rhône-Alpes and the European Regional Development Fund (ERDF) for this project.

I am indebted to Prof. Megumi Kaneko from the National Institute of Informatics (NII) in Tokyo for facilitating the verification of the algorithm on real hardware during my research internship, with access to the NII's equipment. My sincere thanks go to the Open Source community for their exceptional simulation and signal-processing software, which significantly helped my research.

I am grateful to my colleagues, especially my office mates, for their assistance in editing the thesis, their support in my daily life, and their encouragement throughout my time at the university. I extend my appreciation to the librarians, research assistants, and study participants from Université Clermont Auvergne, whose contributions and inspiration have played a vital role in my research.

Lastly, I would be remiss not to mention my family, especially my wife, Qian Huang, whose unwavering support encouraged me to pursue my academic dream. I remember my late sister, Jiahan Liu, who regarded me as her best brother during her lifetime, and my parents, Zhanjun Xiao, and Yanzhi Liu, whose belief in my abilities and competence has been a constant source of motivation. Despite the immense loss of my sister during my Ph.D. thesis due to her illness, their belief and encouragement have kept my spirits and motivation high throughout this process.

I hereby dedicate my academic contributions to science in the loving memory of my late sister, Jiahan Liu.

Summary

Abstract

The development of the Internet of Things (IoT) extends its application to more specific scenarios, such as environmental monitoring with low power consumption. LoRa and LoRaWAN are representative techniques that provide configurable and self-deploying networks. However, deploying more devices in LoRa networks increases the frequency of collisions, which limits the network throughput.

In this thesis, we study the resolution of collisions in LoRa to improve network performance. We investigate and categorize existing collision resolution algorithms to understand their advantages and shortcomings. First, we introduce the concept of uncertainties, which are ambiguities in LoRa frames after the processing of multiple collision resolution algorithms. We propose an algorithm that leverages channel coding in LoRa to resolve these uncertainties. The algorithm serves as a complementary and generic algorithm to improve the decoding performance of existing collision resolution algorithms. The improvements brought by the algorithm are evaluated in a LoRaWAN network simulator. Second, we propose a complete decoding scheme to decode collided LoRa frames in LoRaWAN networks. This decoding scheme requires modifications only on the gateway side of the LoRa network, ensuring compatibility with the LoRaWAN protocol. The decoding scheme tracks the frequency changes of LoRa symbols with virtual sub-slots in LoRa frames and computes symbol edges. Decoding performance depends largely on the number of virtual sub-slots used in decoding. In the presence of uncertainties, the generic algorithm can also decode. We also interpret imperfectness of channels as uncertainties to solve them through processing with the full decoding scheme. The performance of the decoding scheme is evaluated in a LoRa simulator based on GNU Radio, which processes signals captured from real hardware at signal sample levels. Simulation results show that our approach improves network performance. The simulator is also released to empower future development and evaluation of collision resolution algorithms and preamble detection algorithms in LoRa and LoRaWAN networks. We also demonstrate that LoRa frames generated by our simulator are consistent with LoRa frames from commodity LoRa nodes, confirming the realism and proximity of our simulation results to real experiments.

Résumé

Le développement de l'Internet des Objets étend son application à des scénarios plus spécifiques, tels que la surveillance environnementale à faible consommation d'énergie. Lora et LoRaWAN sont les techniques représentatives fournissant des réseaux configurables et auto-déployables. En déployant plus de dispositifs dans les réseaux LoRa, les collisions deviennent plus fréquentes en raison de l'augmentation des transmissions, ce qui limite le débit du réseau.

Dans cette thèse, nous étudions la résolution des collisions dans LoRa, afin d'améliorer les performances du réseau en résolvant les collisions. Nous examinons et catégorisons les algorithmes de résolution de collision existants, afin de comprendre les avantages et les inconvénients de chacun. Tout d'abord, nous introduisons le concept d'incertitudes, qui sont les ambiguïtés dans les trames LoRa après le traitement des différents algorithmes de résolution de collision. Nous proposons un algorithme utilisant le codage de canal dans LoRa pour résoudre les incertitudes. Le but est de fonctionner comme un algorithme complémentaire et générique afin d'améliorer les performances de décodage des algorithmes de résolution de collision existants. Les améliorations apportées par l'algorithme sont évaluées dans un simulateur de réseau LoRaWAN. Deuxièmement, nous proposons un schéma de décodage complet afin de décoder les trames LoRa en collision dans les réseaux LoRaWAN, qui ne nécessite que des modifications du côté passerelle dans le réseau LoRa, en maintenant la compatibilité avec le protocole LoRaWAN. Le schéma de décodage suit les changements de fréquence des symboles LoRa avec des sous-slots virtuels dans les trames LoRa et calcule les bords de symbole. Les performances de décodage dépendent largement du nombre de sous-slots utilisés dans le décodage. En cas de présence d'incertitudes, l'algorithme générique est également capable de décoder. Nous interprétons également les imperfections des canaux en incertitudes, afin de les résoudre par le traitement du schéma de décodage complet. Les performances du schéma de décodage sont évaluées dans un simulateur LoRa basé sur GNU Radio, qui fonctionne au niveau des échantillons de signal et est donc capable de traiter les signaux capturés à partir du matériel réel. Nous montrons par simulations que notre approche améliore les performances du réseau. Le simulateur est également publié pour permettre le développement et l'évaluation futurs des algorithmes de résolution de collision et des algorithmes de détection de préambule dans les réseaux LoRa et LoRaWAN. Nous montrons également que les trames LoRa générées par notre simulateur sont cohérentes avec les trames LoRa provenant des nœuds LoRa commerciaux, ce qui confirme que nos résultats de simulation sont réalistes et proches des expériences réelles.

Contents

List of Figures

Chapter 1

Introduction

LPWANs are wireless networks designed to connect devices over long distances with a low power consumption. LPWANs enable various applications, including real time monitoring, smart metering, smart city, smart transportation and smart logistics [10, 2], among others, such as smart grid, industrial assets monitoring, infrastructure monitoring, agricultural surveillance, wildlife tracking and environmental monitoring [37], etc.

1.1 LPWAN technologies

Several concurrent technologies exist in LPWANs: Sigfox [49], Ingenu [23], NB-IoT [24], and LoRa [43]. While all these technologies are designed to provide low-power, long-range connectivity for IoT devices, there are some key differences and trade-offs between them. Typically, to enable a long communication range, the LPWAN technologies often drastically limit the bit rate and use different frequency bands.

Sigfox. The physical layer of the Sigfox network uses an Ultra Narrow Band wireless modulation on the unlicensed sub-GHz Industrial, Scientific, and Medical (ISM) band, to achieve a bit rate of up to 100 bps. The network layer protocols are not publicly available because the business model of the company operates as an IoT service operator [10]. The technology supports bidirectional communication. However, the payload of packets transmitted by Sigfox is limited to only 12 bytes.

Ingenu. Ingenu is an LPWAN technology owned by On-Ramp Wireless, a company that has been at the forefront of the 802.15.4k standard [10]. The physical layer leverages the patented Random Phase Multiple Access (RPMA) technology. RPMA operates in the 2.4 GHz band with a complex radio environment, as there are potential interference from many other technologies, such as Bluetooth and Wi-Fi [4]. However, this allows RPMA to use a wide bandwidth and a large transmission power (within the limits of the regulation) [4], at the cost of the power consumption.

NB-IoT. NB-IoT is a technology introduced by 3GPP [35]. It leverages the existing infrastructure of the mobile network to transmit data from LPWAN applications through licensed channels. NB-IoT utilizes the single-carrier Frequency-Division Multiple Access (FDMA) for uplink transmissions, which restricts the bit rate to 20 kbps. The Orthogonal Frequency-Division Multiple Access (OFDMA) and Quadrature Phase Shift Keying (QPSK), are used in the downlink [4], which brings up to 200 kbps bit rate.

LoRa. LoRa, which represents Long Range, is a wireless communication technology patented by Semtech and designed for LPWANs. It allows two different modulations: (i) a Chirp Spread Spectrum (CSS) modulation with a low bit rate, from 300 bps to 5500 bps, but with a high robustness, and (ii) a Frequency-Shift Keying (FSK) modulation with a higher bit rate transmission of up to 50 kbps.

Table 1.1 summarizes the main differences between these technologies.

Table 1.1: Comparison of LPWAN technologies.

Technology	Frequency band	Indicative bit rate	Indicative range
Sigfox	ISM (868 MHz/915 MHz)	up to 0.1 kbps	up to 10 km
Ingenu	ISM (2.4 GHz)	up to 30 Mbps (with up to 40 channels)	up to 16 km
NB-IoT	Licensed	20-250 kbps	up to 11 km
LoRa	ISM (868 MHz/915 MHz)	0.3-5.5 kbps (CSS) or 50 kbps (FSK)	up to 16 km

1.2 Focus on LoRa

LoRa is among the most popular LPWAN technologies. One of the reasons is its ability to be highly configurable. Indeed, users of LoRa can adjust a wide range of parameters, including the modulation and the bit rate, in order to optimize the performance of the network for their specific use case. Additionally, LoRa operates in the unlicensed ISM bands, which allows for easy and affordable deployment of devices without requiring a license. Moreover, the LoRa Alliance is promoting the open Long-Range Wide Area Network (LoRaWAN) standard [27], which defines MAC protocol and a network topology for a LPWAN using the LoRa physical layer. LoRaWAN thus provides a standardized way for LoRa devices to communicate over a network, as well as several classes of operation, in order to meet different demands. Lastly, LoRaWAN can operate as both a public or a private network.

In the remainder of this thesis, we focus on LoRa.

1.3 Context of the thesis

This thesis is done in the context of ConnecSenS [31], which is a regional project in Auvergne-Rhône-Alpes (France). ConnecSenS focuses on the modeling of multi-scale dynamic ecosystems and territories. ConnecSenS is building an IoT platform for environmental monitoring in Auvergne, with the objective of comprehending global changes using a sustainable developement approach. The project is funded by the French Contrats de Plan Etat-Régions (CPER), with institutional partners including Centre national de la recherche scientifique (CNRS), University Clermont Auvergne (UCA) and Institut National de Recherche pour l'Agriculture, l'alimentation et l'Environnement (INRAE).

In the ConnecSenS project, a wireless sensor called Sensors Open LoRa Node (SoLo), has been developed in order to monitor the environment for several months without battery change or energy harvesting. Thus, both the energy consumption and the frame delivery rate are crucial for the SoLo devices, and LoRa was selected as their main communication technology.

Our task was to improve the communication capabilities of LoRa, in order to improve the capabilities of the SoLo devices.

1.4 Objective of the thesis

LoRa and LoRaWAN can provide long-range connectivity with a small bit rate. However, they are prone to collisions, due to the ALOHA random access mechanism used in LoRaWAN, especially in dense networks. Indeed, LoRa cannot recover from collisions when multiple devices transmit frames simultaneously using similar parameters. Thus, the frames that are colliding need to be retransmitted, which reduces the throughput while increasing the energy consumption.

Several researchers have proposed to improve the performance of LoRaWAN networks by optimizing the selection of parameters, for instance through modifications of the ADR mechanism of LoRaWAN. However, collisions can still occur in dense networks.

We decided to follow another approach, based on collision resolution, because the indicative bit rate of LoRa is far from the channel capacity given by the Shannon-Hartley theorem. Collision resolution aims to reduce the number of retransmissions by detecting and resolving collisions at the reception. In this way, the throughput is improved, and the energy consumption is reduced. With collision resolution, we believe that we can improve the scalability and reliability of LoRaWAN networks, and make them an attractive option for a wider range of IoT applications.

In this thesis, we focus on the collision resolution in a LoRaWAN network. Our main contribution is a new decoding scheme for LoRa called SF-DS. SF-DS leverages a preamble detection algorithm capable of discovering multiple colliding LoRa preambles. SF-DS is then able to demodulate several cases of colliding

frames, by computing the frequencies several times during each LoRa symbol. In the presence of indistinguishable frames, SF-DS can also take advantage of the redundancies introduced by the LoRa channel coding, in order to attempt to recover some of the frames. We validate SF-DS by an implementation in GNU Radio, which can be directly deployed on a SDR hardware. Our simulation results show that SF-DS outperforms the other protocols from the literature in a large variety of scenarios. Finally, SF-DS requires only modifications at the receiver, and is thus compatible with most LoRaWAN networks.

The work during this thesis was realized in the team of Information and Communication Systems (ICS) of the Laboratory of Informatics, Modelling and Optimization of Systems (LIMOS), at UCA, in Clermont-Ferrand (France). The thesis was directed by Prof. Alexandre Guitton from UCA, supervised by Dr. Nancy El Rachkidy from UCA and Dr. Gil De Sousa from INRAE.

1.5 Outline of the manuscript

The thesis is structured as follows.

Chapter 2 presents the LoRa physical layer, the MAC protocol and network topology of LoRaWAN, and the existing works on collision resolution in LoRa. First, we study the LoRa modulation scheme, the frame structure, the demodulation scheme, and the channel imperfections in LoRa. We explain that LoRa's long-range capability is primarily due to its modulation/demodulation scheme and its ability to withstand noise. We also introduce the channel coding used in LoRa, which is used in LoRa is designed to mitigate the effects of radio channel imperfections. Next, we describe the MAC protocol and network topology of LoRaWAN. We give the format of data frames containing MAC layer information. In addition, we present the LoRaWAN regional settings specifying the parameters used in different regions of the world. Finally, we present the existing works on collision resolution from the literature. We discuss the intrinsic (but limited) capabilities of LoRa to resist collisions, the interference cancellation approaches based on signal power, and the frame tracking approaches. We emphasize the GS-MAC protocol [13], as it is the basis of our proposed SF-DS algorithm.

Chapter 3 proposes the first contribution of the thesis, which is a general algorithm based on LoRa coding. It can be applied to most existing collision resolution algorithms, and attempts to solve uncertainties of unresolved colliding LoRa frames. First, we discuss why existing algorithms are unable to resolve all collisions, and identify situations where existing algorithms may produce uncertain results. Next, we present our core algorithm, and show how it can leverage the current LoRa encoding. We provide an example to illustrate the effectiveness of the algorithm. Then, the limits and drawbacks of the proposed algorithm are described. Finally, we give some implementation details.

Chapter 4 describes the second and foremost contribution of the thesis, which is the SF-DS algorithm. SF-DS is inspired by the existing GS-MAC [13] protocol, but removes its unrealistic assumptions. SF-DS

relies on the detection of frequency changes during LoRa frames in order to decode colliding frames. First, we concentrate on the limitations of GS-MAC, which prevent the protocol from being deployed in a real LoRaWAN network. For each limitation, we discuss the solution used by SF-DS. Next, we present the preamble detection mechanism in ideal conditions. Note that preamble detection is crucial for collision resolution algorithms because it provides essential information about collisions to the decoding algorithm. Finally, we discuss how to decode in non-ideal conditions, by discussing the main sources of imperfection: time offsets, frequency offsets and noise. Note that SF-DS only requires modifications at the receiver, and is thus compatible with legacy LoRaWAN communications.

Chapter 5, presents our third contribution, which is the design of a LoRa simulator based on the GNU Radio platform. The simulator mainly focuses on the physical layer to simulate LoRa signals at the signal sample level. The simulator also includes MAC layer features in order to simulate a large-scale LoRaWAN network. First, we describe the existing LoRa and LoRaWAN simulators from the literature, and show the missing features of the simulators based on our research needs. Next, we present our simulator, which focuses on collision resolution. Our simulator has modular components, including the preamble detection module, the demodulation module, and the frame decoding module. Next, we explain how to simulate a heavy traffic on a large-scale network. Then, we discuss how we improved a LoRaWAN module in NS-3. Finally, we compare a LoRa frame generated by our implementation, with a real LoRa frame.

Chapter 6 regroups most of our simulation results on the comparison of our algorithms with the best algorithms from the literature. We first present the gains of our algorithm based on LoRa coding (described in Chapter 3), when it is used in conjunction with three existing algorithms. Next, we compare several existing algorithms of the literature in the same environment, and we study the influence of the duty-cycle and of the noise level. Finally, we compare SF-DS (described in Chapter 4) with the main existing algorithms from the literature, in various scenarios.

Chapter 7 concludes this manuscript by summarizing our contributions, and by discussing several perspectives.

Chapter 2

LoRa and LoRaWAN

LoRa is a robust modulation that enables long-range communications, and is thus a key enabler for LPWANs. LoRaWAN is largely used in actual LPWAN applications, as it allows the quick and flexible deployment of low-cost networks, including in places that are remote. LoRaWAN defines the upper layers of the protocol stack based on LoRa.

In this chapter, we first describe in details the physical layer of LoRa. Then, we briefly describe the MAC layer and network topology of LoRaWAN. Finally, we explain how collisions can be handled in LoRa.

2.1 LoRa: a robust physical layer

LoRa is a physical layer targeting long range wireless communications, and patented by the Semtech company in [40]. LoRa consists of a robust modulation and a resilient coding technique: both enable LoRa to achieve long range communications.

In this section, we first introduce how signals are modulated with LoRa. Then, we describe how LoRa frames can be demodulated. Third, we present the coding and decoding used by LoRa frames. Finally, we discuss some aspects of the noise robustness of LoRa.

2.1.1 Modulation of LoRa signals

LoRa uses a CSS modulation[1]: the LoRa modulator encodes data using a series of linear chirps, also called symbols. Each chirp consists of a linear frequency sweep over time within a Bandwidth (BW). The duration of a chirp is called Symbol Duration (SD), and is given as follows:

$$SD = \frac{M}{BW} = \frac{2^{SF}}{BW}, \tag{2.1}$$

[1]In the remainder of this manuscript, we do not consider the FSK modulation of LoRa.

where $M = 2^{SF}$ depends on the very important SF parameter, indicating the number of bits that are transmitted per symbol. Increasing SF by one adds one bit per symbol, but doubles the symbol duration. Overall, the theoretical bit rate R_b of the LoRa CSS modulation is defined as follows:

$$R_b = \frac{SF}{SD} = \frac{SF}{2^{SF}} BW. \tag{2.2}$$

In the LoRa modulation, up-chirps and down-chirps represent linear chirps with increasing and decreasing frequency, respectively. In the base band, *i.e.* when the frequency of the symbol is centered at 0 kHz, the frequency of the chirps varies within $[-\frac{BW}{2}; \frac{BW}{2}]$. A normalized up-chirp is an up-chirp whose initial frequency starts at the minimum value, which is $-\frac{BW}{2}$. A normalized down-chirp is a down-chirp whose initial frequency starts at the maximum value, which is $\frac{BW}{2}$. Thus, in the base band, the frequency f_u of a normalized up-chirp and the frequency f_d of a normalized down-chirp are as follows:

$$f_u(t) = \frac{t}{SD} BW = \frac{M}{SD^2} t, t \in [-\frac{SD}{2}, \frac{SD}{2}), \tag{2.3}$$

$$f_d(t) = -\frac{t}{SD} BW = -\frac{M}{SD^2} t, t \in [-\frac{SD}{2}, \frac{SD}{2}). \tag{2.4}$$

To encode a symbol value $m \in [0; M-1]$ in an up-chirp (respectively down-chirp), an offset is applied to the initial frequency of the chirp. The frequency starts increasing (respectively decreasing) from the shifted initial frequency. When the frequency reaches $BW/2$ (respectively, $-BW/2$), the frequency returns to the minimum value $-BW/2$ (respectively the maximum value $BW/2$) in order to remain within BW. Let us denote by $\tau_m = \frac{m}{M} SD$ the moment within a chirp when the frequency reaches the BW boundary. Therefore, the instantaneous frequency $f_u^m(t)$ of an up-chirp encoding symbol value m, and the instantaneous frequency $f_d^m(t)$ of a down-chirp encoding symbol value m, are defined as follows:

$$f_u^m(t) = \begin{cases} \frac{M}{SD^2}(t + \tau_m), t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{M}{SD^2}(t + \tau_m) - \frac{M}{SD}, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}), \end{cases} \tag{2.5}$$

$$f_d^m(t) = \begin{cases} -\frac{M}{SD^2}(t - \tau_m), t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ -\frac{M}{SD^2}(t - \tau_m) + \frac{M}{SD}, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{2.6}$$

Note that if $m = 0$, Equation 2.5 and Equation 2.6 become Equation 2.3 and Equation 2.4, respectively. We can thus denote f_u^0 as the frequency of the normalized up-chirp, and f_d^0 as the frequency of the normalized down-chirp.

Figure 2.1 shows an up-chirp encoding symbol value $m = 48$, with SF7 and $BW = 125$ kHz. The x-axis represents the time for one symbol duration SD, with $SD = 2^{SF}/BW = 2^7/(125.10^3) = 1.024$ ms. The y-axis represents the frequency, which is within $[-BW/2; BW/2] = [-62.5 \text{ kHz}; 62.5 \text{ kHz}]$. The initial frequency of this up-chirp encoding symbol value $m = 48$ (as shown on the right-hand y-axis), is $(m/M).BW - BW/2 = (48/2^7) * 125.10^3 - 125.10^3/2 = -15625$ Hz (as shown on the left-hand y-axis).

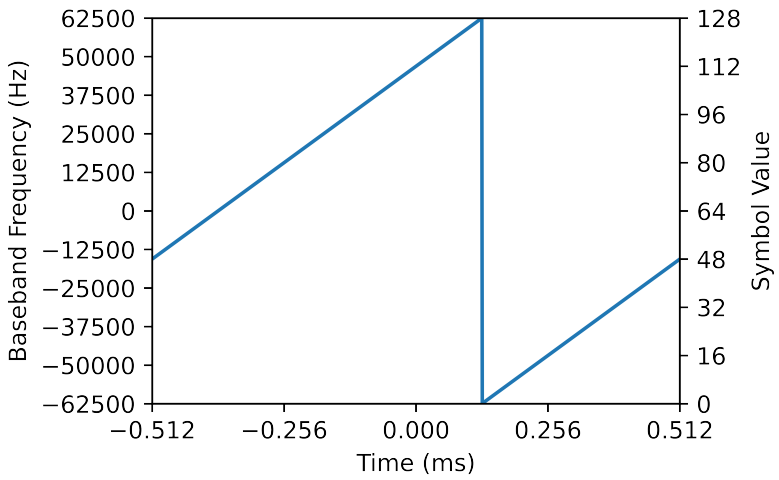


Figure 2.1: An up-chirp encoding symbol value 48 with SF7 and $BW = 125$ kHz.

The actual base band signal of a chirp in the time domain can be represented as follows:

$$s(t) = e^{j\phi(t)}, \tag{2.7}$$

where $\phi(t)$ is the phase of the chirp, and j is the imaginary unit. Given the initial phase ϕ_0 at the beginning of the chirp, the instantaneous phase of a chirp f^m can be described as follows:

$$\phi(t) = \phi_0 + 2\pi \int_{-\frac{SD}{2}}^{t} f^m(\tau)\, d\tau. \tag{2.8}$$

Let us consider a normalized up-chirp $f_u = f_u^0$. We have:

$$\begin{aligned} s_u^0(t) &= e^{j\phi_0 + j2\pi \int_{-\frac{SD}{2}}^{t} f_u(\tau)\, d\tau} \\ &= e^{j\phi_0 + j2\pi \int_{-\frac{SD}{2}}^{t} \frac{M}{SD^2}\tau\, d\tau} \\ &= e^{j\phi_0 + j2\pi \frac{M}{2SD^2}\tau^2 |_{-\frac{SD}{2}}^{t}} \\ &= e^{j\phi_0 + j2\pi (\frac{M}{2SD^2}t^2 - \frac{M}{2SD^2}\frac{SD^2}{2^2})} \\ &= e^{j\phi_0 + j2\pi (\frac{M}{2SD^2}t^2 - \frac{M}{8})}. \end{aligned} \tag{2.9}$$

which is the instantaneous phase of the signal in the base band.

Using Euler's formula, Equation 2.7 can be written as:

$$s(t) = \cos\phi(t) + j\sin\phi(t). \tag{2.10}$$

The real part of $s(t)$ is called the in-phase component, also called the I-component of the signal. The imaginary part of $s(t)$ is called the quadrature component, also called Q-component of the signal.

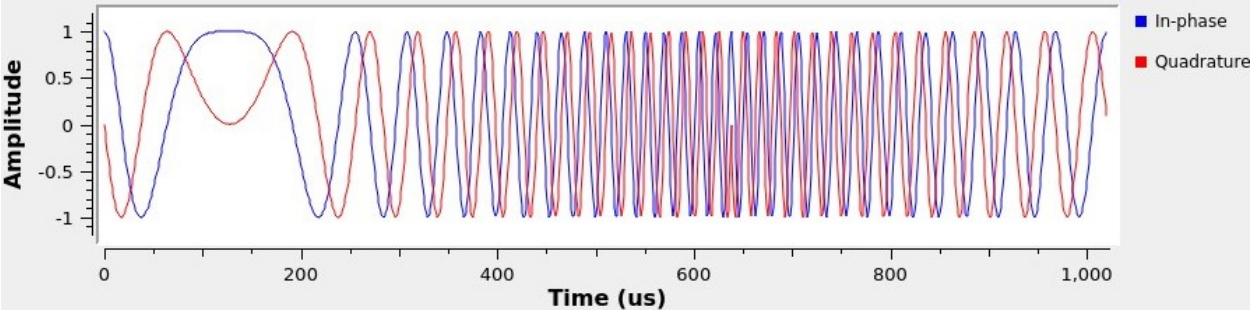


Figure 2.2: The I/Q components of an up-chirp encoding symbol value 48 with SF7 and $BW = 125$ kHz.

Figure 2.2 shows the I and Q components of a LoRa up-chirp encoding symbol 48 with SF7 $BW = 125$ kHz, with an initial phase $\phi_0 = 0$. Note that the symbol duration SD is 1.024 ms as in Figure 2.1, and the amplitude is normalized in $[-1; 1]$.

A LoRa frame can be either an uplink[2] or a downlink[3]. A LoRa uplink consists of a preamble, an optional header and a payload, as shown on Figure 2.3. The preamble consists of several normalized up-chirps, typically eight, plus two up-chirps called the sync word and acting as a network identifier, plus two and a quarter down-chirps representing the SFD. The optional header and the payload follow. A LoRa downlink is similar, except that the I and Q components of up-chirps and down-chirps in the frame are inverted, which results in down-chirps and up-chirps, respectively.

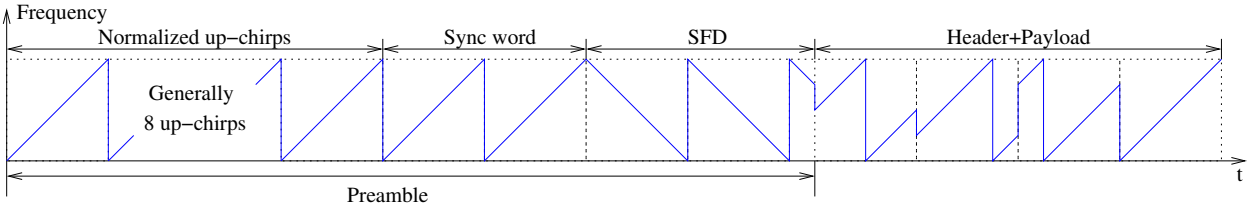


Figure 2.3: An example of an uplink, which consists of eight normalized up-chirps, two up-chirps representing the sync word, two and a quarter down-chirps for the SFD, and four up-chirps for the header and the payload.

2.1.2 Demodulation of LoRa signals

In order to demodulate the symbols of a frame, a LoRa receiver needs to be synchronized with the transmitted symbols. This synchronization is achieved by detecting the repeated symbols of the preamble, until it detects the inversion of the chirps of the SFD. The receiver ensures that the sync word corresponds to the symbol value it expects. Then, the receiver can start decoding the header and payload.

[2]Uplinks are used in LoRaWAN for frames being sent from an end-device to a gateway.
[3]Downlinks are used in LoRaWAN for frames being sent from a gateway to an end-device.

Ideal case: the channel and the synchronization are perfect

Let us consider here the ideal case where the channel and the synchronization are perfect.

The demodulation of the header and payload starts by multiplying each received symbol by a normalized conjugated chirp: for uplink frames, each received up-chirp is multiplied by a normalized down-chirp, and for downlink frames, each received down-chirp is multiplied by a normalized up-chirp. Such multiplication of signals in complex form (see Equation 2.7) can be interpreted as an addition of the frequencies of the chirps. This removes the time-variance of the frequency in the original chirp, and is called a de-chirp. Indeed, after the de-chirp, the instantaneous frequencies of the up-chirp described in Equation 2.5 becomes as follows:

$$f_u^m(t) + f_d(t) = \begin{cases} \frac{M}{SD^2}\tau_m, t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{M}{SD^2}\tau_m - BW, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{2.11}$$

Note that t disappears here compared to Equation 2.11, and $f_u^m(t) + f_d(t)$ gives two constants frequencies separated by BW. Figure 2.4 shows the in-phase and quadrature components after the de-chirp of the up-chirp from Figure 2.2. Note that there are only two frequencies in the symbol duration SD.

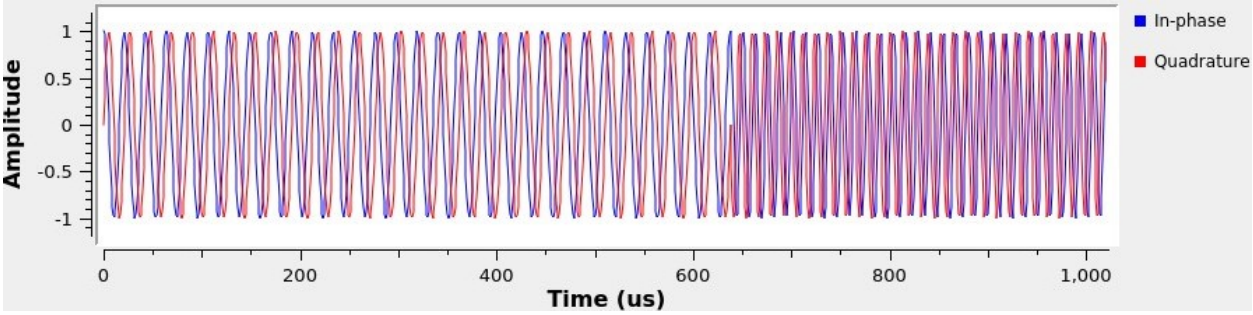


Figure 2.4: The I/Q components of an up-chirp modulating symbol 48 with SF7 and $BW = 125$ kHz, after the de-chirp process (that is, after the multiplication by a normalized down-chirp).

Finally, the demodulator needs to extract the symbol value based on the two resulting frequencies components, for each symbol. To show the relationship between these two frequency components and the value, let us rewrite Equation 2.11 with the expected symbol value m rather than with τ_m, as follows:

$$f_u^m(t) + f_d(t) = \begin{cases} \frac{m}{M}BW, t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{m}{M}BW - BW, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{2.12}$$

The two frequency components only depend on the symbol value m and on the bandwidth BW, which means that m becomes easy to obtain, in theory.

However, estimating m in practice requires several samples of the LoRa signal. The receiver processes several samples on the received signal, with a sample rate $f_s = \frac{1}{T_s}$, T_s being the sampling time. After

sampling, the continuous signal becomes a discrete signal that is observed by the receiver. Then, the demodulator performs an n-point Discrete Fourier Transform (DFT) during each period SD to extract the two frequency components. The DFT takes n samples of the signal in the time domain as input, and transforms them into n elements in the frequency domain, corresponding to the amplitude of each frequency component. In practice, the demodulator can leverage the FFT algorithm, which is a quick implementation of DFT. The time complexity of FFT is $O(n \log n)$, while a naïve implementation of the DFT has a time complexity of $O(n^2)$. Note that if the demodulator uses BW as the sampling rate f_s, the number of samples n during each SD is given as follows:

$$n = SD \times f_s = SD \times BW = \frac{M}{BW} BW = M \tag{2.13}$$

Therefore, the result of the FFT on each symbol contains M items, where each item represents a frequency component with an interval of BW/M.

In order to deduce the symbol value m from the results of the FFT of a symbol, let us first consider the positive frequency component of Equation 2.12, which varies between 0 and $BW/2$. After the FFT, the first item represents the energy distribution of frequency 0, the second item represents the energy distribution of frequency $\frac{1}{M}BW$, and so on. The energy distribution depends on the power levels of the signals and the synchronization. The index of each item thus corresponds to the symbol value m. In this case, the demodulator can simply treat the index of the maximum frequency component within the bandwidth BW as the demodulated symbol value m during each SD. In brief, the demodulator can take the index of the strongest FFT peak as the estimation of symbol value during each SD. Figure 2.5 shows an example of the result of the FFT of a symbol encoding the symbol value 1.

According to the Nyquist-Shannon sampling theorem [36], frequencies recognizable with a sample rate of $f_s = BW$ are limited to the range $[-BW/2, BW/2]$, which is symmetrical around frequency 0. This sample rate produces what is known as aliasing, which limits the highest absolute frequency to a value modulo BW. In the FFT result, the power distribution of positive frequencies is represented by indices from 0 to $n/2-1$, while negative frequencies are represented by the rest, with indices varying from $n/2$ to $n-1$. In the case of LoRa demodulation, aliasing makes the two cases of Equation 2.11 identical in the band-limited frequency domain, as shown in Equation 2.14.

$$\begin{aligned} Aliasing(f_u^m(t) + f_d(t)) &= \frac{m}{M} BW \mod BW \\ &= \begin{cases} \frac{m}{M} BW, m \in [0, \frac{M}{2}), \\ \frac{m}{M} BW - BW \mod BW = \frac{m}{M} BW, m \in [\frac{M}{2}, m). \end{cases} \end{aligned} \tag{2.14}$$

Therefore, if $f_s = BW$ and the synchronization is perfect, the demodulator should observe only one FFT peak during the given SD, enabling it to obtain the correct symbol value of m.

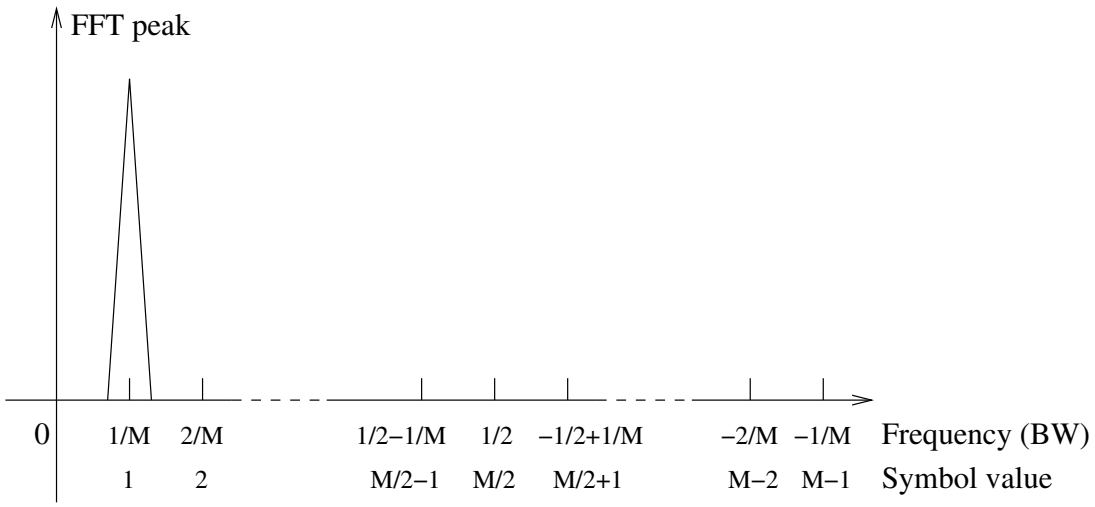


Figure 2.5: An example of FFT peak located at $\frac{1}{M}BW$ frequency, corresponding to the symbol value $m = 1$. The frequencies below $\frac{1}{2}$ represent the positive frequency component in the base band, while the frequencies above $\frac{1}{2}M$ represent the negative frequency component.

We hereby give the formal representation of LoRa demodulation as follows:

$$\hat{m} = \arg\max\left(DFT(S(n) \times conj(S^0(n)))\right), \tag{2.15}$$

where $\hat{m}$ is the estimation of the modulated symbol value m, $S(n)$ is the discrete signals of a symbol (either an up-chirp or a down-chirp) after sampling on the receiver side, *conj* is the conjugate of a chirp, and $S^0(n)$ is the normalized chirp.

The demodulated symbol values during each period SD can then be passed to the decoder, as explained in Subsection 2.1.3.

Realistic case: the channel and synchronization are imperfect

Now, let us consider a more realistic case, where the channel and the synchronization are imperfect. In other words, we now focus on the demodulation in the presence of noise, time offset and carrier frequency offset.

Noise. In a LoRa network, noise is often considered as an Additive White Gaussian Noise (AWGN) with zero mean [16]. Let us denote by A_{signal} and P_{signal} the Root Mean Square (RMS) amplitude and the power of a received signal, and A_{noise} and P_{noise} the RMS amplitude and the power of the noise. The SNR is described as follows:

$$SNR = \frac{P_{signal}}{P_{noise}} = (\frac{A_{signal}}{A_{noise}})^2. \tag{2.16}$$

Therefore, the SNR in logarithmic decibel scale (dB) is defined as follows:

$$SNR[dB] = 20\log_{10}\frac{A_{signal}}{A_{noise}} = 10\log_{10}\frac{P_{signal}}{P_{noise}}. \tag{2.17}$$

LoRa benefits from the fact that the white noise has an expected uniform distribution in the frequency domain, which means that white noise produces similar power levels at each frequency component after performing the FFT. Thus, after the multiplication with a conjugate chirp, the noise is distributed into all FFT bins, while the actual symbols of the LoRa signal concentrate on a single FFT bin. As a result, the demodulator might still be able to deduce the correct symbol value by taking the strongest peak, even when the signal is significantly weaker than the noise (that is, when $SNR < 0$).

Figure 2.6, Figure 2.7 and Figure 2.8 show the waveform of the symbol modulating value 32 (on the top) with SF7 (on the left) and SF12 (on the right), the corresponding FFT results (on the bottom), and the estimated symbol values from channels with a good SNR (20 dB, corresponding to a weak noise), a weak SNR (0 dB, corresponding to a large noise) and a very weak SNR (-20 dB, corresponding to a very large noise) for LoRa.

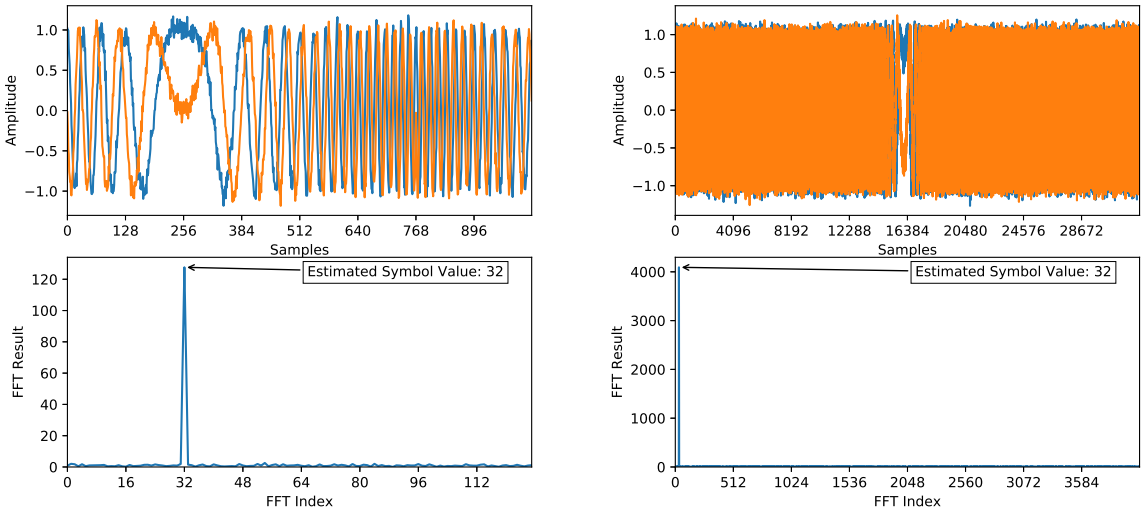


Figure 2.6: Waveform and FFT results of symbol 32 encoded with SF7 (on the left) and SF12 (on the right), under a weak noise ($SNR = 20\ dB$).

For Figure 2.6, with a good SNR of 20 dB, the waveform is relatively clear in the time domain for SF7. After performing the de-chirp and FFT, there are significant peaks with both SF7 and SF12. The receiver is able to demodulate 32 as the correct symbol value.

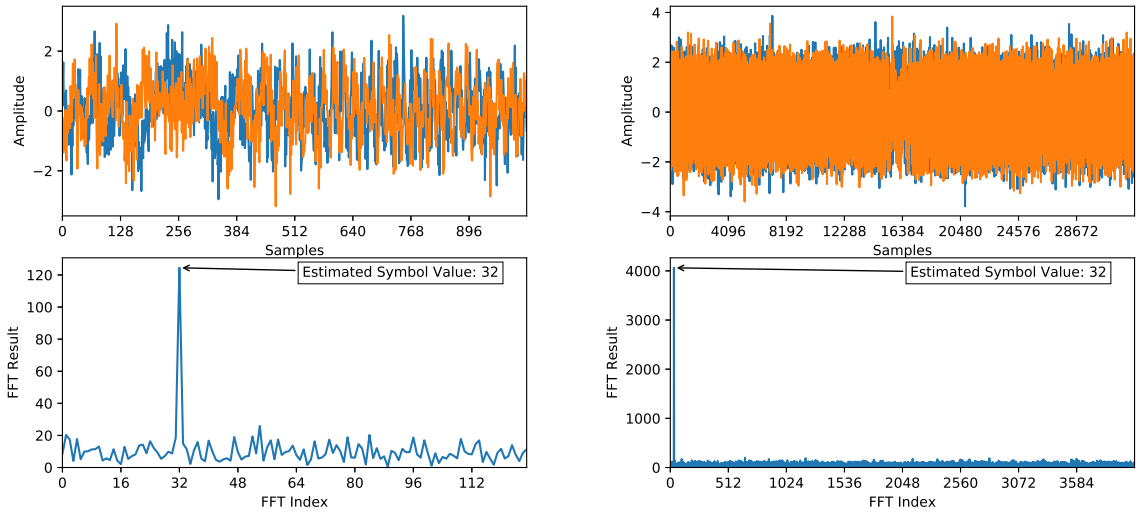


Figure 2.7: Waveform and FFT results of symbol 32 encoded with SF7 (on the left) and SF12 (on the right), under a large noise ($SNR = 0\ dB$).

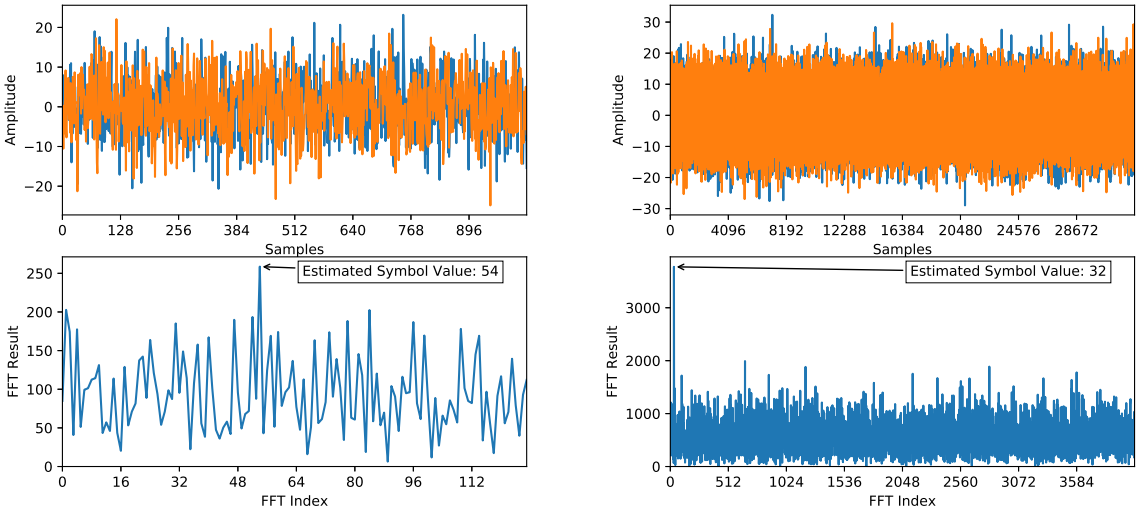


Figure 2.8: Waveform and FFT results of symbol 32 encoded with SF7 (on the left) and SF12 (on the right), under a very large noise ($SNR = -20\ dB$).

For Figure 2.7, with a weak SNR of 0 dB that is when the average power of the AWGN equals to the power of a LoRa symbol, the waveform becomes hard to recognize. However, the peaks in the frequency domain are still recognizable. Thus, the receiver is still able to demodulate the correct symbol value with both SF7 and SF12.

For Figure 2.8, with a very weak SNR of -20 dB, the waveform is unrecognizable as a LoRa signal. The FFT result of SF7 shows that the receiver is no longer able to demodulate the correct symbol value due to the noise, and obtains 54 instead of 32. Indeed, -20 dB is below the threshold for demodulating a symbol with SF7, as this threshold is equal to -6 dB. However, although the peak in the FFT result with SF12 becomes relatively lower, the receiver can still demodulate the symbol value 32. Note that -20 dB is equal to the threshold for correct demodulation with SF12.

Time offset. In case of desynchronization during demodulation, the time offset can cause a shift in the frequency component, which can weaken the energy of the desired frequency component. To illustrate the effect of frequency component offset, let us assume that the time offset is given by $\tau_\sigma = \frac{\sigma}{M} SD$. In this case, the first equation in Equation 2.5 becomes:

$$\begin{aligned} f_u^m(t+\tau_\sigma) &= \frac{M}{SD^2}(t+\tau_\sigma+\tau_m) \\ &= \frac{M}{SD^2}(t+\frac{m}{M}SD) + \frac{M}{SD^2}\frac{\sigma}{M}SD \\ &= f_u^m(t) + \frac{\sigma}{SD}. \end{aligned} \tag{2.18}$$

After removing the time-variance through the de-chirp process, the estimated symbol value of the corresponding frequency becomes $m' = m + \sigma$ instead of m. The same derivation also applies for the second case of Equation 2.5.

Desynchronization can weaken the energy strength of the desired frequency component, as a part of the current chirp is truncated from the symbol duration due to the time offset and replaced by another chirp (either the previous chirp, in case of positive time offset, or the next chirp, in case of negative time offset) . Thus, the desynchronization weakens the FFT peak of the corresponding frequency, due to the missing part of the current chirp. This weakening can hinder the demodulation of a LoRa signal, especially when the SNR is low.

Figure 2.9 illustrates the impact of time offset with an example. We apply FFT to two continuous symbols (48 and 0, encoded with SF7) using two different windows: Win 1, correctly synchronized, and Win 2, slightly desynchronized. The FFT result for Win 1 represents the result for the perfectly synchronized case. The demodulated symbol value of the first symbol is 48. Win 2 is delayed by a quarter SD, Win 2 is delayed by a quarter of SD, resulting in a desynchronized case. The FFT result shows two main peaks: the strongest is 80 and the weaker is 32. The symbol value 80 corresponds to the first symbol 48, with an offset of a quarter of 2^{SF} $(48 + 32 = 80)$. Its energy is also weakened due to the missing first quarter. The

symbol value 32 corresponds to a quarter of the second symbol 0, with an offset of three quarters of 2^{SF} $(0 + 128 - 96 = 32)$.

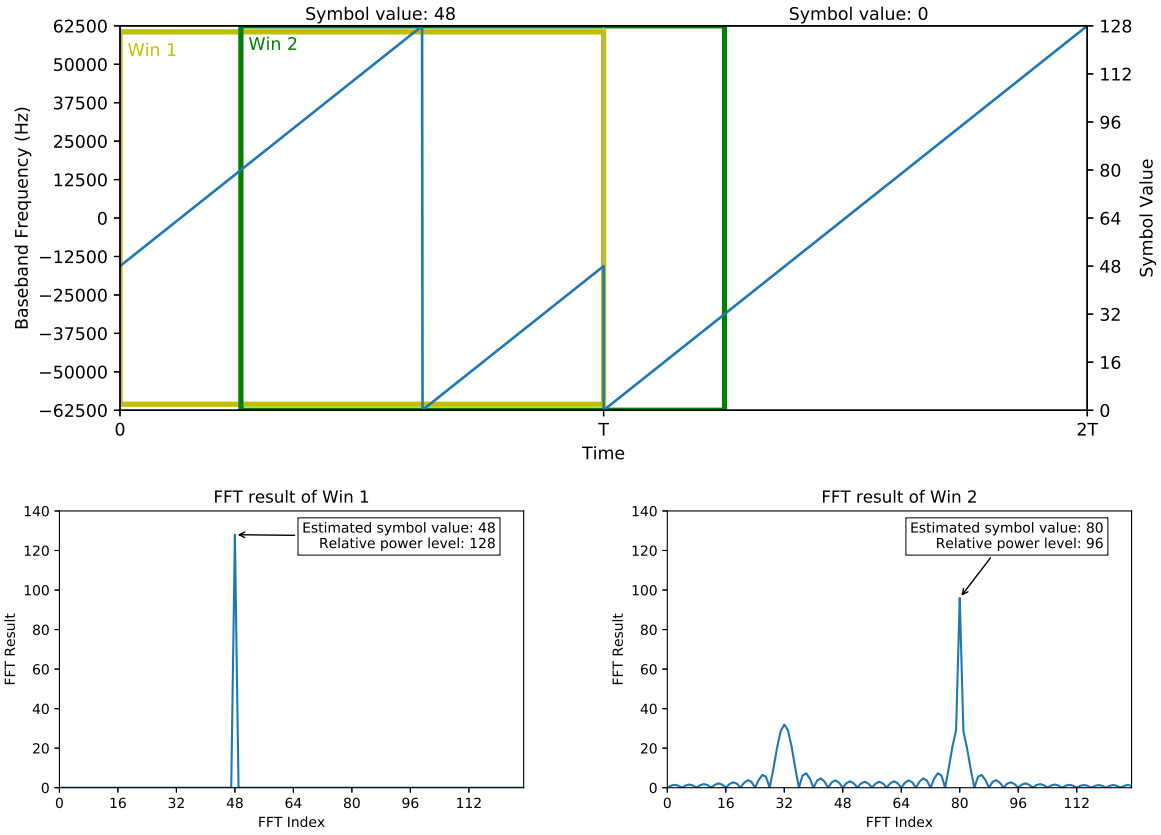


Figure 2.9: FFT results for two windows, while demodulating symbol 48 followed by symbol 0, encoded with SF7. Win 1 represents a window correctly synchronized with the symbol edge of the frame, which results in a single symbol value 48, and the energy strength 128 for the first symbol. Win 2 is desynchronized with the frame by $0.25T$, which results in a weaker peak at FFT index 80 (since $80 = 48 + 0.25 \times 2^{SF} = 48 + 32$) in the FFT result, and a small peak at FFT index 32 (since $32 = (0 - (1 - 0.25) \times 2^{SF}) \mod 2^{SF} = 128 - 96$).

Carrier frequency offset. A CFO may occur between the transmitter and the receiver. The CFO is a difference in the central frequency of the channel, caused by the frequency mismatch of low-cost crystal oscillators [17] of the transmitter and receiver. Given an oscillator error e_O and the frequency of a channel f_B, the carrier frequency offset δf_{CFO} can be computed using the formula as follows:

$$\Delta f_{CFO} = e_O \times f_B, \tag{2.19}$$

where e_O is contributed by the oscillators from both the transmitter and the receiver. Such an error can change each symbol value by $m_{CFO} = \frac{f_{CFO}}{BW} M$. Thus, the frequency in the first case of Equation 2.5 becomes

as follows:

$$\begin{aligned} f_u^m(t) + f_{CFO} &= \frac{M}{SD^2}(t + \tau_m) + f_{CFO} \\ &= \frac{M}{SD^2}(t + \frac{m}{M}SD) + \frac{m_{CFO}}{M}BW \\ &= \frac{M}{SD^2}(t + \frac{m}{M}SD) + \frac{M}{SD^2}\frac{m_{CFO}}{M}SD \\ &= \frac{M}{SD^2}(t + \frac{m}{M}SD + \frac{m_{CFO}}{M}SD) \\ &= f_u^{m+m_{CFO}}(t) \end{aligned} \tag{2.20}$$

which results in a shifted symbol value $m + m_{CFO}$ for each symbol.

Considering that m_{CFO} contains an integral part and a fractional part, we can divide m_{CFO} into m^i_{CFO} and m^f_{CFO}, such that $-0.5 < m^f_{CFO} \leq 0.5$ and $m^i_{CFO} = m_{CFO} - m^f_{CFO}$. In order to correct the CFO and obtain the transmitted symbol values, the receiver has to correct each symbol value obtained from the demodulated symbols, after demodulation by adding (respectively, subtracting) for up-chirp (respectively, down-chirp) m^i_{CFO}. This necessitates the knowledge of the CFO before the demodulation of the symbols in the payload. Authors in [17] showed a detection algorithm for the CFO, based on the combination of up-chirps and down-chirps from the preamble. While the specific mechanism used in LoRa receivers is unknown, it is likely that typical LoRa demodulators implement a similar algorithm to handle CFO within $[-25\%; 25\%]$ of the BW.

From the impacts of time offset and carrier frequency offset, we can conclude that timing and frequency offsets between transmitter and receiver are equivalent. This feature can greatly reduce the complexity of the receiver design and its cost.

2.1.3 Coding and decoding

The theoretical bit rate of LoRa modulation is given by Equation 2.2. However, LoRa introduces channel coding in order to improve the robustness to noise and errors.

The LoRa coding scheme consists of four operations that transform the bitstream generated by the application into the values encoded in symbols: (1) The transmitter first adds redundancy using a Hamming coding, in order to be able to detect or correct errors. (2) Then, it whitens the data sequence. (3) Then, it applies a diagonal interleaver, whose role is to distribute the bits over several symbols. (4) Finally, the symbols are encoded using a Gray decoding scheme. The LoRa decoding scheme performs the inverse operations in reverse order, in order to transform the received values into the original bitstream.

Figure 2.10 shows the overview of each operation, including the inputs and the outputs.

Hamming coding, and error detection or correction. Redundancy is added by the transmitter for ECC. To do this, LoRa coding takes effect on each nybble, composed of 4 bits from the MAC frame. In

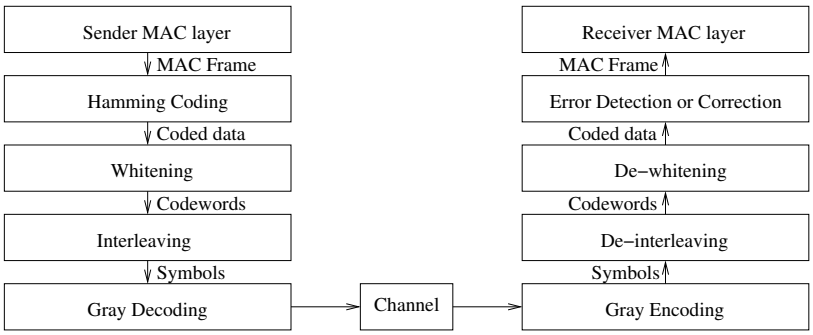


Figure 2.10: Overview of the LoRa coding/decoding process in a transmission.

the encoding stage, a parameter named Coding Rate (CR), and varying from 1 to 4, controls the number of redundant bits to add to each nybble. The redundant bits are derived from each nybble using Hamming coding and concatenated with it to generate a codeword. Several CRs are possible: 4/5 (corresponding to CR1), 4/6 (corresponding to CR2), 4/7 (corresponding to CR3), and 4/8 (corresponding to CR4), which results into codewords of 5 to 8 bits. On the receiver side, the nybbles can be easily retrieved from codewords to obtain the original data. Single-bit errors on each codeword can be detected with CR1 and CR2, and can be corrected with CR3 and CR4.

Whitening and de-whitening. The whitening and de-whitening operation is done by performing an XOR operation with a specific sequence on each codeword. This operation is applied to avoid potential disturbances called direct current component [45], which consists of sequences of continuous 0 or 1.

Interleaving and de-interleaving. LoRa uses an interleaver to distribute the encoded bits into several symbols, so that a short-span error on a few consecutive bits is translated into a 1-bit error on several codewords, which is easy to recover using the redundancy. The interleaver used in LoRa is called diagonal, because it scatters the bits of a codeword, represented as a line in an input matrix, onto a diagonal of the input matrix [30]. More specifically, the LoRa interleaver takes as input SF codewords, each having $CR+4$ bits. These bits are organized into a matrix of $SF \times (CR+4)$ bits, where each codeword is a line, as shown on Figure 2.11. The interleaver outputs a matrix of $(CR+4) \times SF$ bits, where each line represents SF bits, used as a symbol value in the next step. The output matrix is filled line by line. The i-th line of the output (with $i \in [1; CR+4]$) corresponds to the $(CR+4+1-i)$-th column of the input matrix. The bits are filled in the upward direction (with wrapping), and starting on the bit at line $(i-2) \mod SF+1$.

Let us consider the example of Figure 2.11, with CR1 and SF7.

- Line $i = 1$ of the output matrix corresponds to column number $(CR+4+1-i) = 5$ of the input matrix. The bits are taken in the upward direction, from line $(i-2) \mod SF+1 = 6+1 = 7$. Thus, the output line is 0110010.

- Line $i = 2$ corresponds to column 4. The bits are read in the upward direction, from line $(i - 2)$ mod $SF + 1 = 0 + 1 = 1$. Thus, the line starts with 1. Then, the wrapping occurs because the top of the input column is reached, and the line continues with the bottom of the input column, which is 101111.

- Line $i = 3$ corresponds to column 3, starting at line 2. The line starts with 00 and continues after the wrapping with 00101.

- Line $i = 4$ corresponds to column 2, starting at line 3. The line starts with 101 and continues with 1111.

- Line $i = 5$ corresponds to column 1, starting at line 4. The line starts with 1001 and continues with 111.

Let us reexplain the same example of Fig. 2.11 in another way, in order to highlight the use of the diagonal. Let us consider the bits in the third column of the codewords. The bit sequence starts from the bit above the diagonal, starting with 0 from the second codeword 0x03. It then takes the 0 from the first codeword 0x1A, and continues taking the 0 from the last codeword 0x1A, until it reaches the 1 from the third codeword 0x0E. Therefore, the final bit sequence is "0000101" from the Most Significant Bit (MSB) to the Least Significant Bit (LSB). After interleaving, the third symbol consists of this bit sequence, which represents 5 as the symbol value.

The de-interleaver performs the reverse operation, and converts symbol values back into codewords.

Note that the interleaver actually used in LoRa has been reverse-engineered in [38], and differs slightly from the one described in the LoRa patent. Indeed, the actual LoRa interleaver uses an upward direction, while the patent describes the interleaver with a downward direction. Our description relied on the actual implementation, that is it assumes an upward direction.

LoRa introduces a specific mode called low data rate optimization, which is used only with SF11 and SF12. Indeed, when using these large SFs, the symbol duration for $BW = 125$ kHz also becomes too long (respectively 16.384 ms and 32.768 ms), which can cause synchronization issues between the transmitter and the receiver, and corrupt the decoded data. To prevent this, the low data rate optimization mode introduces small modifications in the interleaver: it is applied on $SF - 2$ codewords (instead of SF) and still generates $CR + 4$ symbols per step, but with $SF - 2$ bits each (instead of SF). Then, the symbols are left-shifted by 2 bits, and filled by 0 for the two bits. In this way, the decoding is improved by tolerating synchronization errors of up to 4 in terms of symbol values, corresponding to these 2 bits.

Gray decoding and encoding. The last step of LoRa encoding is a Gray decoding, and is performed in order to limit the number of bit errors when mis-identifying a symbol. Let us consider the worst scenario

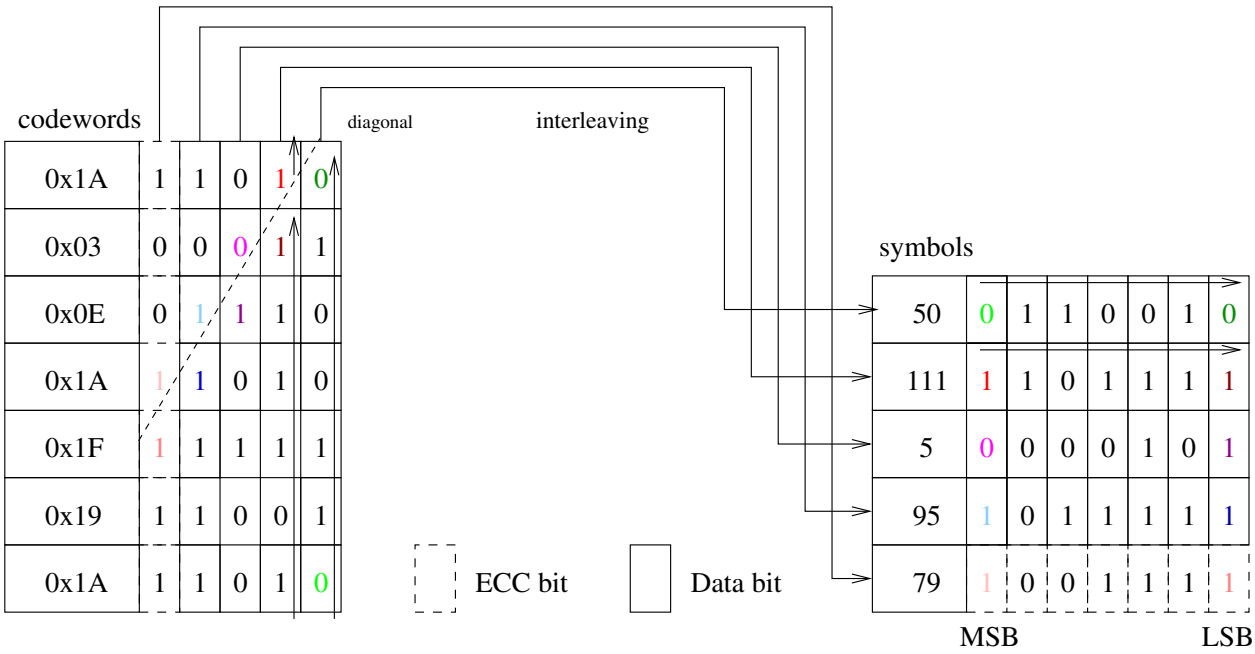


Figure 2.11: Example of the diagonal interleaver applied to a codeword block with $SF = 7$ and $CR = 1$.

without Gray decoding, where a transmitter is sending symbol value 63 on SF7, which is written 0111111 in binary. In presence of noise, a receiver might assume that the symbol value is 64, which is written 1000000 in binary. With this error of only one symbol value, the receiver produced an error of 7 bits. In contrast, with Gray decoding, consider the same scenario, where a transmitter is sending symbol value 63 on SF7, which is written 0111111, and becomes 0100000, that is 32, after Gray decoding. In presence of noise, the receiver might assume that the symbol value is 33, which is 0100001. This value corresponds to 62 after Gray encoding during the LoRa decoding, which produces an error of only a single bit.

Similarly, the first step of the LoRa decoding is a Gray encoding.

Overall, the use of Gray decoding and encoding limits the number of erroneous bits when the estimated received symbol is close to the real transmitted symbol. Such an error in the estimation can occur due to a slight shift in frequency for instance.

2.1.4 LoRa robustness to noise

In the following, we discuss the robustness to noise of LoRa.

SNR ratio as a function of the SF. Authors in [58] show the SNR thresholds for correct decoding of a frame under the presence of noise for different SFs. The corresponding noise levels are given in Table 2.1. The SNR value indicates that a symbol is likely to be correctly decoded if its SNR is above the given value.

Table 2.1: The SNR threshold and the corresponding noise level of LoRa for different SFs.

SF	12	11	10	9	8	7
SNR (dB)	-20.0	-17.5	-15.0	-12.0	-9.0	-6.0
A_{noise}/A_{signal}	10.00	7.50	5.62	3.98	2.82	2.00

Comparison with Sigfox. According to the analytical comparisons in [15], LoRa (without coding) outperforms Sigfox by 3.5 dB, when the required Bit Error Rate (BER) is 10^{-3}. In terms of SER, the analysis and the experimental results of [15] also show that each SF increment brings 3 dB gain to reach the same SER.

Performance of LoRa coding. Results on LoRa coding are given in the analysis of [3]. Their numerical results show that the gain on the Frame Error Rate (FER) between the SFn and SF$n \pm 1$ is about 3 dB.

2.2 LoRaWAN: a simple network architecture and MAC protocol

The LoRaWAN Alliance designs and maintains a protocol called LoRaWAN (Long-Range Wide Area Network), which describes a simple network architecture with an MAC based on ALOHA random access and built on top of the LoRa physical layer.

In this section, we first introduce the basic architecture of a LoRaWAN network. Next, we describe the MAC protocol as well as the three classes of end-devices . Each class corresponds to different application requirements. Then, we present the parameters defined in LoRaWAN for different regions. Finally, we discuss some works on the performance of LoRaWAN.

2.2.1 Network architecture

The basic components of a LoRaWAN network are: low-powered end-devices, gateways, a NS, a Join Server (JS), and several application servers, as shown on Figure 2.12.

The end-devices send uplink frames to all gateways that are reachable, using LoRa. Gateways keep listening on multiple channels and multiple SFs for uplink frames sent by end-devices. When a gateway detects the preamble of an uplink frame, it allocates a demodulator whose role is to demodulate and decode the frame. Then, each receiving gateway sends a packet containing the frame to the NS using the Internet Protocol (IP) backhaul. The NS removes duplicated frames, and forwards the packet to the application server corresponding to the sender identifier, again using the IP backhaul. Downlink packets are sent either by the application server and transit through the NS (for data packets), or by the NS (for control packets). The NS chooses a gateway and forwards the packet to it. Then, the gateway uses LoRa to send the downlink

frame to the end-device. Note that the JS is used by the NS when new end-devices join the network.

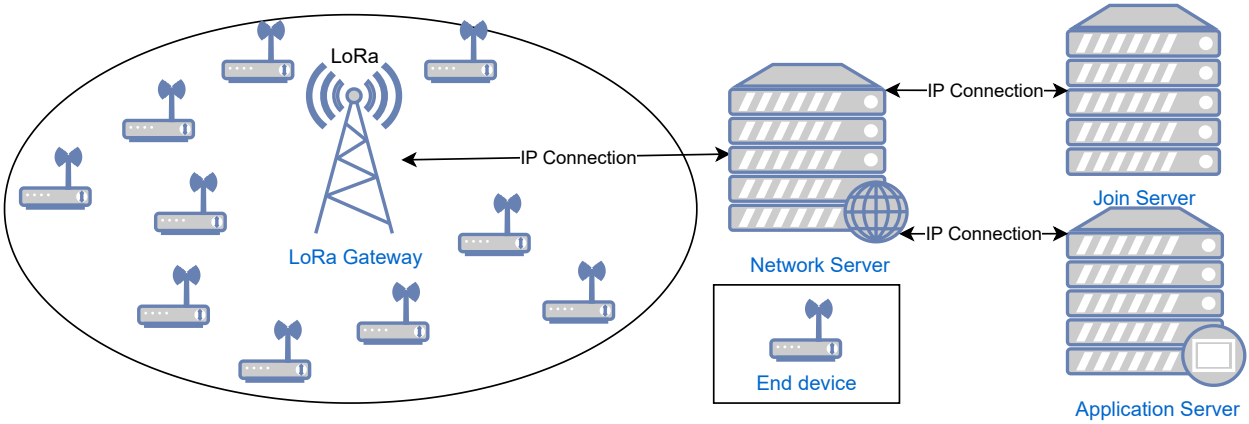


Figure 2.12: A typical LoRa network based on LoRaWAN architecture, with a single gateway.

LoRaWAN provides flexibility to create a private LoRaWAN network. Users can choose to deploy all the network, including the end-devices, the gateways, the NS, the JS and the application server. Users can also deploy the end-devices and the gateways with their own application servers, by using the NS and JS from a network provider. Otherwise, users can purchase only the end-devices and the application server. In this case, they have to use the NS, the JS and the gateways from a LoRaWAN network provider, either from a public network such as The Things Network, or from commercial operators.

For security consideration, LoRaWAN uses end-to-end encryption to protect the privacy and confidentiality of the data transmitted over the network. The encryption is based on the Advanced Encryption Standard (AES) algorithm in counter mode, which is a widely used symmetric-key encryption algorithm. Each device is assigned a unique 128-bit AES encryption key that is used to encrypt the data after successfully joining a network. The application server also has access to this key for decryption. Therefore, only the end-device that transmits the frame and the application server know the information in the frame, while the NS cannot access them in the frame. LoRaWAN uses a message integrity code to ensure the integrity of the transmitted data. The end-devices of LoRaWAN also share a symmetric key with the NS in order to ensure that control messages are confidential.

2.2.2 MAC protocol and classes of end-devices

Classes of end-devices. LoRaWAN defines three classes of end-devices: Class A, Class B and Class C. Class A is mandatory for all the end-devices, while Class B and Class C are optional.

Class A is dedicated to low-power communications. An ALOHA random access mechanism is used: end-devices can transmit at any time. To do so, the end-device randomly chooses a channel and transmit its uplink frame. The end-device then waits for a potential response by opening two short receive windows called

RX1 and RX2, respectively one and two seconds after the end of the transmission, as shown on Figure 2.13.

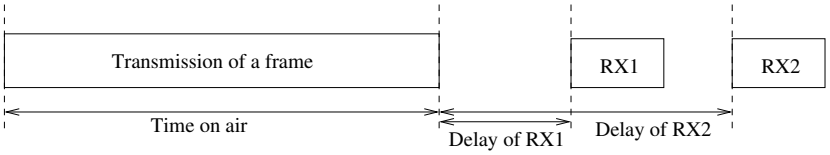


Figure 2.13: Receive windows RX1 and RX2 in Class A of LoRaWAN.

Class B allows delay-guaranteed downlink communications, and is implemented using a beacon-based mechanism. LoRa gateways periodically and synchronously broadcast beacons. When receiving a beacon, each end-device sets up short downlink windows called ping slots. These windows are open in addition to the receive windows of Class A.

Class C allows fast downlink communications by assuming that end-devices do not have energy constraints. The end-devices can thus keep listening at the channels to receive downlink frames at any time.

These three classes are designated to suit different application scenarios. Note that Class A is the one consuming the smallest amount of energy, and is thus by far the most frequent class used in LoRaWAN.

Data frames. The structure of LoRaWAN frames is shown in Table 2.2. The physical layer payload (PHYPayload) can have several types, including data frames, join-request frames, join-accept frames, and rejoin-request frames.

Table 2.2: Format of LoRaWAN frames.

Preamble	PHDR (Packet header)	PHDR_CRC (Packet header CRC)	PHYPayload	CRC

Data frames are used to transfer data from higher layers, including MAC commands. Both the data and the commands can be combined in a single data frame. In the MAC command part, a flag indicates whether or not the uplink data frame requires an acknowledgement. If the acknowledgement request bit is set, the transmitting frame is called confirmed transmission. If it is not set, the frame is called unconfirmed transmission. The data frames can be thus categorized into four types according to the direction of transmission and the requirement of the acknowledgement: confirmed uplink, unconfirmed uplink, confirmed downlink, unconfirmed downlink. Note that uplink data frames optionally contain a CRC for the error detection.

Join-request, join-accept and rejoin-accept frames. Before sending data frames, an end-device must be activated. The activation procedure can be performed using the Activation By Personalization (ABP) procedure or the Over-The-Air Activation (OTAA) procedure. The ABP procedure does not involve any frame exchange, but requires the end-device to contain a priori knowledge of the network. The OTAA procedure involves exchanging join-request, join-accept and rejoin-request frames.

Join-request frames are sent by end-devices to initiate the procedure to join a LoRaWAN network. The join-request frame contains an 8-byte JS identifier called JoinEUI, an 8-byte end-device identifier called DevEUI, and a 2-byte counter. The JoinEUI is a global identifier which identifies the JS used for the join procedure. The DevEUI is a global identifier for the end-device. The counter counts the times that the current end-device has sent join-request frames. When receiving this frame, the NS forwards the data to the JS. If the end-device is permitted to join the network, the JS allocates a device address for the end-device and sends a join-accept frame via the NS (and through a gateway).

Join-accept frames contain a 3-byte counter, a 3-byte home-network identifier, a 4-byte device address, and some downlink settings. The end-device uses this information to derive the network session key (used to exchange confidential control messages with the NS) and the application session key (used for end-to-end confidentiality with the application server).

Rejoin-request frames can be sent by end-devices in order to initialize a new session context. Such frames contain one byte for the type of rejoin, a 3-byte home-network identifier, an 8-byte DevEUI, and a 2-byte counter for the number of rejoin requests sent. Like in the join procedure, the JS replies with a join-accept frame if the end-device is permitted to join the network with a new context.

Adaptive data rate. LoRaWAN defines an ADR mechanism that allocates the data-rate to end-devices, and thus their spreading factor and bandwidth, as a function of the channel quality. The ADR is able to achieve a trade-off between throughput (and thus energy consumption) and communication range. To do so, the NS keeps records of the SNR of communications between each end-device and each gateway. Once the NS has enough records, it computes the best data rate for the end-device by considering the average channel quality, with a margin. Note that the best data rate is the one that ensures a high probability of frame reception with the best throughput.

2.2.3 Regional parameters

LoRaWAN uses regional parameters in order to comply with regional regulations on the use of the wireless medium. For instance, the European regional parameters for LoRaWAN in the band 868 MHz is called EU868 [44]. It operates the unlicensed ISM band of 863-870 MHz. It defines three sub-channels by default for uplink communications: one at 868.10 MHz, one at 868.30 MHz, and one at 868.50 MHz. Downlink communications happen either on the same channel as the uplink (for RX1) or on a specific channel at 869.525 MHz. RX1 is opened one second after the end of the transmission, and RX2 is opened one second after the beginning of RX1. EU868 defines seven Data Rate (DRs) for LoRa communication, from DR0 to DR6. The bandwidth for each channel is 125 kHz for all DRs, except for DR6 for which it is 250 kHz. SF is equal to 7 for DR6, and varies from 7 to 12 for DR5 down to DR0. Thus, the indicative physical bit rate varies between 11000 bps for DR6, reaching as low as 250 bps for DR0. Moreover, due to European

regulations, LoRaWAN implements a duty-cycle of 1% on all channels except the specific channel of RX2, and of 10% on specific channel of RX2: after each transmission of duration d, a LoRaWAN device (either an end-device or a gateway) has to wait during $(100/dc - 1)d$ time units before the next transmission, where dc is the duty-cycle expressed in percent.

2.2.4 LoRaWAN performance

There has been a large number of performance studies for LoRaWAN. In this subsection, we describe a few of them.

In [2], the authors showed that the duty-cycle is a key limiting factor for the maximum frame rate of an independent end-device. Moreover, they explained that the ALOHA random access mechanism, which does not attempt to avoid collisions, does not scale well with the number of end-devices: when the number of end-devices is large, both the throughput per end-device and the overall throughput decrease rapidly.

In [52], the authors studied the latency of Class A end-devices in a LoRaWAN network. They have shown that the latency depends primarily on the duty-cycle, and secondarily on the frame transmission time.

In [54], the authors developed a LoRaWAN module for the NS-3 network simulator. The main distinguishing feature of this module is the integration of the energy model. The authors provide simulation results of the energy consumption per end-device along with the frame delivery rate.

In [9], the authors developed an NS-3 module to simulate thousands of end-devices in a LoRaWAN network, with either confirmed or unconfirmed traffic. The simulated gateway has the same parallel decoding capabilities as the Semtech's SX1301 chip, that is that they can demodulate up to eight frames in parallel. Their simulations study the frame success rate with different percentages of end-devices using confirmed traffic, with a varying number of end-devices. The simulation results show that the confirmed traffic brings negative influence on the throughput. Besides, the authors also studied the causes of frame failure, which can be either interference or non-availability of decoding paths in the gateway. The authors show that the decoding paths are saturated quickly, and cause an increasing number of failures as the number of end-devices increases. The authors improved the module and studied several additional parameter settings in [28].

2.3 Dealing with collisions in LoRa

Since 2017, a few works have focused on resolving collisions in LoRa. These collisions are quite likely to occur due to the MAC protocol defined in LoRaWAN. Each collision has negative impacts on network performance, reducing throughput and often causing end-devices to retransmit collided frames. Thus, performance metrics are degraded, with increased energy consumption and delay for the confirmed transmissions. Since throughput is extremely low and delay is often large in LPWANs, it is crucial to resolve these collisions.

In [64], we proposed a categorization of collision resolution algorithms for LoRa, which includes several algorithms from the state-of-the-art. Our categorization defines two main types of collision resolution algorithms: those based on power and those based on time. The power-based approach is mainly concerned with the capture effect of frames with different power levels, while the time-based approach leverages the desynchronization between collided frames to distinguish them.

In this section, we present collision resolution protocols from the literature. We first describe how legacy LoRa deals with collisions. Then, we describe existing algorithms that focus on collision resolution in LoRa, starting with the power-based approach, where algorithms are based on interference cancellation, and then moving on to the time-based approach, where algorithms are based on frame tracking. Finally, we provide a brief summary of these algorithms. Note that Subsections 2.3.2, 2.3.3, and 2.3.4 are summarized from our survey paper [64].

2.3.1 Partial collision resolution in legacy LoRa

LoRa is able to recover from some cases of collisions, thanks to its robustness. For instance, transmissions on different channels are orthogonal. We detail here several types of collisions that LoRa can deal with.

Inter-SF quasi-orthogonality

LoRa communications on different SFs were initially said to be orthogonal [42], meaning that they do not cause interference with each other. This was due to the fact that signals with different SFs have different symbol durations: when demodulating a superposed signal according to a reference symbol duration, the time-variant of the signal with the incorrect symbol duration is not removed, and thus its energy is distributed into several FFT bins.

However, authors in [11] have shown that this orthogonality is good but not perfect, and it is now referred to as a quasi-orthogonality. When an interfering LoRa signal with SF SF_{int} is much stronger than the desired signal with SF SF_{ref}, with $SF_{int} \neq SF_{ref}$, the demodulator may wrongly take the wrong FFT index and thus demodulate the wrong symbol. Such errors break the orthogonality. Authors in [11] gave the thresholds of the power difference, for each pair of SF, both by simulation and experimentation. Note that this power difference varies from -7 dB to -25 dB for superposed LoRa signals, from SF6 to SF12.

Capture effect

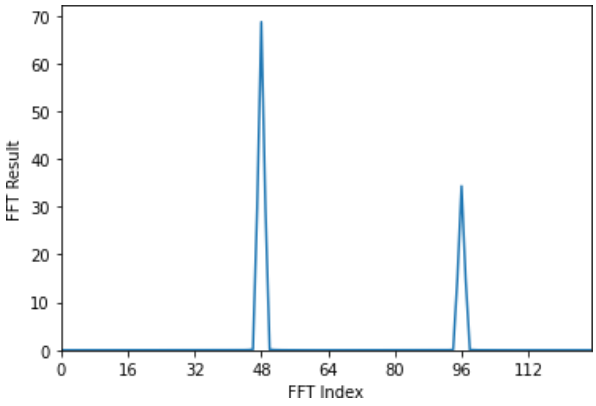
LoRa can leverage the capture effect to correctly demodulate and decode the strongest LoRa signal, even if it is superposed with another signal on the same SF and channel. Two cases can bring the capture effect during the collisions of LoRa frames:

- *Different power levels:* Figure 2.14(a) shows the two peaks produced during a single symbol duration of two frames that are perfectly synchronized: symbol 48 is received with a large power, while symbol 96 is received with a low power. The demodulator outputs the strongest peak, that is the peak with value 48. Due to the power difference in the two frames, it is likely that all the other symbols of the strong frame will be captured too, which illustrates the capture effect of LoRa.

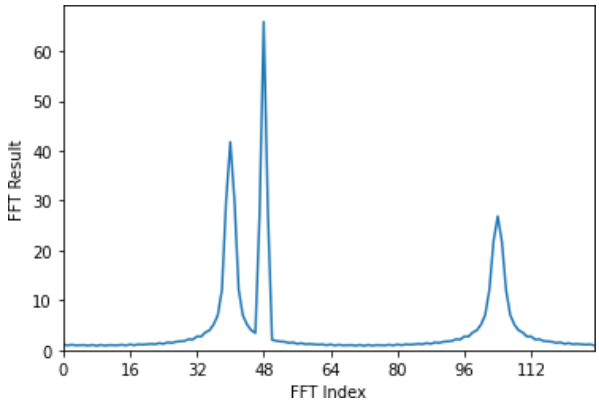
- Power differences from desynchronization: Figure 2.14(b) shows the three peaks produced during a single symbol duration of two slightly desynchronized frames, with the receiver synchronized on the first frame: symbol 48 is the symbol of the first frame, symbol 32 is the end of a symbol of the second frame, and symbol 96 is the beginning of the next symbol of the second frame. Note that the values corresponding to the symbols of the second frame are slightly shifted, due to the delay of the second frame. Even though these two frames are considered here to be received with similar power, the time offset between the frames makes the peaks of the second frame appear weaker. The demodulator outputs the strongest peak, that is value 48, and the correct symbol is decoded. Again, it is likely that all the symbols of the first frame will be captured.

Indeed, due to the nature of ALOHA random access, the frames are seldom synchronized. Therefore, both of the power level and the desynchronization help the decoder to capture one frame, if possible. However, since the weaker frames are lost, they have to be retransmitted eventually.

Authors in [51] showed that a capture effect can happen if the power of the strongest frame exceeds the power of the weakest frame by a threshold varying from 7.5 dB for SF7 to 22.5 dB for SF12.



(a) Symbol 48 and symbol 96 from two synchronized frames. The frame with symbol 48 is received with twice the power of the other frame.



(b) Symbol 48 from the synchronized frame, and symbol 32 followed by symbol 96 from a desynchronized frame, both frames having the same power level.

Figure 2.14: FFT peaks of two superposed frames with SF7.

Quasi-orthogonality of uplink and downlink modulations

Up-chirps and down-chirps are orthogonal, even if they use the same SF. However, it has been shown in [39] that uplink and downlink frames are not orthogonal, due to the fact that uplinks (respectively, downlinks) contain a few down-chirps (respectively, up-chirps), which interfer with most chirps of downlinks (respectively, uplinks) frames.

To the best of our knowledge, FlipLoRa [65] is the only protocol that uses this orthogonality to resolve collisions. By alternating up-chirps and down-chirps during the payload modulation of a frame, the likelihood of two colliding frames not being decoded is decreased. Figure 2.15 illustrates the overlay of a red frame and a blue frame, both utilizing alternating up-chirps and down-chirps. The receiver is synchronized to the red frame, while the blue frame is delayed by 1/4-th of a symbol. During the first red symbol, the receiver multiplies the signal with a down-chirp and identifies two FFT peaks: a high peak for the red up-chirp symbol and a small peak for the partial up-chirp of the blue frame. During the second red symbol, the receiver multiplies the signal with an up-chirp and extracts two FFT peaks: the high peak for the red symbol and a small peak for the blue down-chirp section. The conditions required for this approach to work effectively are: the time interval between frames should minimize the overlap between the up-chirp (and down-chirp) symbols of different frames, and simultaneous downlink communications must be limited since they are less orthogonal with uplink communications than in legacy LoRaWAN.

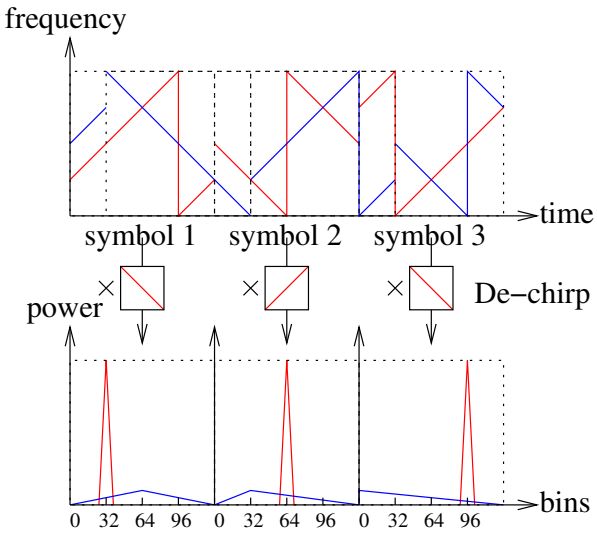


Figure 2.15: Frequency changes of up-chirps and down-chirps from two superposed LoRa frames modulated using FlipLoRa. The receiver is synchronized with the red frame, starting with a full up-chirp and followed by a full down-chirp and a full up-chirp. The blue frame arrives earlier, starting with the end of an up-chirp, and then having a full down-chirp. The orthogonality between the up-chirps and the down-chirps makes it easier to estimate the symbol values during each SD.

2.3.2 Collision resolution based on interference cancellation

SIC was first proposed in [32] to leverage the capture effect and improve the network performance. The authors give an analysis of SIC based on the capture effect of LoRa. In [26], the authors implement SIC on real hardware to decode two superposed frames: the receiver recursively decodes the frame with the strongest signal, reproduces the corresponding signal, and removes it from the received superposed signals in order to decode weaker signals.

For example, if the receiver captures the frames containing the symbol value 48 in both Figures 2.14a and 2.14b, it can then reproduce the ideal signal corresponding to this frame, and subtract it from the superposed signals. Although there can still be some residues, the receiver is likely to be able to capture recursively the second strongest signal. The corresponding symbol values are 96 in Figure 2.14a, and are 32 and 96 in Figure 2.14b.

The main drawback of SIC is that the capture effect cannot be applied when the superposed frames have similar power: indeed, in this case, the demodulator is unable to identify the correct FFT peak from several peaks of similar strengths. Note that SIC requires a difference of power varying from 7.5 dB for SF7 to 22.5 dB for SF12 [51]. In addition, such power-based approaches need to store the entire signal from the earliest frame to the latest frame, which requires a large amount of memory.

Authors in [55] proposes CoLoRa, which is the main collision decoding algorithm of the literature based on SIC.

2.3.3 Collision resolution based on frame tracking

We consider an example of two collided frames encoded with SF7, as shown in Figure 2.16. The synchronized frame, which is the blue frame, contains 16 and 80 as symbols, while the other frame, which is the orange frame, is desynchronized by $SD/2$, and consists of symbols 0 (partially shown), 48 and 32 (partially shown). This example will be used to describe several existing algorithms.

CHOIR

CHOIR [14] uses tiny frequency offsets in order to distinguish frames. The authors show that these tiny offsets come from hardware imperfections of the transmitters, and are stable during the whole frame duration. In order to capture them, the authors apply the FFT on a large window (whose size is ten times the original size).

Figure 2.17 shows the tiny frequency offsets in Frame 1 and Frame 2 discovered by CHOIR. The receiver detects a peak at frequency 160: it is able to deduce that symbol 160/10=16 is from a frame with 160%10=0 as its tiny frequency offset, while both symbols 647/10=64 (translating to 0 for the second frame) and 1127/10=112 (translating to 48 for the second frame) are from another frame, because they have 7 as their

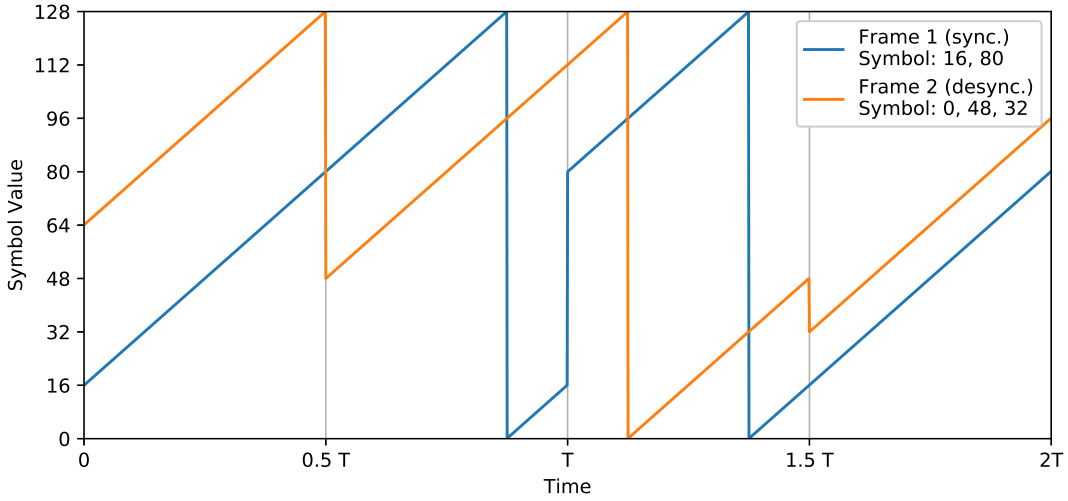


Figure 2.16: Frequency changes of up-chirps from two superposed LoRa frames modulated with SF7. The receiver is synchronized with Frame 1, which consists of symbols 16 and 80. Frame 2 has a delay of $SD/2$, and contains half of symbol 0, the entire symbol 48, and half of symbol 32.

tiny frequency offset.

Limits. While this approach distinguishes the transmitters and decodes collisions of a few frames, it does not scale with the number of concurrent transmitters.

FTrack

FTrack [59] uses the time offset among colliding frames to identify them. When the receiver uses the dechirp operation, each symbol becomes a constant frequency over SD, called a track by the authors. As the frequency of the symbols of a frame only changes at each symbol edge, FTrack uses these symbol edges to match the symbols to the correct frame.

Figure 2.18 shows the tracks of superposed symbols from the example of Figure 2.16. The length of the tracks depend on the steps and the sizes of the window. On this example, it is easy to reconnect the tracks, and thus to deduce the whole frames.

Limits. The main drawback of FTrack is that it requires a sufficient time offset between colliding frames, which has to be greater than one tenth of the symbol duration.

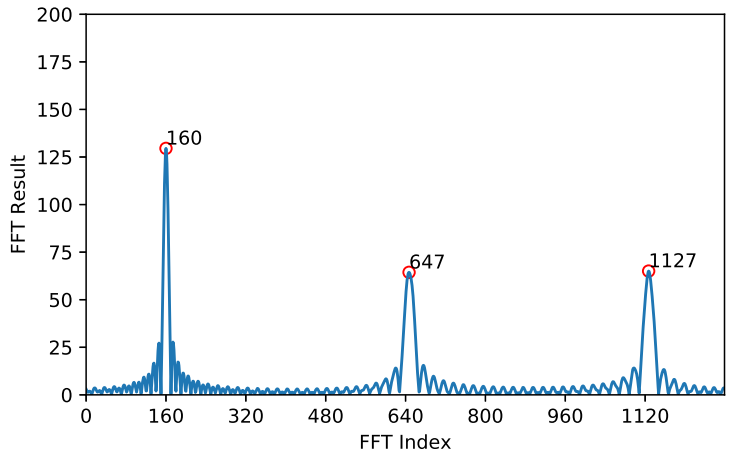


Figure 2.17: FFT results of CHOIR for the first symbol duration of Figure 2.16. The peak 160 represents the symbol 16 of Frame 1. The peaks 647 and 1127 are generated by the symbol 0 and the symbol 48 from Frame 2.

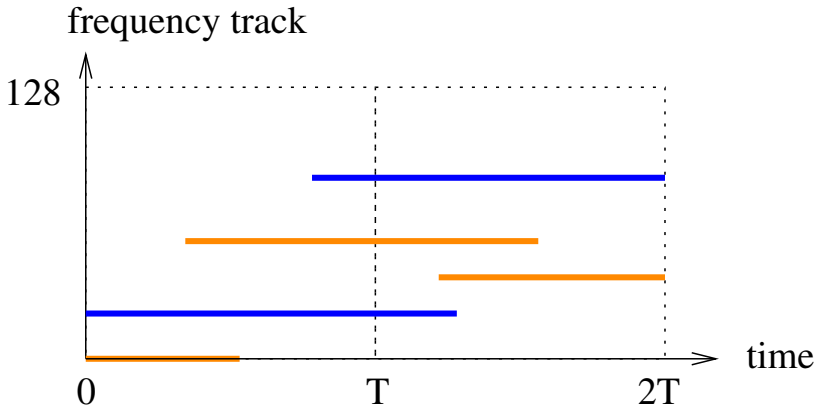


Figure 2.18: Overview of FTrack. The blue lines and the yellow lines are the frequency tracks of the symbols from the two frames in Fig. 2.16. Notice that, depending on the steps and the sizes of the sliding window, the length of tracks can exceed the duration of each symbol. However, all of symbols will result in the tracks with similar duration. Therefore, decoder is still able to tell the symbol edges from the tracks.

OCT

OCT [57] also uses the time offset to match the symbols to the transmitters. The receiver divides times into segments, where each segment is delimited by the symbol edges of colliding LoRa frame. In other words, there cannot be a symbol edge of any frame during a segment. Then, OCT performs an FFT on each segment. It stores all the peaks of each FFT, and determines the sent symbols based on the symbols that are repeated on all the corresponding segments. When the colliding frames are quasi-synchronized, the previous approach cannot be used and OCT instead uses the power level of symbols to match the symbols to the correct frame.

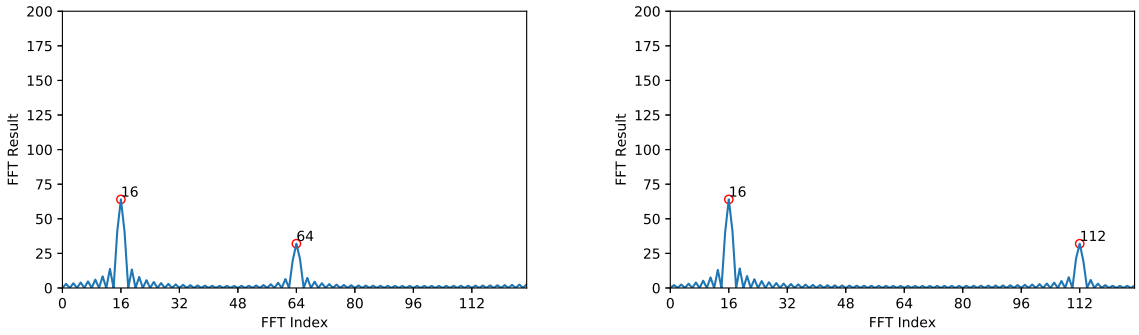


Figure 2.19: FFT results of the first two segments in OCT on the example of Fig. 2.16. Symbol 16 exists in both the two segments. Thus, symbol 16 is considered as a symbol starting at the beginning of the first segment by OCT, which belongs to Frame 1 in Fig. 2.16.

On the example of Figure 2.16, OCT defines the following segments: one is $[0; SD/2]$, and the other is $[SD/2; SD]$. Figure 2.19 shows the FFT results on these two fragments. Note that the first symbol of the blue frame, encoding value 16, presents in both two fragments. Thus, the common value in these two FFT results is the symbol value. In this case, we obtain 16, which is the correct value.

Limits. OCT requires a sufficient time between the symbol edges of colliding frames, in order to obtain segments that are sufficiently long.

CIC

CIC [48] is similar to OCT, and also leverages the time offset to divide each SD into segments, with an FFT being performed on each segment. CIC computes the intersection of the set of symbols of contiguous segments to extract the correct symbol.

Figure 2.20 shows the FFT results of the example of Figure 2.16. The symbol set for the first segment is {16,64}, and the symbol set for the second segment is {48,80}. The symbol values between the two segments

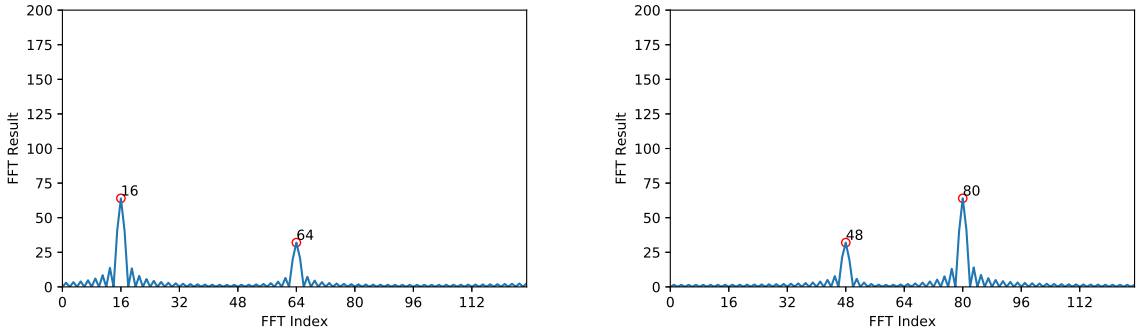


Figure 2.20: FFT results of the first two segments in CIC on the example of Fig. 2.16. By considering the time offsets between the segment, the symbol set {48,80} produces another symbol set {16, 112}, corresponding to the symbol values that should exist in the symbol set of the first segment. By intersecting the symbol set {16,64} and the symbol set {16, 112}, decoder is able to match the symbol 16 to Frame 1.

should be shifted by a symbol value corresponding to half of SD, which is 64. The shift of symbol value results in {16, 112} as the set of the derived symbol values from the second segment. Since the first symbol of Frame 1 is composed of these two segments, by intersecting these two sets, the receiver is able to know that the symbol is 16.

Limits. CIC also requires a sufficient time between the symbol edges of colliding frames.

SCLoRa

SCLoRa [22] leverages both time and power difference to attribute the demodulated symbols to the correct frame. SCLoRa uses sliding windows to identify the symbol edges of each frame from the superposed signal, and to find the best power level of a symbol. The complexity of SCLoRa depends on the number and size of these sliding windows. To our knowledge, SCLoRa is currently the collision resolution protocol with the best performance. Figure 2.21 shows an example of SCLoRa, which leverages the different power levels to classify the symbols to the different frames.

Limits. SCLoRa still fails when the power of the frames are similar, or when the relative time offsets are small.

GS-MAC

GS-MAC has been proposed in [13]. Since we use GS-MAC as a basis in the remainder of this thesis, we describe it in details in the following.

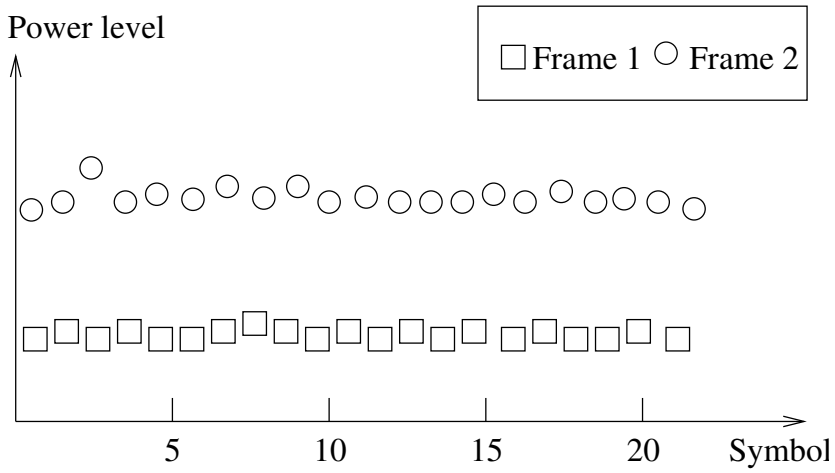


Figure 2.21: Overview of SCLoRa, which classifies the symbols to the different frames using the different power levels.

Time in GS-MAC. GS-MAC is a slotted MAC protocol: time is divided into slots, and each slot is further divided into sub-slots of length SD/M_s, where SD is the symbol duration and M_s is the number of sub-slots per symbol duration. End-devices can only start the transmission of a frame at the beginning of a slot, with a small random backoff between 0 and $x - 1$ sub-slots. Figure 2.22 shows an example of a few frames sent with GS-MAC. Note that the random backoff delays the transmission by less than one symbol compared to the slot starting time.

A receive window is opened at the end of each slot (not explicitly shown here), and has a length equal to the maximum frame duration. The end-devices use this receive window in order to receive acknowledgements, or to synchronize their slots through beacons. The total slot size is equal to twice the maximum time on air of a frame plus one symbol, in order to accommodate for the transmission of a frame (including the backoff) and the reception of the acknowledgement. Note that the gateway acknowledges successfully decoded frames during the same slot as the frame transmission.

When a frame is not successfully decoded by the gateway, the gateway still stores the sender ID and the possible values of each symbol. Upon retransmission of the frame, the gateway uses the possible values of each symbol in order to help decode the retransmission. Thus, a gateway might be able to decode a frame that is always in collision, by reducing the possible values of each symbol at each retransmission.

GS-MAC uses slots to ensure that colliding frames start at a similar time. GS-MAC uses sub-slots to correctly track the symbols of different colliding frames. Typically, GS-MAC is likely to decode the frames that are sent alone in their own sub-slot. If several frames are sent in the same sub-slot, they are all indistinguishable. However, such frames do not impact the decoding of the frames of the other sub-slots.

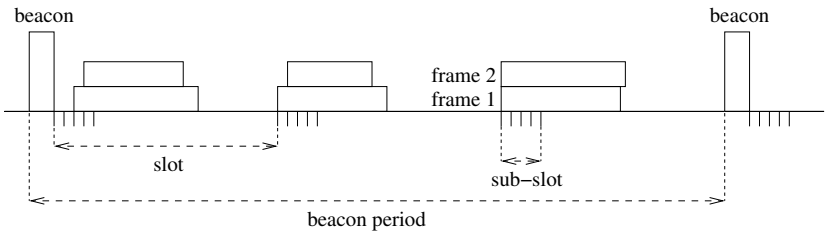


Figure 2.22: Overview of GS-MAC. Three slots are notified by one beacon at the beginning. In each slot, two frames are sent by end-devices at one of the four sub-slots, randomly chosen.

Core algorithm. The core algorithm of GS-MAC depends on the frequency changes in LoRa chirps. From Equation 2.5 and Equation 2.6, we can write the difference between the frequencies of neighboring sub-slots as follows:

$$\begin{aligned}
\Delta f_u^m &= \begin{cases} \frac{M}{SD^2}\frac{i+1}{M_s}SD - BW - \frac{M}{SD^2}\frac{i}{M_s}, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m \\ \frac{M}{SD^2}\frac{i+1}{M_s}SD - \frac{M}{SD^2}\frac{i}{M_s}, \text{ otherwise} \end{cases} \\
&= \begin{cases} \frac{M}{SD^2}\frac{1}{M_s} - BW, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m \\ \frac{M}{SD^2}\frac{1}{M_s}, \text{ otherwise} \end{cases} \\
&= \begin{cases} -\frac{M_s-1}{M_s}BW, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m \\ \frac{1}{M_s}BW, \text{ otherwise} \end{cases}
\end{aligned} \tag{2.21}$$

$$\begin{aligned}
\Delta f_d^m &= \begin{cases} -\frac{M}{SD^2}\frac{i+1}{M_s}SD + BW + \frac{M}{SD^2}\frac{i}{M_s}, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m \\ -\frac{M}{SD^2}\frac{i+1}{M_s}SD + \frac{M}{SD^2}\frac{i}{M_s}, \text{ otherwise} \end{cases} \\
&= \begin{cases} BW - \frac{M}{SD^2}\frac{1}{M_s}, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m \\ -\frac{M}{SD^2}\frac{1}{M_s}, \text{ otherwise} \end{cases} \\
&= \begin{cases} \frac{M_s-1}{M_s}BW, \text{ if } \frac{i+1}{M_s} > \tau_m \text{and} \frac{i}{M_s} < \tau_m, \\ -\frac{1}{M_s}BW, \text{ otherwise.} \end{cases}
\end{aligned} \tag{2.22}$$

Therefore, the frequencies in a given sub-slot are expected to change by $\pm\frac{1}{M_s}BW$ (or $\pm\frac{M_s-1}{M_s}BW$ modulo BW) in the following sub-slot, until the end of the corresponding symbol. Figure 2.23 shows an example where the LoRa symbol duration is divided into $M_s = 4$ sub-slots. The symbol values in each sub-slot of the modulated symbol 48 , for SF7, are 48, 48+32=80, 48+64=112 and 48+96-128=16, respectively.

From the set of possible symbol values for each sub-slot, GS-MAC deduces the actual values in each symbol duration by performing an intersection of the sets of each sub-slot. Indeed, symbol changes can only occur at symbol edges.

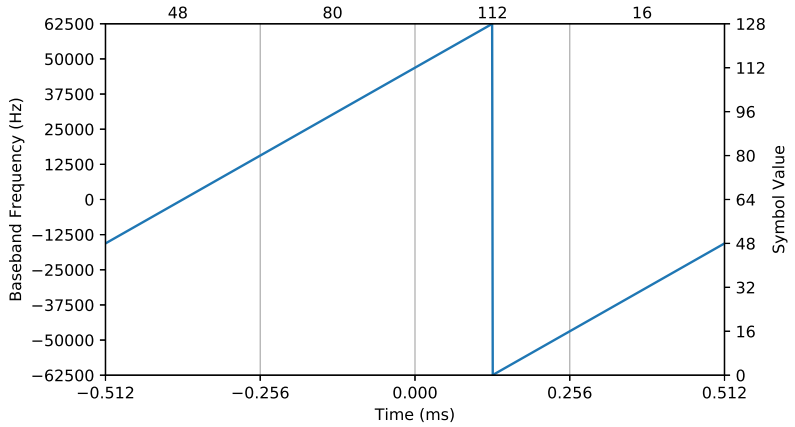


Figure 2.23: An up-chirp encoding value 48 with SF7 and $BW = 125$ kHz, divided into four sub-slots.

Examples of frames sent in different sub-slots of the same slot. When several frames are sent in the same slot, they will generate a collision. Collisions are not necessarily negative, as GS-MAC can resolve cases of collisions in which frames are sent in different sub-slots. Indeed, when frames are sent in different sub-slots of the same slot, the symbol edges are different for each frame. Therefore, an unexpected symbol value at a given sub-slot indicates a symbol change for the frame starting at this sub-slot.

Figure 2.24 shows an example of GS-MAC with SF7 and $M_s = 4$ sub-slots. The figure shows 1.25 symbols of two colliding frames: Frame 1 in blue is sent in sub-slot 1 and is synchronized with the receiver, and Frame 2 in orange is sent in sub-slot 2. Frame 1 consists of symbol 12 (fully shown) and symbol 34 (partially shown), and Frame 2 consists of symbol 48 (partially shown) and symbol 32 (fully shown). From the preamble (not shown here), GS-MAC knows that there is a frame starting in sub-slot 1 and another in sub-slot 2.

- For the first symbol of Frame 1: GS-MAC computes the set of all instantaneous frequencies for the first M_s slots, removes the drift caused by the linear frequency sweep of LoRa symbols, and intersects the sets. The obtained value is $\{12, 16\} \cap \{32-32, 44-32\} \cap \{64-64, 76-64\} \cap \{96-96, 108-96\} = \{12, 16\} \cap \{0, 12\} \cap \{0, 12\} \cap \{0, 12\} = \{12\}$, which is indeed the correct value for the first symbol of Frame 1.

- For the second symbol of Frame 1: The value of the second symbol of Frame 1 cannot be computed as the last three sub-slots are missing, but it can be observed that 34 indeed belongs to the set of frequencies of the first sub-slot of the second symbol, which is $\{0, 34\}$.

- For the first symbol of Frame 2: The value of the first symbol of Frame 2 cannot be computed as the first three sub-slots are missing, but it can be observed that 48 belongs to the set of frequencies of the last sub-slot of the second symbol, which is $\{12-96+128, 16-96+128\} = \{44, 48\}$.

- For the second symbol of Frame 2: The second symbol of Frame 2 is equal to $\{32, 44\} \cap \{64 - 32, 76 - 32\} \cap \{96 - 64, 108 - 64\} \cap \{0 + 128 - 96, 34 + 128 - 96\} = \{32, 44\} \cap \{32, 44\} \cap \{32, 44\} \cap \{32, 66\} = \{32\}$.

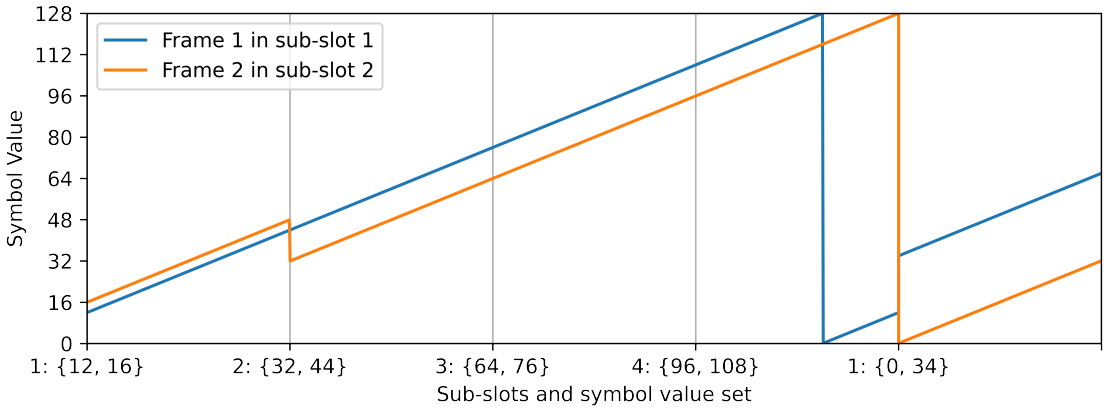


Figure 2.24: Frequency changes of up-chirps from two superposed LoRa frames, transmitted respectively in sub-slot 1 and sub-slot 2, for the GS-MAC protocol.

Examples of frames in the same sub-slot of the same slot. In case of several frames being sent in the same sub-slot of the same slot, the symbol edges are the same for each of those frames. The symbols of these frames are unfortunately indistinguishable by GS-MAC. However, the impact of those frames on the decoding of other frames is limited.

Figure 2.25 shows another example of GS-MAC with SF7 and $M_s = 4$ sub-slots. The figure shows 1.25 symbols of three colliding frames: Frame 1 and Frame 2 are sent in sub-slot 1 and are synchronized with the receiver, and Frame 3 is sent in sub-slot 2. Frame 1 consists of symbol 12 (fully shown) and symbol 34 (partially shown), Frame 2 consists of symbol 0 (fully shown) and symbol 16 (partially shown), and Frame 3 consists of symbol 48 (partially shown) and symbol 24 (fully shown).

- For the first symbol of Frames 1 and 2: By performing the intersections, the obtained value at sub-slot 1 is $\{0, 12, 16\} \cap \{24 + 128 - 32, 32 - 32, 44 - 32\} \cap \{56 + 128 - 64, 64 - 64, 76 - 64\} \cap \{88 + 128 - 96, 96 - 96, 108 - 96\} = \{0, 12, 16\} \cap \{0, 12, 120\} \cap \{0, 12, 120\} \cap \{0, 12, 120\} = \{0, 12\}$, which contains the values for the first symbols of Frame 1 and Frame 2. However, the receiver cannot match the two symbols to the two frames. Thus, we say that the symbols of Frames 1 and 2 have uncertainties, and a retransmission of at least one frame will be required eventually.

- For the second symbol of Frame 3: This symbol can be obtained by computing $\{24, 32, 44\} \cap \{56 - 32, 64 - 32, 76 - 32\} \cap \{88 - 64, 96 - 64, 108 - 64\} \cap \{16 + 128 - 96, 34 + 128 - 96, 120 - 96\} = \{24, 32, 44\} \cap \{24, 32, 44\} \cap \{24, 32, 44\} \cap \{24, 48, 66\} = \{24\}$.

It is interesting to notice that with GS-MAC, although uncertainties remain for Frame 1 and Frame 2, Frame 3 sent in a distinct sub-slot can still be decoded.

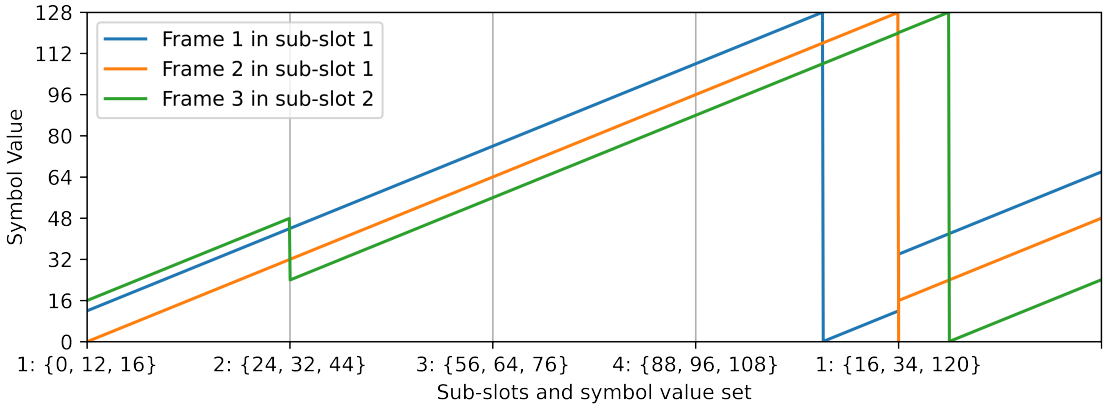


Figure 2.25: Frequency changes of up-chirps from three superposed LoRa frames, two of them being transmitted in sub-slot 1, while the other one is transmitted in sub-slot 2, for the GS-MAC protocol.

Limits. The limits of GS-MAC will be detailed in Section 4.1.

2.3.4 Summary

In [64], we summarized and categorized the existing collision resolution algorithms, as shown in Table 2.3. Note that we introduced a category for protocols based on modulation, with FlipLoRa being the single protocol.

In this thesis, we compare our proposed algorithm of collision resolution mainly to CHOIR [14], CIC [48] and FTrack [59] because the implementation and parameters of these protocols are clearly described. Since CIC and OCT [57] share a similar idea, we chose CIC which is the most recent. We also compare our work with SCLoRa [22], since the environment of the evaluation is easy to reproduce, although there is no existing implementation nor details on its implementation.

Table 2.3: Categorization of several LoRa collision decoding mechanisms according to the main signal feature they use. ◯ stands for primary use, and △ stands for secondary use.

Name	**Reference**	**Power-based**		**Time-based**	**Modulation-based**
		SIC	**Power-difference**		
CHOIR	[14]			◯	
CIC	[48]		△	◯	
CoLoRa	[55]	◯		◯	
FlipLoRa	[65]				◯
FTrack	[59]			◯	
GS-MAC	[13]			◯	
SCLoRa	[22]		△	◯	

Chapter 3

Uncertainties resolution using LoRa coding

In the description of GS-MAC in Subsection 2.3.3, we defined the uncertainties as a set of at least two symbols (including the correct symbol), that are obtained while demodulating a frame. However, uncertainties are not limited to GS-MAC. Although several algorithms have been proposed to decode colliding frames even under specific conditions, there are always conditions in which frames are indistinguishable due to the uncertainties in some symbols.

Therefore, we propose a new collision resolution algorithm based on LoRa coding. This algorithm aims to reduce the uncertainties generated by the collision of several LoRa signals, for any collision resolution algorithm. This algorithm is published in [61].

The remainder of this chapter is organized as follows. In Section 3.1, we first consider a simplified model of LoRa coding in order to introduce the context. In Section 3.2, we present our algorithm to recover frames in the presence of uncertainties. We also show an example of frame recovery based on the simplified model for better and easier understanding. In Section 3.3, we present the limits of our proposed algorithm. Finally, in Section 3.4, we describe in detail our implementation, based on the state-of-the-art on the LoRa coding techniques, as well as the remaining issues we faced with the legacy LoRa coding.

3.1 Context

Let us suppose that two frames collide with each other. Due to the ALOHA random access mechanism, it is very likely that the LoRa signals are desynchronized, which means that one frame is shifted in time, compared to the other one. Let us first consider the case where the time shift for frames is not a multiple

of the symbol duration [1] , as shown in Fig. 3.1. In this case, we can observe that the symbol edges of the symbols in each frames are not aligned. Now, let us consider the case where the time shift for frames is a multiple of the symbol duration, as shown in Fig. 3.2. We can observe that the symbol edges of the symbols in each frames are aligned. Note that the case where frames are completely synchronized from the beginning of the frames is similar to the second case.

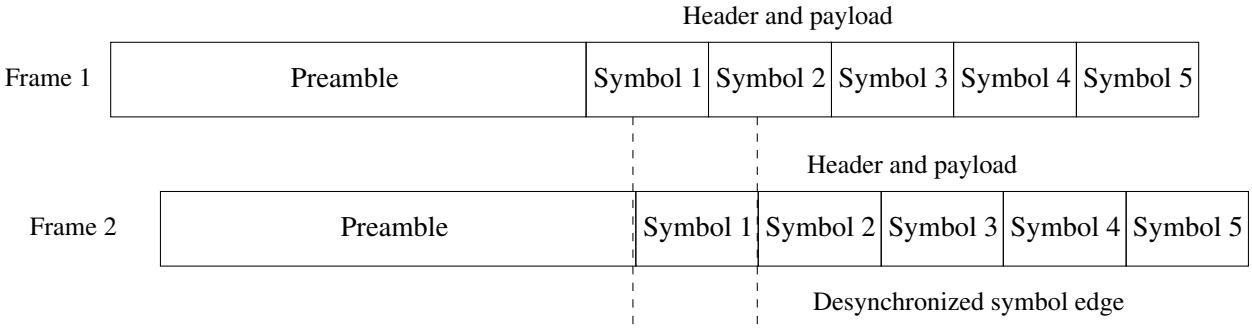


Figure 3.1: Case 1 (desynchronized symbols): Example of two desynchronized frames in collision. Frame 2 arrives later than Frame 1, with a delay of about $SD/3$.

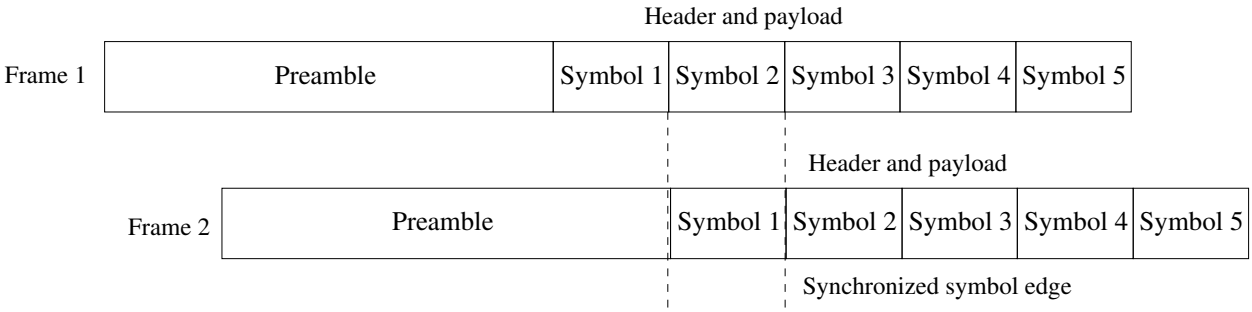


Figure 3.2: Case 2 (synchronized symbols): Example of two desynchronized frames in collision, with overlapping symbol edges. Frame 2 arrives later than Frame 1, with a time offset equal to SD.

In the scenario where two frames (f_1 and f_2) are sent by two transmitters and are desynchronized by $SD/2$, the first values of f_1 and f_2 are shown in Table 3.1. The symbol values of the second frame observed by the receiver are shifted by $\frac{2^{SF}}{2}$. During each symbol of f_1, the receiver will observe a single value for f_1 and two values for f_2. For instance, during symbol 1 of f_1, the receiver will detect 50 (which is the correct value for f_1), 16, and 31. During symbol 2 of f_1, the receiver will detect 111 (which is the correct value for f_1), 31, and 24. This helps the receiver determine which symbol value belongs to which frame.

Let us now consider the case where the frames are completely synchronized (that is, without a time

[1]For simplification, we ignore here that the preamble length is 12.25 symbols. Without this simplification, the condition of non-overlapping symbol edges depends on the position of the symbol of each frame, in the preamble or in the payload.

Table 3.1: Example of the payload symbols of two colliding frames, f_1 and f_2, desynchronized by $SD/2$, and encoded with SF7.

f_1	50	111	5	95	79
f_2	16 (= 80 - $\frac{2^{SF}}{2}$)	31 (= 95 - $\frac{2^{SF}}{2}$)	24 (= 88 - $\frac{2^{SF}}{2}$)	93 (= 29 + 2^{SF} - $\frac{2^{SF}}{2}$)	40 (= 104 - $\frac{2^{SF}}{2}$)

offset), as shown in Table 3.2. The receiver, using a collision resolution algorithm, is able to obtain the set {50, 80} for the first symbol, the set {111, 95} for the second symbol, and so on. Note that this case might produce uncertainties in the entire frame, if the receiver is not able to match the symbol of each set to the correct frame.

Table 3.2: Example of the payload symbols of two completely synchronized frames, f_1 and f_2, encoded with SF7.

f_1	50	111	5	95	79	26	33	94	124	18	52	27	58	116	94	1	0	32	16	0
f_2	80	95	88	29	104	67	31	81	45	91	98	9	19	10	99	1	64	32	16	8

In both cases, depending on the algorithm used and its features, the receiver might be able to match the different symbol values to the corresponding frames, *i.e.*, the receiver might be able to distinguish the frames. For example, SIC is able to distinguish the frames by ignoring the symbols from the weaker frame (which will typically happen in the first case shown in Table 3.1, because of the capture effect based on the desynchronization). CHOIR can conditionally resolve the collision using the tiny frequency offset hidden in the symbols of f_1 and f_2 (typically in the both cases, if the two transmitters have the different tiny frequency offsets).

However, some frames remain indistinguishable when they have similar features. For instance, in SIC, this happens when the frames are received with a similar power and with synchronized symbols. In CHOIR, this happens when the two transmitters produce a similar tiny frequency offset. In GS-MAC, this happens when the frames are sent in the same slot and sub-slot. The synchronization of the frames is not the only factor that brings uncertainties. Indeed, time-based algorithms cannot correctly match symbols to collided frames when their symbol edges are nearly synchronized, even if the start of the frames is not fully synchronized. Power-based algorithms are not able to capture a frame when several frames are received with a similar power. Burst noise or short interference could also bring uncertainties to a small number of symbols.

3.2 Our mechanism based on LoRa coding

Our algorithm manipulates the uncertainties based on groups of symbols generated by the LoRa coding from groups of codewords. Specifically, recall that during the LoRa coding, the interleaver distributes the data bits and the redundancy bits of codewords into a group of several symbols, called a symbol block, as shown in Figure 2.11. By leveraging this property, the receiver can significantly reduce the number of uncertainties when decoding.

3.2.1 Symbol blocks

After LoRa coding, the total number of symbols in the header and the payload of a complete LoRa frame is defined as follows [42]:

$$N = 8 + \max\left(\left\lceil \frac{2PL - (SF-2) + 4CRC + 5H}{SF - 2LD}(CR+4)\right\rceil, 0\right), \tag{3.1}$$

where PL is the number of bytes encoded in the frame, SF is the spreading factor, CRC denotes the presence of payload CRC in the frame (and is equal to 1 if CRC is enabled, and to 0 if CRC is disabled), H indicates whether the frame contains a header (H=1) or not (H=0), LD indicates the low data rate mode (1 for enabled, 0 for disabled), and CR (with values from 1 to 4) is the parameter controlling the amount of redundancy in the frame. Let us explain this formula, as it provides some implicit information about the structure of the header and payload in a LoRa frame.

- Value $2.PL$: The CR used by LoRa always works based on nybbles. Each nybble contains four bits to be encoded (which is half of a byte with eight bits). Thus, the number of nybbles is two times the number of payload data in bytes, PL, which results in $2.PL$ in Equation 3.1.
- Value $5H$: The information of the frame in the header, if exists, consists of five nybbles, which results in $5H$ in Equation 3.1. Note that in the header, the first two nybbles encode the length of the payload in bytes. The third nybble indicates the CR used in the current LoRa frame. The remaining two nybbles encode the CRC of the header.
- Value 8: The initial value eight indicates that there are always at least eight symbols in a LoRa frame. These symbols are mainly used to encode the information of the frame in the header with CR4 and low data rate mode. However, it can also encodes a few payload data with the remaining spaces.
- Value $4CRC$: This value indicates that when the CRC of the payload is present, it occupies two bytes, which is four nybbles.
- Numerator: The entire numerator indicates the number of nybbles that need to be encoded except the nybbles that can be already encoded by the initial eight symbols.

- Value $SF - 2LD$: The number of bits encoded into a symbol is usually SF, but when the low data rate mode is enabled (that is, when $LD = 1$), this value is limited to $SF - 2$.
- Value $CR + 4$: This value represents the amount of redundancy introduced by the parameter CR, for every four bits of data.
- Use of a maximum: The numerator can be negative if the first eight symbols can encode all of the header and the payload data. In this case, the number of symbol blocks becomes zero, except the header symbol block. When SF is large, the initial eight symbols yield some extra space in addition to the nybbles encoding the information of the frame in header (more precisely, the extra space has $(SF-2)-5$ nybbles). This extra space allows a small amount of payload data to be encoded into these eight symbols, which brings $(SF - 2) + 5H$ in Equation 3.1. For SF8, the extra space is 1 nybble, and for SF12, the extra space is 5 nybbles. The implicit header mode, used when $H = 0$, also enables the same behavior even with low SF. When $H = 0$, all of the first eight symbols are used for the payload data encoding, which enables 5 nybbles for SF7 and 10 nybbles for SF12.

Figure 3.3 shows the overview of the structure of the LoRa frame. LoRa coding takes the original data as input, divides it into nybbles, and encodes the nybbles with optional extra information (if $H = 1$) into several symbol blocks. When there is extra space for more nybbles in the header, some nybbles from the payload are also encoded into the header block.

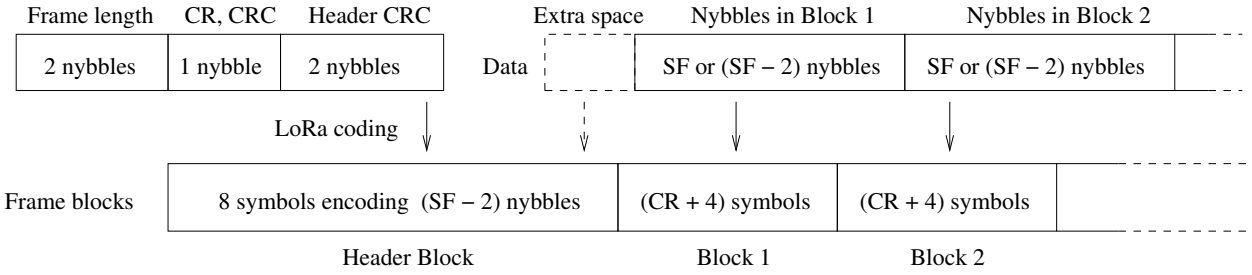


Figure 3.3: Overview of the structure of LoRa frame before and after LoRa coding.

Equation 3.1 can be rewritten as follows:

$$N = 8 + (CR + 4)K, K \geq 0, \tag{3.2}$$

where K is the number of symbol blocks containing $(CR + 4)$ symbols in each. The number K is a function of parameter settings such as CR and SF. With this representation, we can thus consider a LoRa frame as a concatenation of one symbol block as header, and K symbol blocks as payload. The header symbol block consists of eight symbols, encoding the payload length, the CR and the presence of CRC, all encoded with a fixed CR of $CR = 4$. Each symbol block in the payload consists of $CR + 4$ symbols (that is, encoded with the CR set in the header).

We can apply this to the case of two synchronized frames, shown in Table 3.2, without a header block but with four payload blocks. The resulting uncertainties can be represented by four symbol blocks with $CR = 1$ and SF7, as shown in Table 3.3.

Table 3.3: Example of the uncertainties from the payload of two superposed frames, grouped by blocks.

Block 1				
Symbol 0	Symbol 1	Symbol 2	Symbol 3	Symbol 4
{50, 80}	{95, 111}	{5, 88}	{29, 95}	{79, 104}
Block 2				
Symbol 5	Symbol 6	Symbol 7	Symbol 8	Symbol 9
{26, 67}	{31, 33}	{81, 94}	{45, 124}	{18, 91}
Block 3				
Symbol 10	Symbol 11	Symbol 12	Symbol 13	Symbol 14
{52, 98}	{9, 27}	{19, 58}	{10, 116}	{94, 99}
Block 4				
Symbol 15	Symbol 16	Symbol 17	Symbol 18	Symbol 19
{1}	{0, 64}	{32}	{16}	{0, 8}

3.2.2 Core algorithm

Our algorithm based on LoRa coding uses two steps. First, all possible codewords are generated and validated using the redundant bits controlled by CR. Second, all possible frames are generated and validated using the CRC.

The first step of our algorithm focuses on each symbol block independently. If there are any uncertainties in a symbol block, which are interpreted as several possible symbol values, the decoder generates all the potential blocks and attempts to deinterleave them. It then computes the expected redundant bits *compECC* from the data bits, and compares them with the redundant bits from the received LoRa frame, referred to as $refECC$. The encoded redundant bits $refECC$ should match *compECC* for all codewords in a potential block, as a candidate. If the bits differ, this candidate symbol block is discarded. The detailed algorithm is given in Algorithm 1.

The second step of our algorithm takes as input the validated blocks. Note that each block takes the position in the LoRa frame to which it belongs. This position does not change during interleaving or deinterleaving. Thus, after the validation of codewords, the data encoded in the codewords at each position can be retrieved and assembled to get the possible data for the upper layer, which are MAC frames. The

Algorithm 1: Validation of codewords (step 1) in our LoRa coding based mechanism.

```
S ← all possible symbol blocks generated from the uncertainties
validatedBlocks ← {}
foreach block b in S do
    cb ← deinterleave b
    cb_validated ← true
    foreach codeword c in cb do
        d ← extract data bits from c
        refECC ← extract ECC from d
        compECC ← compute Hamming coding from d
        if refECC ≠ compECC then
            cb_validated ← false
    if cb_validated then
        add cb into validatedBlocks
```

CRC of the possible MAC frames, encoded into the last two bytes of the MAC frame, can be computed from the data in the MAC frames. By leveraging the CRC, we can further reduce the amount of possible MAC frames by comparing the CRC calculated from the data with the CRC retrieved from the MAC frames. Algorithm 2 shows how the second step of our algorithm works.

Algorithm 2: Validation of the CRC (step 2) in our LoRa coding based mechanism.

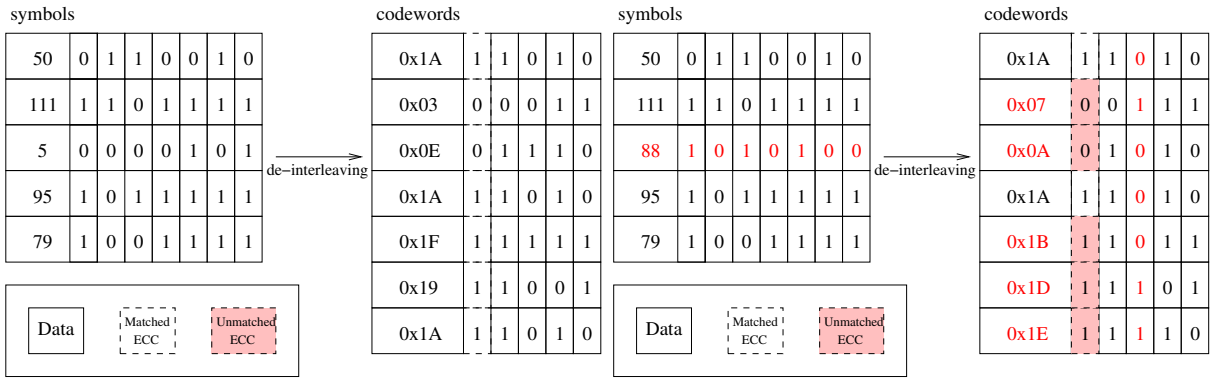
```
foreach possible frame f in decoded MAC frames do
    truncate f according to the frame length
    refCRC ← retrieve CRC from f
    compCRC ← compute CRC from f
    if refCRC = compCRC then
        f is a possible MAC frame
        send f to the network server
    else
        drop f
```

Time complexity. Thanks to the symbols being grouped into blocks, the number of possibilities in the first step is limited for each block. Instead of combining all the symbols in the entire LoRa frame, the decoder can only combine all the symbols in each block, which limits the number of possibilities to ${N_u}^{CR+4}$, where N_u is the maximum number of possibilities in a symbol block. Therefore, the time complexity of Algorithm 1 is $\mathbf{O}(K \times {N_u}^{CR+4})$. The time complexity of Algorithm 2 is limited by the maximum number of validated codeword blocks N_{cwb}, giving a complexity of $\mathbf{O}({N_{cwb}}^{K})$. If all of the symbol blocks pass the validations

in Algorithm 1, the worst time complexity of Algorithm 2 can reach $\mathbf{O}({N_u}^{(CR+4)K})$, which is equivalent to traversing all the possible combinations of uncertainties. However, this is highly unlikely to occur due to the existence of ECCs.

Example. Let us demonstrate how Algorithms 1 and 2 reduce uncertainties in the following example.

For the first step, let us consider the two frames in Table 3.3, encoded with SF7 and $CR = 1$. There are generally two uncertainties for each symbol of the frames. The gateway knows that each group of $CR + 4$ symbols composes a symbol block. Ze consider the first five uncertainty sets from Block 1 of Frame 1 and Frame 2: {50, 80}, {111, 95}, {5, 88}, {95, 29}, and {79, 104}. By combining these uncertainties, we obtain 32 possible blocks for each frame. Figure 3.4 shows two of these 32 blocks: one with symbols (50,111,5,95,79), and another with symbols (50,111,88,95,79). Note that the only difference between these two candidate blocks is the third symbol (with a value of either 5 or 88, as expected from the uncertainty set). After deinterleaving, we observe that the third column (the only different column) of codewords is 0010100 for the first block, and is 0100011 for the second block. We can see that, for the first block containing ECC bits, the first column of codewords matches the expected ECC, and this block is validated. However, for the second block, the first column of the codewords does not match the expected ECC: this second block is thus incorrect and invalid. In other words, the symbol block {50, 111, 88, 95, 79} is considered as a valid possible symbol block, while the symbol block {50, 111, 88, 95, 79} is discarded. Through Algorithm 1, we reduce the number of possible blocks for Block 1 from 32 to 4, as shown in Table 3.4.



(a) Successful validation on ECCs.

(b) Failed validation on ECCs.

Figure 3.4: Example of the diagonal interleaver applied on a valid codeword block (left) and on an invalid codeword block (right) with SF7 and $CR = 1$. The wrong choice of one single symbol caused five unmatched ECCs, which fails the validation.

For the second step, at a given position in the frame, each possibility contains SF codewords. These codewords are deinterleaved from the corresponding symbol blocks. In other words, as shown on Table 3.4,

Table 3.4: Example of the blocks validated by Algorithm 1, on the example of Table 3.3.

Validated codewords from Block 1	Validated codewords from Block 2
1. {0x1A 0x06 0x1A 0x1A 0x0B 0x0E 0x0D}	1. {0x19 0x1F 0x0E 0x12 0x12 0x1A 0x0D}
2. {0x1A 0x06 0x02 0x1A 0x03 0x0E 0x0D}	2. {0x11 0x1F 0x06 0x1A 0x12 0x1A 0x0D}
3. {0x1A 0x03 0x0E 0x1A 0x1F 0x19 0x1A}	3. {0x08 0x1F 0x08 0x05 0x05 0x1C 0x0E}
4. {0x1A 0x03 0x06 0x1A 0x17 0x19 0x1A}	4. {0x00 0x1F 0x00 0x0D 0x05 0x1C 0x0E}
Validated codewords from Block 3	**Validated codewords from Block 4**
1. {0x14 0x12 0x03 0x14 0x0B 0x17 0x1C}	1. {0x0D 0x00 0x00 0x00 0x00 0x00 0x00}
2. {0x1C 0x1A 0x0B 0x14 0x03 0x1F 0x14}	2. {0x1F 0x00 0x00 0x00 0x00 0x00 0x00}
3. {0x00 0x03 0x14 0x14 0x1A 0x11 0x0D}	
4. {0x08 0x0B 0x1C 0x14 0x12 0x19 0x05}	

from the 4 possibilities for Block 1, the 4 possibilities for Block 2, the 4 possibilities for Block 3, and the 2 possibilities for Block 4, we obtain $4 \times 4 \times 4 \times 2 = 128$ possible MAC frames. For example, if we take the first possibility for each block, the following frame can be reconstructed: (0xA6 0xAA 0xBE 0xD9 0xFE 0x22 0xAD 0x42 0x34), and the following CRC (0xB7 0xCD). Algorithm 2 computes the CRC from the data of this frame and compares it with the CRC value. In this case, the CRC does not match, so the frame is incorrect. Table 3.5 shows the three only frames that are decoded from the uncertainties with the help of Algorithm 2, from the 128 frames resulting from Algorithm 1. Among these three frames, Frame 1 and Frame 2 are the correct frames transmitted by the end-devices, as shown on Table 3.2. Frame 3 is an incorrect frame: it passes the CRC test and was decoded, although it was not sent by any device. In practice, such incorrect frames are rare when the number of uncertainties is small, as discussed later.

3.3 Limits

Computation time. The main limitation of our algorithm is the computation time required at the gateway. If many end-devices simultaneously send frames, the computational overhead might be significant for the gateway, delaying the overall decoding, reception, and acknowledgment. For example, for N indistinguishable frames, $CR + 4$ symbols can generate up to $N^{(CR+4)}$ possible blocks, which could be a large number. To overcome this issue, the gateway can set one threshold per step and wait for retransmissions if the number of indistinguishable frames exceeds either thresholds.

Incorrect frames. Our proposed algorithm might also recover incorrect frames that have not been actually transmitted. Such an example is shown for Frame 3 in Table 3.5. This occurs when a frame (here, Frame 3)

Table 3.5: Remaining frames that are decoded by our LoRa coding mechanisms. The first two frames were the correct frames in collision, while the last frame was not sent by any end-device.

Validated frames		
Frame 1	Symbol values	50 111 5 95 79 26 33 94 124 18 52 27 58 116 94 1 0 32 16 0
	Decoded frame	0xA3 0xEA 0xF9 0xA8 0xF8 0x55 0xCE 0xCA 0xB4
	CRC	0x3F 0x4D
Frame 2	Symbol values	80 95 88 29 104 67 31 81 45 91 98 9 19 10 99 1 64 32 16 8
	Decoded frame	0xA3 0x6A 0x79 0xA8 0xF8 0x55 0xCE 0x8B 0xC4
	CRC	0x29 0x5F
Frame 3	Symbol values	50 111 5 29 79 26 33 94 124 18 98 9 19 116 99 1 64 32 16 8
	Decoded frame	0xA6 0x2A 0x3E 0xD1 0xF6 0xA2 0xAD 0x03 0x44
	CRC	0xA1 0xDF

is completely composed of symbols from the other frames (here, Frame 1 and Frame 2), and has a valid CRC. Designing an algorithm that would recognize Frame 3 as being constructed from Frame 1 and Frame 2 is possible, but that would not be sufficient to ensure that Frame 3 is incorrect, as such a frame might have been actually sent. Such errors are theoretically impossible to avoid [12] without making assumptions on the upper layers. However, they can be detected and handled at a higher layer by the NS because the data is likely to be corrupted, which is out of the scope of this thesis. The NS is not likely to reply to incorrect frames, due to data corruption. Moreover, acknowledging incorrect frames at the gateway is not an issue, as either the end-device would not exist, or the end-device would not have opened the receive window for an acknowledgment (since it has not sent the frame in the first place).

Table 3.6 evaluates the percentage of incorrect frames with different CRCs and different numbers of colliding frames, based on Monte-Carlo simulations, with SF8 and 2000 repetitions. As CRC increases, there are fewer incorrect frames, because the ECC matching in Algorithm 1 can leverage more bits to reduce the uncertainties. Overall, incorrect frames occur very rarely when there are not many colliding frames.

Table 3.6: Percentage of incorrect frames.

Colliding frames	2	4	6	8
Encoding with CR1, SF8	0.05%	2.82%	37.50%	91.12%
Encoding with CR2, SF8	0.09%	1.67%	16.11%	55.21%
Encoding with CR3, SF8	0.05%	0.07%	0.22%	0.88%
Encoding with CR4, SF8	0.05%	0.05%	0.20%	0.63%

3.4 Details on our implementation

While implementing our LoRa coding and attempting to decode real LoRa frames, we came across a few issues that are discussed here. In this section, we thus describe our implementation of the complete coding scheme for LoRa, which encodes the data from the upper layer (*e.g.*, LoRaWAN MAC layer) into symbol values. With this implementation, we are able to decode the frames encoded by our own implementation, and also from commodity nodes.

Note that the literature on the actual implementation of LoRa coding is rather scarce. To the best of our knowledge, [25] is the first research work that explored the LoRa coding through reverse engineering. Authors in [38] concentrate on the demodulation and decoding of the LoRa frames, without considering the error detection or the error correction brought by the Hamming coding. However, they pointed out some wrong assumptions in the implementation of [25], and notably the incorrect whitening sequence. Authors in [30] describe their own understanding on the channel coding in LoRa. However, they ignored the Gray coding for the symbol values. Authors in [53] also describes their own implementation of LoRa coding in GNU Radio.

We primarily referred to [38] to implement our LoRa coding, and used [53] as an assistance.

Nybble swapping

The data in bytes is first divided into nybbles of four bits. The nybbles are then allocated into several blocks. The first block is the header block with eight symbols, which can encode $SF - 2$ nybbles in total. The potential extra space is used to encode the first several nybbles of the payload. The following blocks are just for the remaining nybbles of the payload, and they encode either SF nybbles each when the low data rate mode is disabled, or $SF - 2$ nybbles each in low data rate mode.

The nybbles in the payload are generated from the original data in bytes, after shuffling. Figure 3.5 shows the allocation of nybbles into blocks for a frame encoded with SF7 and CR1, in the explicit header mode, without the low data rate mode. Here the header block contains only the header information. Note that the nybbles from the payload bytes are swapped. Figure 3.6 shows another example of the allocation of nybbles into blocks for a frame encoded with SF7 and CR1, in the explicit header mode, again without the low data rate mode. In this case, the header block does not contain the header information. The header block thus encodes the first two bytes, as well as the lower four bits of the third byte. Then, Block 1 encodes the higher four bits of the third byte, along with the fourth byte, the fifth byte and part of the sixth byte. During the decoding, the decoder also needs to swap the nybbles, in order to retrieve the original data in bytes.

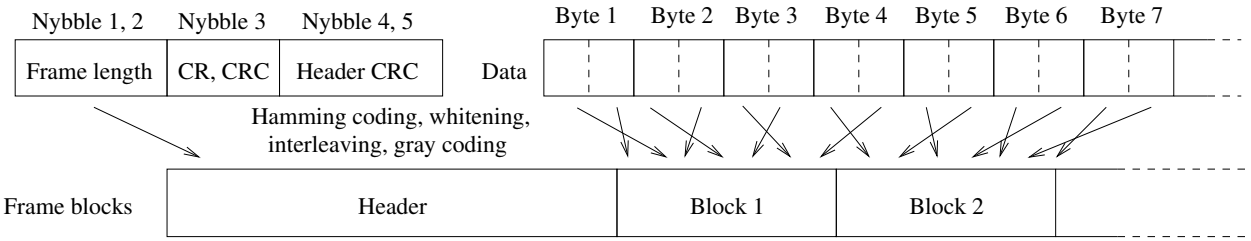


Figure 3.5: Allocations of nybbles in LoRa coding with SF7 and $CR = 1$, in the explicit header mode. Note that the nybbles are swapped.

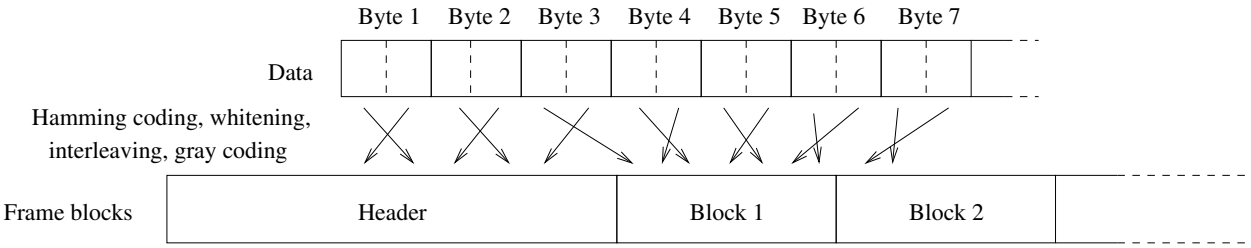


Figure 3.6: Allocations of nybbles in LoRa coding with SF7 and $CR = 1$, in the implicit header mode.

Hamming coding

After the swapping, all the nybbles (including the header information) are encoded using Hamming coding.

Let us consider the generic case for CR2, CR3 and CR4. The encoder takes each nybble with four bits as an input. The Hamming coding $(CR + 4, 4)$ is used to generate $CR + 4$ bits, producing $CR + 4$ bits for each nybble. Table 3.7 shows the mapping between the nybbles and the results of Hamming coding. The highlighted bits from left to right in Table 3.7 are the four bits from the original nybbles, from the highest one to the lowest one. Table 3.7 also shows the available bits in CR2, CR3 and CR4, which contain six, seven and eight bits, respectively.

The case of CR1 is different, as the encoder computes a parity bit for each nybble. The extra bit and the nybble itself are combined to generate five bits. To make this case consistent with the case for the other CRCs, the bits are reorganized as follows: the data bits from the highest bit to the lowest bit are placed into Bit 5, Bit 3, Bit 2 and Bit 1, respectively, while the parity bit is placed in Bit 4. We then pad the produced bits to an entire byte with zero (note that this is also used for CR2 and CR3).

The output of the encoding with any CRC becomes a series of bytes, with a padding of zeros for CR1, CR2 and CR3. The padding does not impact the encoding, because it is skipped during the interleaving process. During the decoding, the decoder can directly retrieve the corresponding bits to compose the nybbles, as done in [38]. Otherwise, the decoder can recover the data by correcting the errors in case of $CR \geq 3$, which is not implemented in [38], but implemented in [30] and [17].

Table 3.7: Hamming coding (8,4) used in our implementation of LoRa coding. The gray columns, from left to right, contain the Bit 3 (the highest bit in the nybble), Bit 2, Bit 1, Bit 0 of the nybble in order. The three bottom lines show the actual bits used by CR2, CR3 and CR4.

Nybble	Hamming coding	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0	0x00	0	0	0	0	0	0	0	0
0x1	0xd2	1	1	0	1	0	0	1	0
0x2	0x55	0	1	0	1	0	1	0	1
0x3	0x87	1	0	0	0	0	1	1	1
0x4	0x99	1	0	0	1	1	0	0	1
0x5	0x4b	0	1	0	0	1	0	1	1
0x6	0xcc	1	1	0	0	1	1	0	0
0x7	0x1e	0	0	0	1	1	1	1	0
0x8	0xe1	1	1	1	0	0	0	0	1
0x9	0x33	0	0	1	1	0	0	1	1
0xA	0xb4	1	0	1	1	0	1	0	0
0xB	0x66	0	1	1	0	0	1	1	0
0xC	0x78	0	1	1	1	1	0	0	0
0xD	0xaa	1	0	1	0	1	0	1	0
0xE	0x2d	0	0	1	0	1	1	0	1
0xF	0xff	1	1	1	1	1	1	1	1
Composition of bits with CR2		\	\	Available bits with CR2					
Composition of bits with CR3		\	Available bits with CR3						
Composition of bits with CR4		Available bits with CR4							

Whitening

The series of bytes are then whitened using a whitening sequence. There is one whitening sequence for CR1 and CR2, and another for CR3 and CR4. We directly use the sequences from [38]. The bytes derived from the nybbles encoding the frame information are not whitened (which is the same as using a whitening sequence with only 0s). However, our encoder whitens the bytes derived from the nybbles for the header block, using the whitening sequence for CR3 and CR4, because the header is encoded with CR4. The remaining bytes of the payload are whitened using the corresponding whitening sequence, according to the given CRC. Such implementation is different from the implementation in [38]. During the decoding, the decoder needs to de-whiten the bytes using the same whitening sequence, in order to obtain the original bits from the corresponding bytes.

Shuffle and interleaving

After whitening, the encoder takes every SF (or $SF-2$ in low data rate mode) codewords with $CR+4$ bits in each block, as the input of interleaving. Inverleaving generates the symbol blocks with $CR+4$ symbols in each. However, after whitening, we have eight bits in each byte. In order to obtain the codewords, the encoder needs to retrieve the corresponding $CR+4$ bits from the series of bytes which is whitened, in order to encode the nybbles. Authors in [38] refer to this operation as a shuffle manipulation. The shuffle manipulation moves the bits to various positions in the codewords. Figure 3.7 shows the shuffle manipulation, which is unique for all CRCs. Bit 5 in the encoded byte comes from Bit 3 (the highest bit) in the original nybble, and is moved to Bit 3 in the codeword. Bit 3 (originally Bit 2) is moved to Bit 2. Bit 2 (originally Bit 1) is moved to Bit 1. Finally, Bit 1 (originally Bit 0) is moved to Bit 0. Thus, the four bits from the original nybbles are moved to their original positions. The redundancy bits added during the encoding are then moved to Bit 7, Bit 6, Bit 5 and Bit 4, respectively. According to the CRC, the encoder then trims the bits to obtain the codewords with $CR+4$ bits. After shuffling, we get the codewords from the bytes. Then, the encoder performs interleaving for each block, as described in Subsection 2.1.3 and shown in Figure 2.11, which results into symbol blocks. For the blocks containing an insufficient number of codewords, we pad them with zeros before interleaving.

During the decoding, the decoder performs de-interleaving. During the de-shuffling operation, we also pad the non-existing bits with 0s in the codewords. This padding does not impact the decoding, because the corresponding bits are ignored during the decoding.

Gray decoding

Finally, the symbols in the blocks are Gray decoded, as described in Subsection 2.1.3. The modulator then modulates the symbol in order to transmit it.

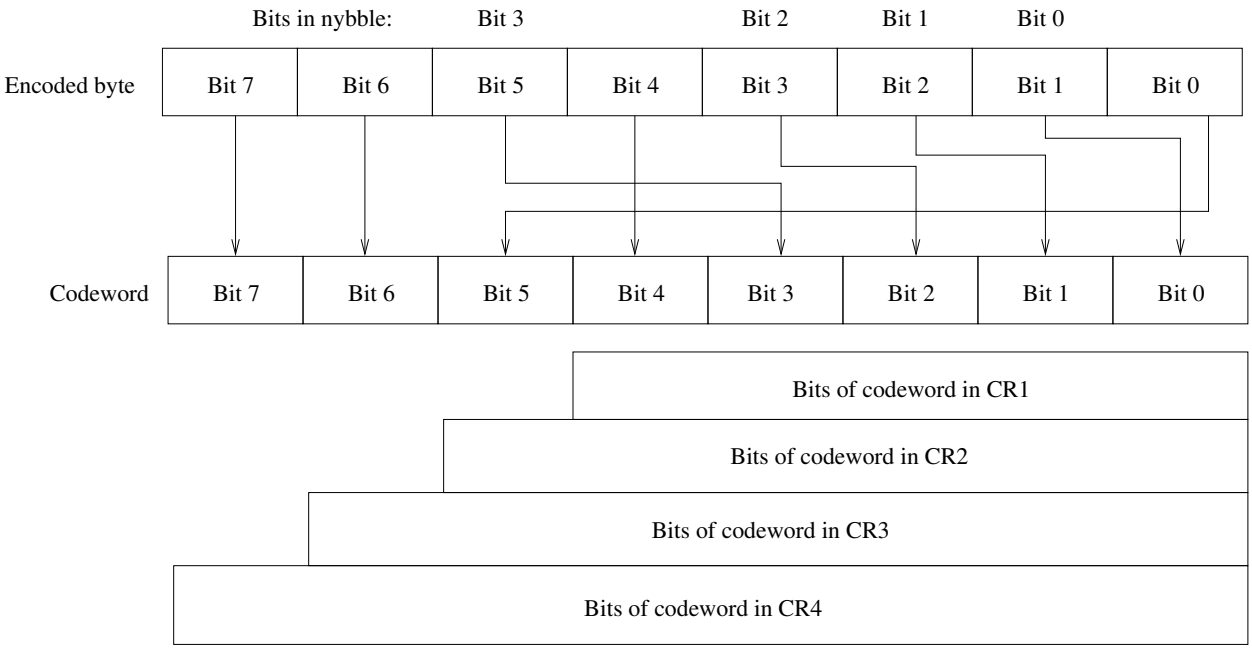


Figure 3.7: Shuffle scheme for all CRCs. The four bits from the nybbles (Bit 5, Bit 3, Bit 2 and Bit 1, which are originally Bit 3, Bit 2, Bit 1 and Bit 0, respectively) are moved to their original positions (Bit 3, Bit 2, Bit 1 and Bit 0) in the nybbles. The redundancy bits are then moved to Bit 7, Bit 6, Bit 5 and Bit 4.

At the reception, the symbols are Gray encoded, in order to reduce the amount of bit errors when there is a small symbol error.

Remaining issues with the legacy LoRa coding

Although we implemented the LoRa coding based on the state-of-the-art, there are still some points that are not clearly described in the literature, which caused us issues in the implementation. For example:

- In the header block, we are not sure whether the whitening sequence used to whiten the encoded bytes from the payload should always be the whitening sequence for CR3 and CR4 (because the header block is composed of nybbles encoded using CR4), or should be the whitening sequence used for the payload (especially for CR1 and CR2).
- We do not know how the padding is done before interleaving, and how to pad the last block.
- We are not sure about how to manipulate the parity bit generated in case of CR1.

We are able to decode the frames from the signals generated using our implementation of LoRa coding. Our implementation is also able to decode the frames from the commodity node, as a end-device in LoRa. We do not consider the redundancies during decoding, as described in [38]. It is important to note that only

CR3 and CR4 can aid in decoding, which is not what we evaluated in our results. Nevertheless, we will present some examples of differences between frames generated using our implementation of LoRa coding and those generated by a commodity node, *i.e.*, an end-device in LoRa, in Section 5.5.

Chapter 4

Slot-Free Decoding Scheme

This chapter describes the core contribution of this thesis, called SF-DS [64]. SF-DS is a novel decoding scheme proposed to improve the performance of LoRaWAN networks in case of collisions.

The SF-DS algorithm is inspired by the algorithm of GS-MAC [13]. However, SF-DS removes the main limitations of GS-MAC, which are: (1) the need for an oracle to determine instantaneous frequencies, and (2) the slot mechanism. By being slot-free, SF-DS is compatible with legacy LoRaWAN devices, unlike GS-MAC. Additionally, SF-DS also uses specificities from the LoRa coding scheme in order to resolve additional uncertainties [61], as explained in Chapter 3.

In this chapter, we first describe how SF-DS removes the main limitations of GS-MAC, including the need for the oracle and the use of slots. Next, we provide a detailed description of the preamble detection mechanism used by SF-DS, which is crucial for implementing SF-DS on real hardware, and can be challenging to achieve in the presence of collisions. Finally, we analyze the impact of channel imperfections on the SF-DS algorithm, including frequency offsets, time offsets and various noise levels.

4.1 Removing the main limitations of GS-MAC

In this section, we first recall the two main assumptions made by the GS-MAC protocol. Then, we explain how SF-DS addresses each of these assumptions.

4.1.1 Zoom on the assumptions of GS-MAC

GS-MAC relies on two main assumptions: it assumes that an oracle is capable of detecting frequencies instantaneously (rather than during one full symbol duration), and it assumes that time is slotted (which makes it incompatible with legacy LoRaWAN end-devices). Additionally, GS-MAC also has a few more weaknesses: it is unable to resolve collisions of several frames transmitted during the same sub-slot, it

requires a complex MAC protocol to accurately synchronize end-devices, and colliding frames that have been resolved cannot be acknowledged easily.

Oracle for instantaneous frequency detection

In GS-MAC, an oracle detects normalized frequencies at the frontier of each sub-slot. The frequencies from the M_s sub-slots of a given symbol are then intersected in order to remove residual frequencies from symbols of frames using different sub-slots. If this intersection returns a set with a single frequency, the frequency corresponds to the symbol value in the frame sent during that sub-slot.

However, it is not possible to perform instantaneous frequency detection in case of collision and in the presence of noise. This is also why LoRa detects frequencies over a long duration SD.

Slots, sub-slots and synchronization

GS-MAC is a slotted protocol: it requires all end-devices to share a common time through synchronization, and end-devices can only start sending at the beginning of a slot, with a small random backoff, as the sub-slot.

However, this is no longer compatible with LoRaWAN devices, which transmit using ALOHA random access. Moreover, the synchronization of GS-MAC is difficult to achieve in practice for the low power devices.

Other weaknesses of GS-MAC

Repeated symbols cause uncertainties. GS-MAC considers that intersecting the frequencies of M_s sub-slots will remove all values of symbols sent on different sub-slots. This property is only true when the values of these symbols are changed. When consecutive symbols have identical values, that is when they are repeated, GS-MAC will not be able to remove the corresponding value from the intersection of sets, and uncertainties will arise. Let us consider the example of Figure 4.1, where two frames in collision are sent in different sub-slots. Frame 1 (in blue) contains the value 12 repeated twice, and Frame 2 (in orange) contains symbol 44 followed by symbol 32.

- For the first symbol of Frame 1: Based on the intersections of the frequency sets, we obtain $\{12, 16\} \cap \{32-32, 44-32\} \cap \{64-64, 76-64\} \cap \{96-96, 108-96\} = \{12, 16\} \cap \{0, 12\} \cap \{0, 12\} \cap \{0, 12\} = \{12\}$ as the first symbol in Frame 1, which is correct.
- For the second symbol of Frame 1: Only the first sub-set appears on the example, which is not sufficient to decode this symbol. We can still notice that the correct value, which is 12, belongs to $\{0, 12\}$.
- For the first symbol of Frame 2: Only the last sub-set appears on the example, which is not sufficient to decode this symbol. We can still notice that the correct value, which is 44, belongs to $\{12 - 96 + 128, 16 - 96 + 128\} = \{44, 48\}$.

- For the second symbol of Frame 2: Based on the intersections of the frequency sets, we obtain $\{32, 44\} \cap \{64-32, 76-32\} \cap \{96-64, 108-64\} \cap \{0-96+128, 12-96+128\} = \{32, 44\} \cap \{32, 44\} \cap \{32, 44\} \cap \{32, 44\} = \{32, 44\}$. Thus, there are still uncertainties in the decoded value of the second symbol of Frame 2: one value corresponds to the expected value (which is 32), while the other one corresponds to both the end of the first symbol of Frame 1 and the beginning of the second symbol of Frame 1, which are identical[1].

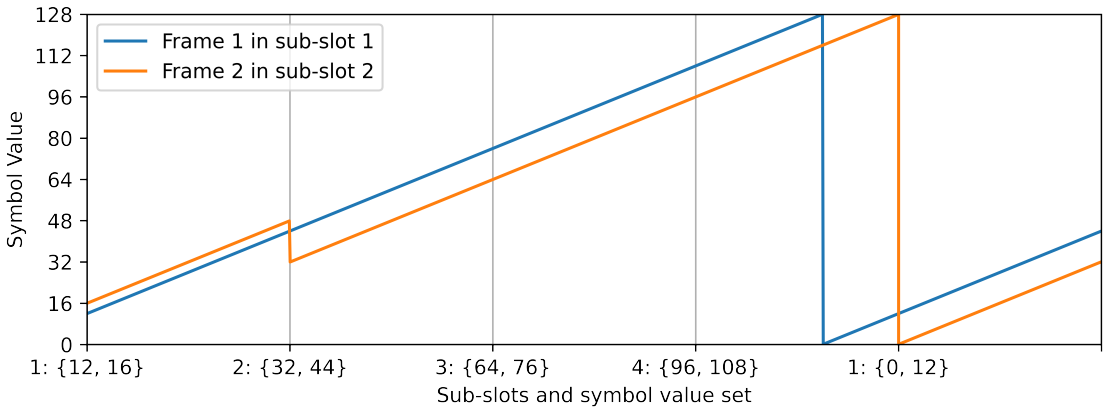


Figure 4.1: Frequency changes of up-chirps from two superposed LoRa frames, transmitted in sub-slot 1 and sub-slot 2, for the GS-MAC protocol. Because of the repeated symbol values in Frame 1, uncertainties occur while decoding Frame 2.

Accurate synchronization is hard to achieve. The synchronization among different end-devices is hard to implement, for two reasons.

- First, note that LoRa targets long-range communications. In the absence of multi-path effects, the propagation time is given by:

$$Propag = \frac{d}{c}, \tag{4.1}$$

where d is the distance between an end-device and the gateway, and c is the speed of transmission of the carrier. The frames from different LoRa end-devices can thus be delayed by different propagation times. To synchronize them in terms of sub-slot, a fixed end-device has to compute its propagation time and compensate it, which requires a precise clock and accurate geographical information. With mobile end-devices and considering the effects of multi-path, this becomes unfeasible. In addition, although end-devices generally equips a Real-Time Clock (RTC), they can also progressively loose synchronization. Some end-devices could have GPS module to correct the time. However, it will be activated with parsimony in order to save energy.

[1]Note that our proposal from [61], and described later in this thesis, is likely to be able to remove such uncertainties.

- Second, using a synchronization mechanism similar to the beacons of Class B of LoRaWAN might help to implement a synchronization. However, these beacons do not provide an accurate synchronization among the different end-devices. While the ping slots of Class B tolerate a symbol-level delay (as the minimum duration of a ping slot is 32 ms according to [41]), GS-MAC requires a sample-level synchronization, which is much lower. Indeed, the interval between two samples in a signal of LoRa frame is $SD/M = 1/BW$, which is only 8 μs with $BW = 125$ kHz bandwidth.

Insufficient downlink window for acknowledging collided frames. GS-MAC defines a downlink window at the end of each transmission slot. When two colliding frames are successfully decoded by GS-MAC, which often occurs when they are sent on different sub-slots, the gateway may have to acknowledge both of them during the same receive window. However, this issue is not discussed in [13]. Although acknowledgements are rather small and two of them might be sent during a single receive window, the receive window clearly does not scale with a large number of resolved colliding frames.

4.1.2 Removing the instantaneous frequency detection oracle

It is not possible to detect the frequencies at the frontier of each sub-slot instantaneously in practice, especially when there are colliding LoRa frames. The main reason is that the signals received by a LoRa digital receiver is composed of several phase samples. When there is only a single LoRa frame, the detection of frequency is possible by iterating over several samples. The gradient between the neighboring phase samples gives the change of frequency, when there is no noise. However, such frequency detection is inaccurate because of the presence of noise. Interference can also greatly change the gradient due to superposed phases. The standard LoRa receiver thus performs de-chirp on all of the samples during an interval of duration SD, and computes the FFT to extract the frequencies, as described in Subsection 2.1.2. The accumulation and averaging of frequencies in such processing bring robustness to noise and to interference.

In this Subsection, we present how SF-DS can obtain the set of frequencies at the frontier of each sub-slot by sampling over the entire sub-slot.

Estimating the instantaneous frequency at the middle of a sub-slot. In the demodulation of LoRa, each symbol is represented by 2^{SF} samples, while the sample rate is equal to BW. As SD is divided into M_s sub-slots in GS-MAC, each sub-slot has $N_s = \frac{2^{SF}}{M_s}$ samples. To keep the same resolution as the standard LoRa demodulation, the samples in each sub-slot are moved to the center, where the samples are around $t = 0$ in a SD between $-T_s/2$ and $T_s/2$, and padded with 0. Let us denote the time offset from the center

of the sub-slot i to $t = 0$, as τ_i. The time offset depends on the i-th sub-slot, as follows:

$$\tau_i = \begin{cases} -\frac{i}{M_s}SD - \frac{1}{M_s}\frac{SD}{2}, & \text{for } 0 < i < \frac{M_s}{2}, \\ \frac{i}{M_s}SD + \frac{1}{M_s}\frac{SD}{2}, & \text{for } \frac{M_s}{2} \leq i < M_s. \end{cases} \tag{4.2}$$

After the centering, the equivalent frequency for each sub-slot is:

$$\begin{aligned} f_u^m(t+\tau_i) &= \frac{M}{SD^2}(t+\tau_i+\tau_m) \\ &= \begin{cases} \frac{M}{SD^2}(t+\frac{m}{M}SD) - \frac{M}{SD^2}\frac{i}{M_s}SD - \frac{M}{SD^2}\frac{1}{M_s}\frac{SD}{2}, & \text{for } 0 < i < \frac{M_s}{2}, \\ \frac{M}{SD^2}(t+\frac{m}{M}SD) + \frac{M}{SD^2}\frac{i}{M_s}SD + \frac{M}{SD^2}\frac{1}{M_s}\frac{SD}{2}, & \text{for } \frac{M_s}{2} \leq i < M_s, \end{cases} \\ &= \begin{cases} f_u^m(t) - (\frac{i}{M_s}BW + \frac{1}{M_s}\frac{BW}{2}), & \text{for } 0 < i < \frac{M_s}{2}, \\ f_u^m(t) + (\frac{i}{M_s}BW + \frac{1}{M_s}\frac{BW}{2}), & \text{for } \frac{M_s}{2} \leq i < M_s, \end{cases} \\ &= \begin{cases} f_u^m(t) - f_{subslot}^i, & \text{for } 0 < i < \frac{M_s}{2}, \\ f_u^m(t) + f_{subslot}^i, & \text{for } \frac{M_s}{2} \leq i < M_s. \end{cases} \end{aligned} \tag{4.3}$$

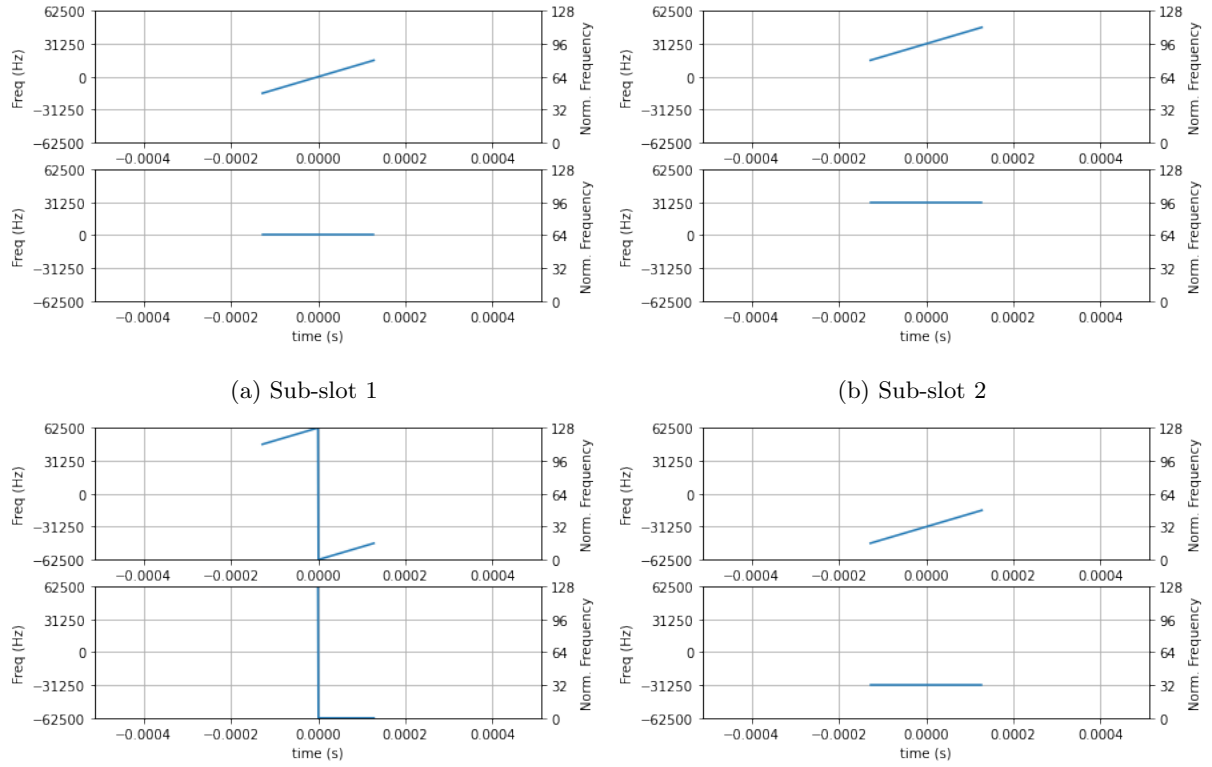
As in standard LoRa, let us now multiply the N_s samples by a normalized down-chirp to remove the time-variant. Thus, we can obtain a constant frequency component from the FFT results for one symbol in each sub-slot. The corresponding frequency can be described as follows:

$$f_u^m(t+\tau_i) + f_d(t) = f_u^m(t) + f_d(t) + f_{subslot}^i = \begin{cases} \frac{m}{M}BW + f_{subslot}^i, & \forall t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{m}{M}BW - BW + f_{subslot}^i, & \forall t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{4.4}$$

This frequency corresponds to the instantaneous frequency at the middle of a sub-slot (with an offset $\pm(\frac{i}{M_s}BW + \frac{1}{M_s}\frac{BW}{2})$) for up-chirps and down-chirps. The resulting frequency can be then translated to the instantaneous frequency at the beginning of the sub-slot by subtracting an offset corresponding to $\frac{1}{M_s}\frac{BW}{2}$. Therefore, the symbol value of each sub-slot i is equal to $m_i = (m + \frac{i}{M_s}M) \mod M$.

Let us consider the example of Figure 4.2, which shows this computation for sub-slot 1 (top left) and sub-slot 2 (top right), both for an up-chirp of value 48 sent during sub-slot 1 with SF7 and $BW = 125$ kHz, and with $M_s = 4$ sub-slots. The sub-figures on top of Figure 4.2(a) and on top of Figure 4.2(b) show the part of the up-chirps received during sub-slots 1 and 2, respectively. The sub-figures on the bottom of Figure 4.2(a) and on the bottom of Figure 4.2(b) show the result of the multiplication by the down-chirp. After the de-chirp operation, the resulting frequency for sub-slot 1 is 64 (which corresponds to 0 Hz for SF7, as SF7 yields $2^7 = 128$ values), which corresponds to the instantaneous frequency of the chirp value 48 at the middle of the first sub-slot (that is, after a duration of $SD/8$). The instantaneous frequency at the beginning of the first sub-slot is thus $64 - 2^{SF}/8 = 64 - 16 = 48$, as expected. Similarly, the instantaneous frequency of the chirp at the beginning of the second sub-slot is equal to $96 - 2^{SF}/8 = 96 - 16 = 80$. Note that 80

is the correct value, as $48 + 32 = 80$. The same process can be repeated for sub-slots 3 and 4, represented respectively at the bottom left and bottom right of Fig. 4.2.



(a) Sub-slot 1

(b) Sub-slot 2

(c) Sub-slot 3

(d) Sub-slot 4

Figure 4.2: The frequencies for sub-slots 1 to 4, when encoding symbol 48 with SF7 and $BW = 125$ kHz, with 4 sub-slots. For each sub-slot, the top sub-figure represents the partial up-chirp received by the receiver, and the bottom sub-figure represents the result of the de-chirp operation.

By implementing frequency detection at the beginning of a sub-slot, using continuous sampling over the whole sub-slot, SF-DS is able to reuse the core algorithm of GS-MAC. Therefore, we expect our new mechanism to follow the theoretical performance of GS-MAC: it is to be able to decode most frames sent alone in their own sub-slot, even if there are colliding frames sent during other sub-slots.

It's important to note that this version of SF-DS is still unable to decode frames that are sent in the same sub-slot. Figure 4.3 shows an example of this, where the left part displays the FFT results for the first sub-slot, and the right part displays the FFT results for the second sub-slot. In both cases, two main peaks can be observed. Since the reception power of the two frames is the same in this example, the height of each peak is the same, and the receiver cannot match the symbols to the frames. It's worth noting that the peaks 64 (on the left) and 96 (on the right) correspond to the symbol value 64, while the peaks 112 (on the left) and 16 (on the right) correspond to the symbol value 112.

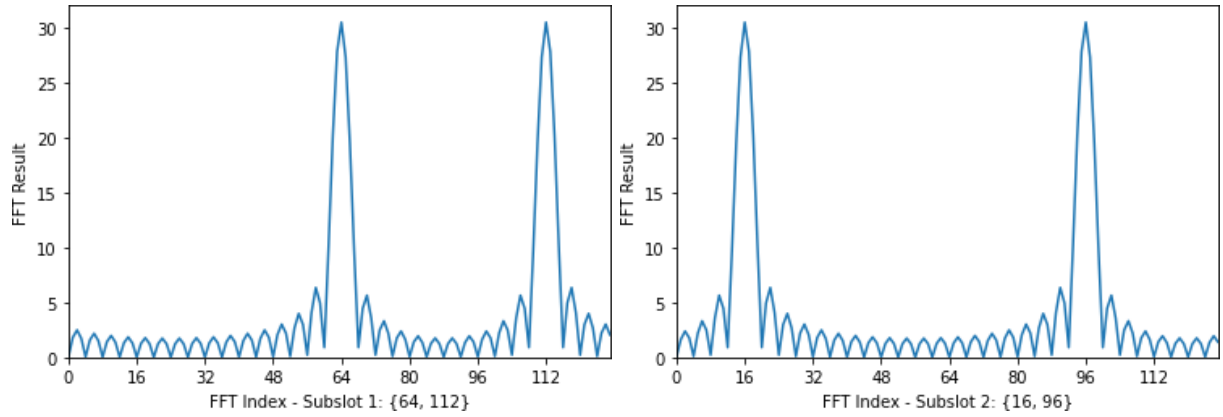


Figure 4.3: Result of the FFT for sub-slots 1 (on the left) and 2 (on the right), for two symbols that are synchronized. The values of the symbols are 64 and 112. $SF = 7$, $BW = 125$ kHz, and there are four sub-slots overall.

Alleviation of the imperfections due to the FFT window. Dividing the SD into intervals, causes imperfect FFT results, and yields inaccuracy in the value estimation. Let us take a look at the properties of LoRa symbols to understand the origin of these FFT imperfections in SF-DS.

Typical LoRa demodulators perform an FFT to detect the symbol frequency based on 2^{SF} samples. At the beginning of each symbol, that is when $t = -\frac{SD}{2}$, the signal has an initial phase of ϕ_0. At the end of the same symbol, that is when $t = \frac{SD}{2}$, the signal returns to its initial phase as follows:

$$\begin{aligned} s(\frac{SD}{2}) &= e^{j\phi_0 + j2\pi(\frac{M}{2SD^2}(\frac{SD}{2})^2 - \frac{M}{8})} \\ &= e^{j\phi_0 + j2\pi(\frac{M}{8} - \frac{M}{8})} \\ &= e^{j\phi_0}. \end{aligned} \tag{4.5}$$

Since $s(\frac{SD}{2}) = s(-\frac{SD}{2})$, the signal over the entire SD can be considered periodic and continuous, which results into a single peak after the FFT in LoRa demodulation.

However, in SF-DS, the signals are divided into several segments (one per sub-slot) and padded with zero. This operation breaks the previously perfect periodicity, which yields to spectral leakage in the FFT. This can be observed by comparing Fig. 2.14 (perfect FFT results) and Fig. 4.3 (imperfect FFT results).

In order to reduce the effects of spectral leakage, the signal can be multiplied by a window function before performing the FFT on each sub-slot. The window function is a mathematical function that smoothens the signal at its edges to avoid sudden changes in amplitude and phase at both the beginning and the end of the signal. The main issue with spectral leakage is that it can widen the main lobe of the FFT, to the point where the main lobe has a width larger than one value. This width depends on the window function used by the FFT. We chose to use a Hamming window, which brings a resolution of about 5 symbol values. In other words, the difference between the highest detected peak and the actual value can be up to 5 due to

the close symbols from superposed frames. This behavior can be observed on the second peak of the top-left sub-figure of Figure 4.4, which is wider than the first (however, the peak width is still smaller than one in this example though). Another example is when two close symbols are superposed, for instance 92 and 94: the detected value can be erroneous, with an error of 1 or 2 values. Consequently, SF-DS might not be able to decode the symbols by performing intersections over several virtual sub-slots. To correct this intrinsic demodulation error, we decided to add a tolerance of up to 5 symbols in SF-DS, which corresponds to the resolution of the window function. This tolerance is taken into account when performing the intersection of the frequency sets. While it brings occasional symbol errors of less than 5 values, it decreases the missing symbols due to the intersections in GS-MAC.

4.1.3 Removing slots and sub-slots

In this subsection, we explain how SF-DS removes the synchronization requirement, as well as slots and sub-slots. Note that SF-DS still uses the sub-slot to compute symbol values. However, we call *virtual sub-slots* from here, in order to distinguish the virtual sub-slot in SF-DS and the sub-slot in GS-MAC. However, the transmissions do not need to be aligned with these virtual sub-slots, making SF-DS compatible with legacy LoRaWAN devices.

In SF-DS, the receiver constantly checks for preambles. The first detected preamble becomes the reference preamble, and enables the receiver to be synchronized with the frame. When new preambles are detected, the receiver stores the relative delay between the new preamble and the reference preamble. Then, SF-DS divides SD into $2.M_s$ intervals which are synchronized with the reference frame, but not necessarily with the new frames. The relative delay of each frame is translated into one of these $2.M_s$ intervals. Intervals $2.M_s - 1$ and 0 correspond to virtual sub-slot 1 of GS-MAC, intervals 1 and 2 correspond to virtual sub-slot 2 of GS-MAC, and so on. When a symbol s_{dec} is obtained at the frontier of a virtual sub-slot, the original symbol s_{org} can be computed based on the relative time delay d of the frame and the frontier of a virtual sub-slot, using $s_{org} = s_{dec} - (2^{SF}.d/SD)$.

Figure 4.4 shows two examples of delayed frames. On the top, the delay is $d = (1/4).(SD/M_s)$, while on the bottom, the delay is $d = (3/4).(SD/M_s)$, both with $M_s = 4$ virtual sub-slots. The sub-figures on the left show the FFTs for virtual sub-slot 1, and the sub-figures on the right show the FFTs for virtual sub-slot 2. For the top sub-figures, the delay is $SD/16 \in [0; SD/8)$, so the second frame is considered as being received in interval 0. Thus, our algorithm performs the FFTs as if the two frames were completely synchronized. The FFT peaks in virtual sub-slot 1 are 64 and 104, which correspond to an original symbol of the second frame equal to either $64 - 8 = 56$ or $104 - 8 = 96$ (which is the correct value), due to the delay d. For the bottom sub-figures, the delay is $3.SD/16 \in [SD/8; SD/4)$, so the new frame is considered as being received in interval 1. Thus, our algorithm performs the FFTs as if the two frames were sent on different virtual

sub-slots. The FFT peaks in virtual sub-slot 2 are 96 and 120, which correspond to an original symbol of the second frame equal to either $96 - 24 = 72$ or $120 - 24 = 96$ (which is the correct value). In both cases, the correct value of the symbol was successfully retrieved.

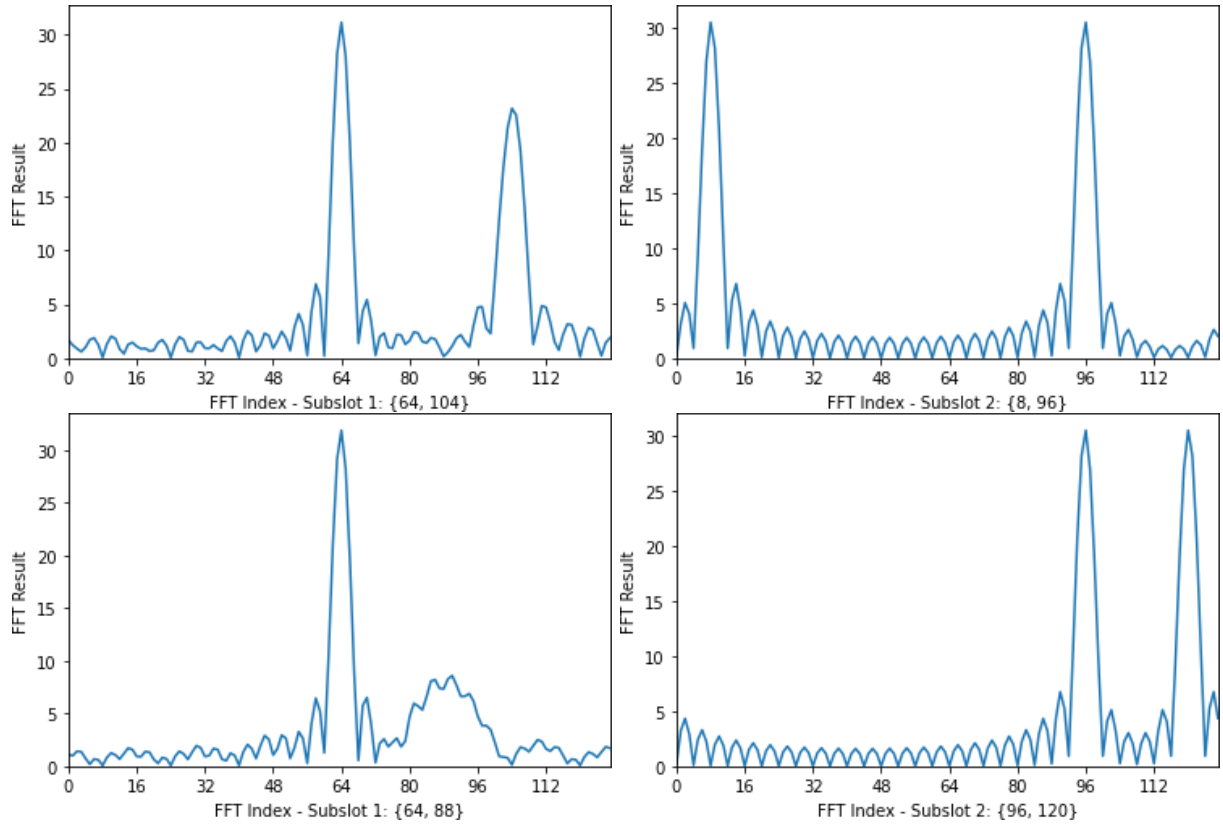


Figure 4.4: FFT results in virtual sub-slot 1 (top-left and bottom-left) and virtual sub-slot 2 (top-right and bottom-right) for two symbols that are desynchronized by one quarter of a virtual sub-slot (top-left and top-right) and three quarters of a virtual sub-slot (bottom-left and bottom-right). The values of the first symbol is 48, and the value of the second symbol, which is delayed, is 96. SF is 7, $BW = 125$ kHz, and there are eight intervals (corresponding to four virtual sub-slots).

The FFT peaks of a delayed frame are usually weaker than the FFT peaks of the reference frame, due to the delay. This can be seen on the second peak of the top-left sub-figure of Figure 4.4. Indeed, even if the second frame was transmitted shortly after the first frame, it is still considered being transmitted in the same virtual sub-slot due to the delay rounding mechanism, introduced to remove the synchronization. This weaker FFT peak is easily influenced by the side lobes from the first frame, and is thus likely to be incorrectly decoded. In order to cope with this, we also use the FFT results from the next virtual sub-slot (shown on the the top-right sub-figure of Figure 4.4), as these results are less influenced by this delay, and are thus stronger. For instance, the demodulator might be unable to determine whether the peak of the

delayed frame in virtual sub-slot 1 is 104 (which is the correct value) or 105, due to its relatively large side lobe. However, the demodulator is able to identify that the peak is weaker than what is expected. Thus, demodulator checks for virtual sub-slot 2: it identifies a clear peak at value 8, which corresponds to $8 - 32 + 128 = 104$, and thus identifies the correct value of the first virtual sub-slot as 104.

4.2 Preamble detection in ideal conditions

Let us now describe the preamble detection mechanism for SF-DS. The preamble detection mechanism is crucial in all collision resolution mechanisms, as its detection performance directly affects the decoding performance. Indeed, several features of the LoRa signal are extracted during the preamble detection. However, it is quite challenging to accurately detect preambles when frames collide. Note that few collision resolution papers actually describe their preamble detection mechanism.

When there is no ongoing transmission detected, the preamble detection of SF-DS is the same as the one of legacy LoRa: the receiver continuously performs FFT on the received signal for each SF and for each channel, in order to detect potential preambles. Once a series of repeated up-chirps is detected (typically, three or four repeated symbols are required), the receiver can synchronize with this potential preamble by shifting the FFT window with a value equal to the repeated symbol value (as there is only a time offset in ideal conditions), *i.e.* by either advancing or delaying a certain time to synchronize the FFT window with the symbol edge. Figure 4.5 demonstrates this process.

After several identical up-chirps (usually, eight), the receiver still needs to check the sync word, which is encoded using two up-chirps. EU868 defines the sync word used by public LoRa networks as equal to 0x34. Finally, the receiver needs to find 2.25 down-chirps representing the SFD in order to confirm the existence of an incoming LoRa frame. A receiver might discard a preamble for the following reasons: the sync word does not match, there is no sync word, or there is no down-chirps after several SD.

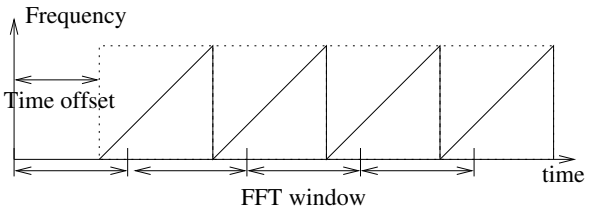


Figure 4.5: A legacy LoRa receiver performs continuous FFTs on the received signal. When it detects a repeated symbol value, it resynchronizes itself with the supposed preamble: the resynchronization delay is equal to the repeated value.

In SF-DS, the information of the potential preamble is stored in a list by the receiver. The receiver can also measure the power level of the symbols of the preamble. Then, the receiver begins to divide each SD

into virtual sub-slots and starts to demodulate the symbols.

Algorithm 3 describes our preamble detection in ideal conditions. Its role is to populate two sets S_u and S_d with a list of possible symbol values for up-chirps and down-chirps, respectively. Algorithm 4 describes the frame detection in ideal conditions. Its role is to follow the frame format, and to extract its symbols. It introduces several states for the potential preambles, including $PREAMBLE$, $SYNC1$, SFD, $HEADER$ and $PAYLOAD$. The $PREAMBLE$ state means that the receiver is still in the preamble, the $SYNC1$ state means that the first word of the SFD has been found, the SFD state means that the receiver is in the SFD, the $HEADER$ state means that the receiver is decoding the first eight symbols in the frame, and the $PAYLOAD$ state means that the remaining symbols are being decoding.

Algorithm 3: Ideal preamble detection.

Input : Samples of signals;
Threshold of FFT peaks T;
Spreading factor SF;
Number of repeated symbols N_{repeat}.

Output: Detected LoRa frames $frames$.

$S_u \leftarrow$ a list of the sets of symbol values for the detected peaks from up-chirps.
$S_d \leftarrow$ a list of the sets of symbol values for the detected peaks from down-chirps.
$frames \leftarrow$ a list of information of detected LoRa frames.
foreach 2^{SF} *samples* **do**
  values, *powers* $\leftarrow$ FFT(2^{SF} samples $\times$ a normalized down-chirp for SF)
  values_set $\leftarrow \emptyset$
  foreach *value, power in values, powers* **do**
    if *power* $\geq T$ **then**
      values_set $\leftarrow$ *values_set* $\cup \{value\}$
  Append *values_set* to S_u
  values, *powers* $\leftarrow$ FFT(2^{SF} samples $\times$ a normalized up-chirp for SF)
  values_set $\leftarrow \emptyset$
  foreach *value, power in values, powers* **do**
    if *power* $\geq T$ **then**
      values_set $\leftarrow$ *values_set* $\cup \{value\}$
  Append *values_set* to S_d
  $frames \leftarrow$ ideal-frame-detection(S_u, S_d, SF, N_{repeat})

Algorithm 4: Ideal frame detection.

Input : A list of the sets of symbol values for the detected peaks from up-chirps S_u;
A list of the sets of symbol values for the detected peaks from down-chirps S_d;
Spreading factor SF;
Number of repeated symbols N_{repeat};
Existing LoRa frames *old_frames*.

Output: Detected new LoRa frames *frames*.

occurrence ← a mapping of symbol values into the number of occurrences
sync_word_1 ← the first sync word
sync_word_2 ← the second sync word
foreach *values_set in the last* N_{repeat} *sets in* S_u **do**
  foreach *value in values_set* **do**
    if *value in occurrence* **then**
      Increment the number of occurrences of *value* in *occurrence*
    else
      Set the number of occurrences of *value* to 1 in *occurrence*
foreach *value in occurrence with* N_{repeat} *or more occurrences* **do**
  frame ← the frame with $2^{SF} - value$ time offset and $PREAMBLE$ state in *old_frames*
  if *there is no such frame* **then**
    Append a new frame to *frames* with $2^{SF} - value$ time offset and $PREAMBLE$ state
  else
    so ← $2^{SF}-$ time offset of *frame*
    state ← state of *frame*
    if *state is* $PREAMBLE$ *and so* + *sync_word_1* = *value* **then**
      Change the state of *frame* to $SYNC1$
    else
      if *state is* $SYNC1$ *and so* + *sync_word_2* = *value* **then**
        Change the state of *frame* to SFD
      else
        if *state is* SFD *and so is in the last two sets of* S_d **then**
          Change the state of *frame* to $HEADER$
          if *there is no frame with state* $HEADER$ *in old_frames* **then**
            `// Not yet synchronized frame, we synchronize with the first one.`
            to ← time offset of *frame*
            Resynchronize with the first frame by *to* samples
            Update the symbol values in both S_u and S_d with *to*
    Append *frame* to *frames*

4.3 Decoding in an imperfect channel

In practice, the channel is not perfect, and both channel and hardware imperfections make the preamble detection and the decoding challenging. For instance, LoRa signals can suffer from time offsets, frequency offsets and noise. In this section, we will introduce their impacts on SF-DS.

4.3.1 Dealing with time offsets

Let us focus on the time offset experienced by a LoRa frame. Recall that the sample rate of a LoRa receiver is $F_s = BW$, that is, the receiver takes F_s samples per second. The sampling time is thus $\frac{1}{F_s}$ second, which is also the time resolution with the given sample rate. Let us consider that there is a single LoRa frame. Since the LoRa receiver needs to synchronize with the symbol edges of the LoRa frame, the time offset to the symbol edge can be thus divided into two parts: the integral time offset and the fractional time offset, as shown in Figure 4.6 on a simplified example (with a very low sampling rate). The integral time offset is proportional to the sampling time, and can be compensated after being detected during the preamble. However, the receiver is not able to compensate the fractional time offset because of the limited time resolution. This fractional part causes fractional STO.

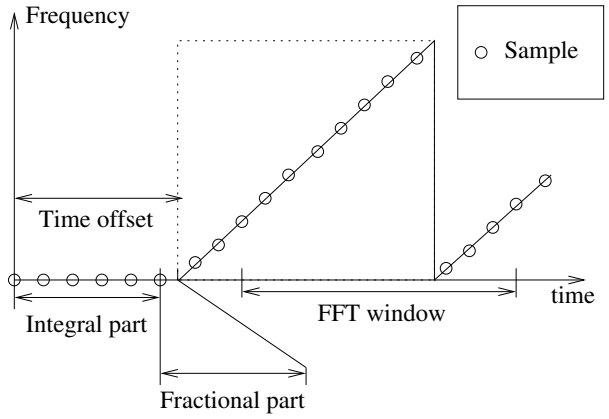


Figure 4.6: LoRa uses several samples to compute symbols. The time offset between the symbol deduced from the observed samples and the actual symbol can be divided into two parts: an integral time offset and a fractional time offset.

In the following, let us denote by T_{off} the integral time offset, and by t_{off} the fractional time offset. Thus, the total time offset is $\tau_{sigma} = T_{off} + t_{off}$. In the following, we discuss how SF-DS aims at resolving the integral time offsets and the fractional time offset, during the preamble.

When a receiver received the first LoRa frame, it is not yet synchronized with a frame, as shown in Figure 4.6. For a better synchronization, the receiver needs to use a higher sample rate (typically $k.BW$,

where $k \geq 1$ is a factor for oversampling), in order to reduce the synchronization errors due to the large sampling interval. The receiver thus needs to shift the FFT window within SD, with less than $k.M$ samples, where M is the minimum number of samples to decode a symbol after the synchronization, using a sample rate equals to BW. This shift allows the receiver to synchronize with the symbol edges, that is in order to use the symbol edges as a reference. Indeed, the time offset τ_{sigma} changes the symbol value by σ, according to Equation 2.18 (if we do not consider the frequency offsets). For instance, if the demodulated symbol value at the beginning of preamble is 48 (while it should be 0), the receiver needs to advance the FFT window by 48 samples to compensate for the time offset and to align with the actual symbol edge.

When a receiver detects a new frame in collision with the current one, it is no longer able to adjust the FFT window to the second frame due to the synchronization with the first frame. In SF-DS, the integral time offset is used to allocate the incoming frames to the corresponding virtual sub-slot. The strategy (described in Subsection 4.1.3) is to allocate the frame to the nearest virtual sub-slot. The computation of the index of the nearest virtual sub-slot i, whose index varies from 0 to $M_s - 1$, can be described as follows:

$$i = \left\lfloor (T_{off} + \frac{1}{2}\frac{SD}{M_s})\frac{M_s}{SD} \right\rfloor \mod M_s. \tag{4.6}$$

In addition, SF-DS needs to know the start of the incoming frames in terms of symbols. Similarly, the integral time offset also helps to compute the number of samples and the number of symbols between the preambles ΔN_s, as follows:

$$\Delta N_s = \left\lfloor (T_{off} + \frac{1}{2}\frac{SD}{M_s})\frac{1}{SD} \right\rfloor. \tag{4.7}$$

Figure A.9 shows an example of two frames (Frame 1 and Frame 2) whose preambles overlap. The receiver has already synchronized with Frame 1 when it detects Frame 2. The integral part of the relative time offset tells the receiver that Frame 2 is allocated to the second virtual sub-slot ($i = 1$), and it starts with the same symbol as Frame 1 ($\Delta N_s = 0$).

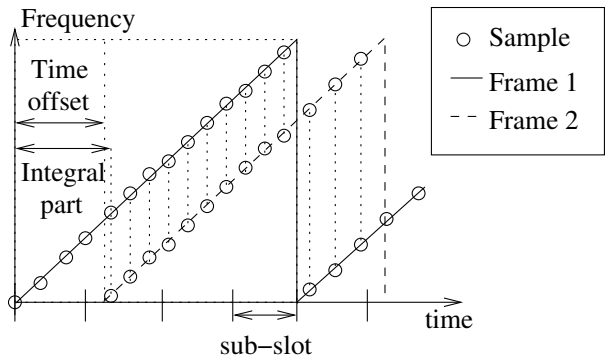


Figure 4.7: Since SF-DS is synchronized to the first received frame, the STO from all the other colliding frames cannot be ignored.

While the fractional time offset has little influence when the receiver is synchronized with a single frame

(as it is the case with legacy LoRa), the fact that SF-DS cannot synchronize with all colliding frames causes the STO to have a large impact on all frames, except the one to which the receiver is synchronized with. Recall that the demodulation uses a sample rate equals to BW, which is usually lower than the sample rate during the preamble detection. Indeed, the possible STOs of non-synchronized frames become larger due to the lower sample rate after the synchronization, which results in longer intervals between samples. Such STO can change the symbol value by at most one, provided that $|t_{off}| > 0.5$.

In SF-DS, we only take STO into account by rounding the integral part T_{off}. Indeed, this consideration gives the property that $|t_{off}| \leq \frac{1}{2k.F_s}$ for the synchronized frame, because the frame are synchronized using a higher sample rate $k.F_s$. Meanwhile, the time offset for the following collided frames in SF-DS are limited by $|t_{off}| \leq \frac{1}{2F_s}$, because the receiver then uses a lower sample rate F_s for decoding. Any $|t_{off}| > 0.5$ contributes to a ± 1 value to the T_{off}.

4.3.2 Dealing with frequency offsets

LoRa signals suffer from a CFO, due to the frequency mismatch between the low-cost crystal oscillator of the transmitter and the crystal oscillator of the receiver [17]. Assuming that oscillators in both transmitter and receiver have a precision within ± 20 ppm, the relative offset can thus reach up to ± 40 ppm. In a LoRa network operating at 868 MHz, the CFO is thus $868 \times 40 = 34.72$ kHz at maximum. Similarly as the time offset, the CFO can also be considered as having an integral part CFO_i and a fractional part CFO_f. The initial frequency of a chirp can thus have a frequency offset of $CFO_i + CFO_f$. It can be noticed that CFO remains stable during the transmission of an entire LoRa frame (as it is an impairment from the hardware).

Equation 2.18 and Equation 2.20 show that both time offsets and CFO can lead to different symbol values in the preamble, the header and the payload of a LoRa frame. Authors in [6] show that they are indistinguishable and prove that the detectable CFO in LoRa is between $\pm 25\%$ of the BW. Figure 4.8 shows the same synchronized LoRa frame without CFO, with a negative CFO of -25% of the BW, and with a positive CFO of $+25\%$ of the BW. Assuming that the LoRa frame is encoded with SF7 under 125 kHz BW, the resulted symbol value in the preamble, which should be 0, varies between $2^{SF} \times (1 - 0.25) = 96$ for -25% CFO and $2^{SF} \times (1 + 0.25) = 32$ for $+25\%$ CFO.

The receiver does not know whether the symbol value is changed by time offsets, CFO or both. If such error is not processed, the synchronized frame will have all symbol values in the preamble, the header and the payload shifted, and will not be successfully decoded.

Considering that the time offset and the frequency offset are equivalent in LoRa, there are two potential solutions to cope with CFO during the demodulation: one is the conversion of the CFO to time offset, and the other is to distinguish CFO and time offsets. Let us now describe these two solutions.

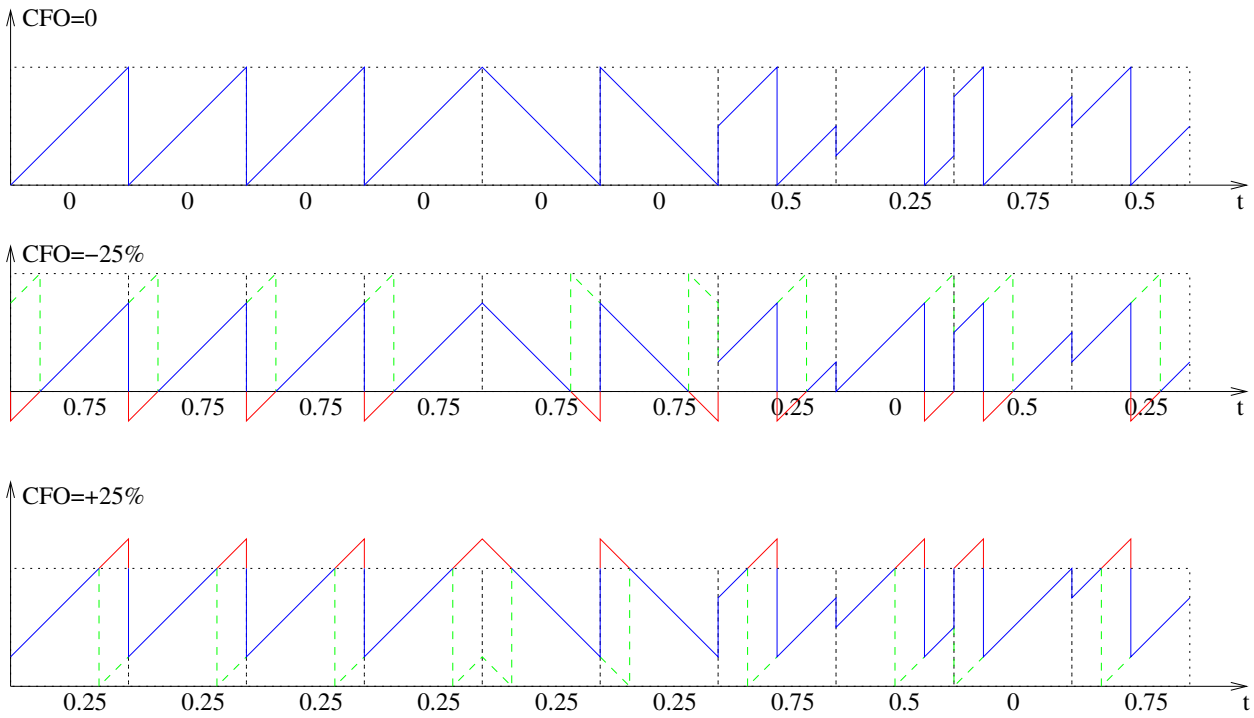


Figure 4.8: CFO effect on a (simplified) LoRa frame. The top sub-figure has a CFO of 0, the middle sub-figure has a CFO of -25%, and the bottom sub-figure has a CFO of +25%.

Conversion to time offset

To convert the CFO to a time offset, the receiver does not require special processing. Instead, the CFO_i can be fully converted to the time offset by the synchronization of the receiver with the symbol edge. We will first explain this solution, and show why it is not suitable to SF-DS.

Figure 4.9 shows an example of a simplified LoRa frame. The repeated symbol values in the preamble are 0 when the frame is already synchronized and transmitted without CFO (which is represented with dashed lines). When the CFO is equal to -25% of the BW, all the frequencies in the signals reduce accordingly in each chirp (which is represented with solid lines). If we consider that all of them are caused by the time offset, we obtain from Equation 2.18:

$$\frac{\sigma}{SD} = -0.25BW \implies \tau_\sigma = -0.25SD, \tag{4.8}$$

which means that the symbol edges seen by the receiver are advanced by $\frac{SD}{4}$. The receiver thus is supposed to delay by $\frac{SD}{4}$ in order to synchronize with the (incorrect) symbol edges, which corresponds to the green squares in Figure 4.9. Such synchronization also changes the symbol values in the header and the payload correspondingly, which brings the equivalent symbol values at the new incorrect symbol edges. Such approach works with an arbitrary CFO less than half of the BW using the standard demodulation in LoRa.

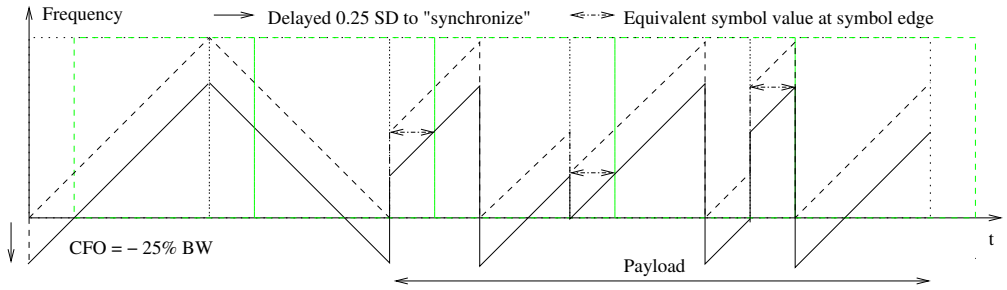


Figure 4.9: Effect of conversion of a $CFO = -25\%$ of the BW to a time offset on a (simplified) LoRa frame.

However, this solution cannot be applied to SF-DS. Indeed, a frame might disappear from the first virtual sub-slot to which it belongs if the compensation of the CFO brings a time offset that is too large (*e.g.*, larger than half of one virtual sub-slot). If this happens, the receiver is not able to find the correct symbol values. Besides, due to the incorrect symbol edges, the missing parts of the LoRa signals reduce the height of the FFT peaks, which degrades the decoding capabilities of SF-DS. Thus, we consider that the conversion of CFO to time offset is not a good option for SF-DS.

Distinguishing CFO and time offsets

Another solution is to distinguish the CFO and the time offset. With this solution, the receiver first synchronizes with the actual symbol edges using the time offset information. After demodulation, the receiver

needs to eliminate the change of symbol values caused by the CFO.

Authors in [6] present the basic synchronization algorithm that can extract both the time offsets and CFO. This algorithm leverages both up-chirps and down-chirps in the preamble to estimate the integral parts of the two types of offsets as follows:

$$CFO_{int}/BW = (S_{up} + S_{down})/2, \tag{4.9}$$

where S_{up} is the value of the repeated up-chirps of the preamble, and S_{down} is the value of the 2.25 down-chirps of the preamble. This algorithm works well when the CFO is between -25% and +25%, and is dysfunctional otherwise.

Therefore, we replace the processing of the SFD stage of Algorithm 4 with Algorithm 5.

Algorithm 5: Processing of up-chirps and down-chirps when a frame is in state SFD

Input : A list of the sets of symbol values for the detected peaks from up-chirps S_u;
A list of the sets of symbol values for the detected peaks from down-chirps S_d;
Spreading factor SF;
Number of repeated symbols N_{repeat};
Existing LoRa frames old_frames.

Output: Detected new LoRa frames $frames$.

if *state is* SFD **then**

```
// Find the down-chirps.
```

foreach $down-chirp$ *in the last two sets in* S_d **do**

Change the state of $frame$ to $HEADER$

$actual_{to} \leftarrow (down-chirp-so)/2$

$fo \leftarrow (down-chirp+so)/2$

Update the time offset of $frame$ to $actual_{to}$

Update the frequency offset of $frame$ to fo

if *there is no frame with state* $HEADER$ *or* $PAYLOAD$ *in* old_frames **then**

```
// There is no synchronized frame, we synchronize with the first one.
```

$to \leftarrow$ time offset of $frame$

Resynchronize with the first frame by to samples

Update the symbol values in both S_u and S_d with to

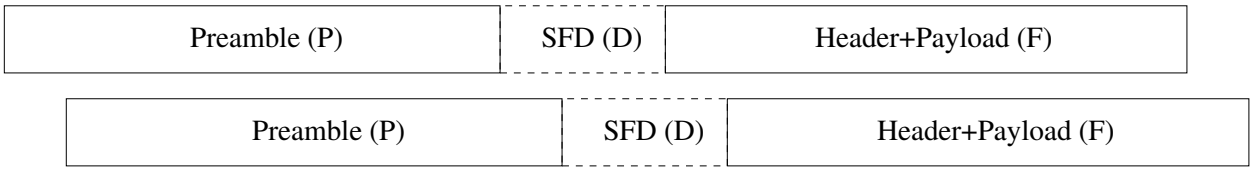
SF-DS is based on this solution. However, there are still certain cases of collisions that SF-DS cannot resolve due to incorrect preamble detection. To explain these cases, let us consider two frames that collide with each other. Each frame can be divided into three parts: the up-chirps of the preamble (referred to as P), the down-chirps of the preamble (referred to as D), and the up-chirps of the header and the payload

(referred to as F), as illustrated in Figure 4.10. Depending on the arrival time of the second frame, we can classify the collisions into three categories:

- Case 1 is when the second frame arrives during the preamble of the first frame and the down-chirps of both frames overlap partially. We call it P-D-P collision.
- Case 2 is when the second frame arrives during the preamble of the first frame, but the down-chirps of both frames do not overlap. We call it P-P collision.
- Case 3 is when the second frame arrives during the header or the payload of the first frame. We call it P-F collision.

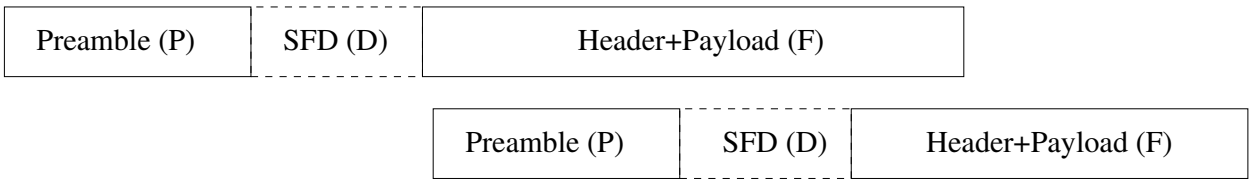
These three cases are shown on Figure 4.11.

Figure 4.10: The structure of a LoRa frame according to the type of chirps: up-chirps are used for the beginning of the preamble, down-chirps for the end of the preamble, and up-chirps for the header and payload.



(a) Case 1: P-D-P collision

(b) Case 2: P-P collision



(c) Case 3: P-F collision

Figure 4.11: The three cases of collisions between two LoRa frames.

Case 1, illustrated in Figure 4.11a, can be further divided into two sub-cases.

- Case 1.1: In the first sub-case of Case 1, the time offset between the two frames is less than SD. Note that the 2.25 down-chirps overlap (since this is a sub-case of Case 1). The receiver needs to extract the time offset and the CFO of both frames using the symbol values from the up-chirps and the down-chirps. The superposed down-chirps from the two frames represent two possible values. For example, let us assume that the receiver demodulated {16, 48} from the consecutive up-chirps of the preamble, and {32, 16} from the two full down-chirps of the preamble. The number of possible combinations to compute the CFO is four, as shown in Table 4.1. This brings ambiguity between the CFO and the time offset for each frame. Although it could be possible to iterate over all four possibilities to decode the frames, our SF-DS algorithm and implementation is not able to directly discover the correct frames. Fortunately, Figure 4.12 shows that Case 1.1 is a rare case.

- Case 1.2: In the second sub-case of Case 1, the time offset between the two frames is larger than SD, but less than $2.25 \times SD$. Thus, there are at most 1.25 superposed down-chirps. The down-chirps of the two frames only contribute to at most one symbol in common. In our algorithm for preamble detection, the receiver uses the two repeated symbol values from the down-chirps to identify the end of the preamble. The time offset makes the down-chirps from the two frames distinguishable. There is no more ambiguity of the CFO and time offset. Our SF-DS can decode both frames without ambiguity.

Table 4.1: Four possible CFOs computed from two up-chirps {16, 48} and two down-chirps {16, 32}.

Up-chirp, down-chirp	16, 16	16, 32	48, 16	48, 32
CFO	16	24	32	-8

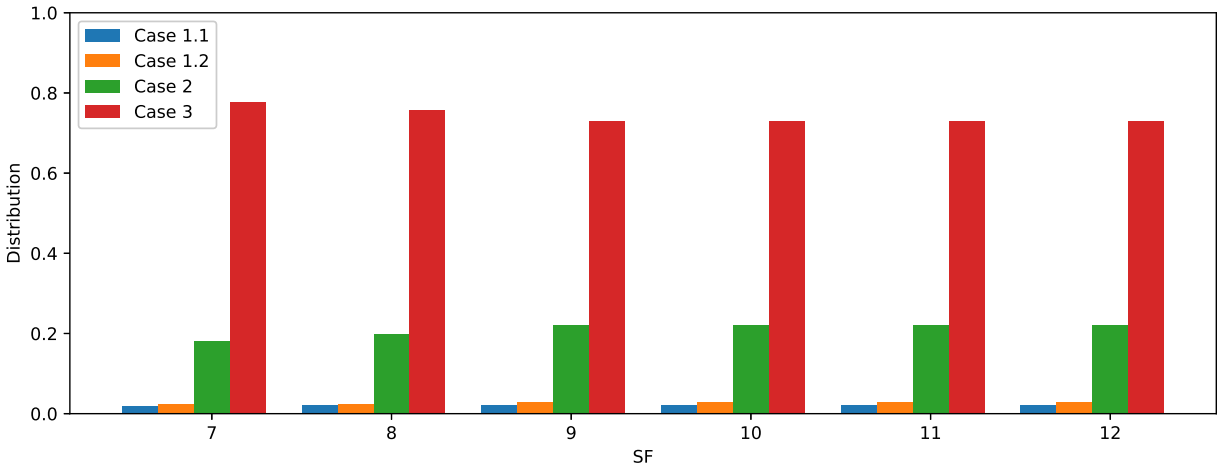


Figure 4.12: Distribution of the different types of collisions between two LoRa frames.

Case 2 is shown in Figure 4.11b. The time offset and CFO can be calculated by combining the up-chirps and down-chirps. Despite the negative impact on the demodulation of the superposed up-chirps due to P-F overlapping, the receiver can eliminate the incorrect symbol values from the preamble using information obtained during the beginning of the preamble detection. Subsequently, SF-DS can decode the remaining symbols from the header and the payload, which has F-F overlap. Additionally, since the up-chirps and down-chirps are orthogonal, as explained in Subsection 2.1.2, the overlap between the up-chirps and the down-chirps (P-D and F-D overlapping). SF-DS can decode both frames.

Case 3 is shown in Figure 4.11c. To clarify, in addition to P-F and F-D overlapping, there may also be F-F overlapping, depending on the length of the earlier frame. However, all of these types of overlapping can be processed using the approach described in Case 2. Again, SF-DS can decode both frames.

Note that Case 1.1 is the only case that SF-DS cannot decode. However, it is a rare occurrence, as shown in Figure 4.12.

Also note that SF-DS is able to resolve collisions among more than two frames.

4.3.3 Dealing with noise

There are several aspects of noise that SF-DS has to deal with.

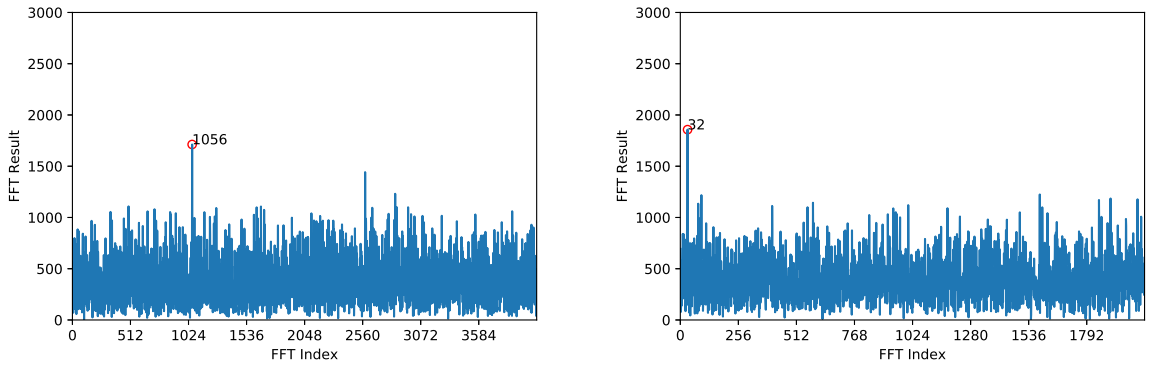
Impact of virtual sub-slots on the robustness to noise. The noise strength is not impacted by the division of virtual sub-slots. Indeed, the division of virtual sub-slots also divides the noise into several segments. After de-chirping, the noise can be still regarded as white noise, which results in similar power levels in the frequency domain.

However, the signal strength is impacted by the division of virtual sub-slots. Since there are less samples that contribute to peaks from the signal, thus the peaks representing the symbol values are weakened. We have shown previously the FFT results of a symbol of value 32, encoded with SF12 under $SNR = -20\ dB$, in Figure 2.8. We hereby show the FFT results of the same symbol in the same conditions, but with several virtual sub-slots. We do not normalize the FFT result, in order to compare the absolute values of the power distributions. Thus, Figure 4.13a and Figure 4.14a show the FFT result of the first virtual sub-slots in SF-DS, with either two virtual sub-slots (for Figure 4.13a) or four virtual sub-slots (for Figure 4.14a), respectively. The index of the desired peak in the first virtual sub-slot is given by:

$$\hat{m} = m + \frac{2^{SF}}{2M_s}, \tag{4.10}$$

which is 1056 in case of two virtual sub-slots, and 544 in case of four virtual sub-slots. We show all the peaks that are higher than the desired symbol values. From Figure 4.13a, in case of two virtual sub-slots, we can observe that the absolute power level of the desired symbol 1056 is around 1700. This absolute power level is about half of the result of the entire symbol, which was shown on Figure 2.8. From Figure 4.14a, in case

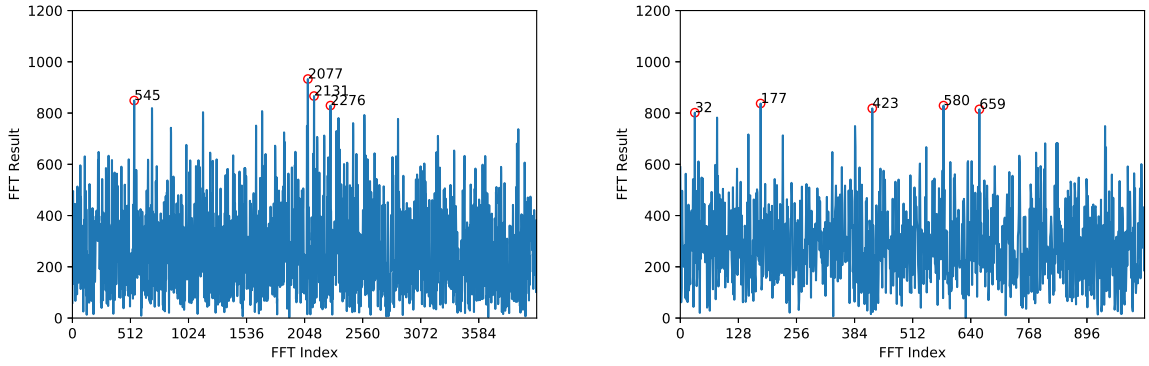
of four virtual sub-slots, the peak of the desired symbol is already biased, and becomes 545 instead of 544. The corresponding absolute power level is also around 800, which is about one fourth of the absolute power of the entire symbol. Note that the noise brings the other peaks higher than the desired symbol values, which falsifies the symbol value estimation, if the receiver uses the legacy LoRa demodulation without extra operations.



(a) SF12 with two virtual sub-slots

(b) SF11

Figure 4.13: Peaks and demodulated symbol values for symbol 32 encoded with SF12, using different numbers of virtual sub-slots under $SNR = -20\ dB$.



(a) SF12 with four virtual sub-slots

(b) SF10

Figure 4.14: Peaks and demodulated symbol values from symbol 32 encoded with SF12, using different numbers of virtual sub-slots under $SNR = -20\ dB$.

Since the absolute power levels depend on the number of samples contributing to the peak, the demodulation performance with virtual sub-slots is similar to the demodulation without virtual sub-slots and with lower SF. For example, using two virtual sub-slots under SF12 brings similar robustness to the noise as

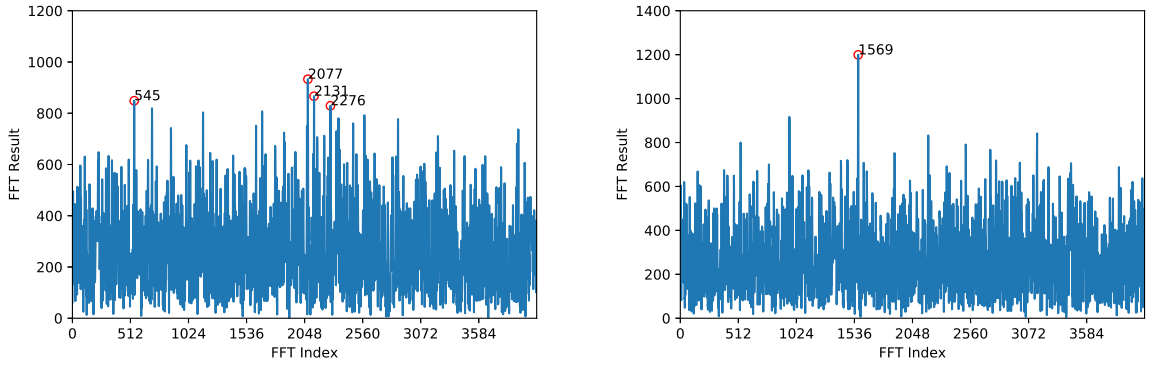
under SF11 in legacy LoRa, while using four virtual sub-slots under SF12 brings similar robustness to the noise as under SF10 in legacy LoRa. To validate this, we performed FFTs on the symbol value 32 encoded with SF11 in Figure 4.13b, and with SF10 in Figure 4.14b. By comparing Figure 4.13a and Figure 4.13b, as well as Figure 4.14a and Figure 4.14b, we can conclude that the division of virtual sub-slots weakens the robustness to the noise, with a loss of about 3 dB each time the number of virtual sub-slots doubles.

Robustness to burst noise. Burst noise on a certain frequency could also affect the estimation of symbol values, due to lower signal peaks in the FFT. Indeed, we can observe several peaks caused by burst noise in the first virtual sub-slot of Figure 4.15, which are not present in the second virtual sub-slot of the same figure. SF-DS considers all the peaks above a certain threshold, including the peaks due to burst noise. Although this produces more uncertainties, it also increases the probability of obtaining the correct symbol value. Indeed, in SF-DS, there are two features that remove the erroneous symbol values caused by the noise.

- SF-DS is able to remove the peaks from the noise, if they do not follow the frequency changes that are expected in adjacent virtual sub-slots. For example, with a tolerance of 5 symbol values, the peaks {545, 2077, 2131, 2276} of Figure 4.15a should produce peaks within {1569 (= 545 + 1024) ± 5, 3101 (= 2077 + 1024) ± 5, 3155 (= 2131 + 1024) ± 5, 3300 (= 2276 + 1024) ± 5}. However, in Figure 4.15b, there is only 1569 in the second virtual sub-slot, thus the correct value for the first slot was 1569-1024=545.
- In case of a large amount of detected peaks in each virtual sub-slot which cannot be removed by the previous feature of adjacent virtual sub-slots, the algorithm from Chapter 3 can help remove the remaining uncertainties.

Therefore, although the division of virtual sub-slots brings a loss of about 3 dB each time the number of virtual sub-slots doubles, our decoding scheme is still able to decode the symbol values by combining the results from different virtual sub-slots. We will show the noise robustness of SF-DS later, in Subsection 6.3.4.

Spectral leakage. The spectral leakage due to the division of virtual sub-slots is another shortcoming of SF-DS. We proposed to apply an FFT window function in order to alleviate this imperfectness. However, the window functions can also attenuate the power levels of the peaks, which decreases the robustness to the noise. The choice of the number of virtual sub-slots is crucial for spectral leakage, and impacts both the decoding performance and the robustness to the noise. We will show our choice for the number of virtual sub-slots later, in Subsection 6.3.5.



(a) Virtual sub-slot 1

(b) Virtual sub-slot 2

Figure 4.15: Peaks and demodulated symbol values in the first virtual sub-slot (virtual sub-slot 1) and the second virtual sub-slot (virtual sub-slot 2) from symbol 32 encoded by SF12, using four virtual sub-slots under $SNR = -20\ dB$.

Chapter 5

Open source LoRa simulator

LoRa and LoRaWAN simulators allow researchers to study the performance of LoRa-based IoT applications and devices without the need for physical hardware, which can be costly and time-consuming to set up. There are several simulators with different simulation levels and granularity, as detailed in [29].

We believe that the LoRa physical layer must be accurately modeled, in order to accurately simulate both LoRa and LoRaWAN. Note that the development of the physical layer is difficult due to the fact that the LoRa modulation is proprietary, but since several researchers have retro-engineered the modulation, they have helped to design accurate simulation models.

In this chapter, we first present the existing implementations for LoRa and LoRaWAN simulators. Second, we present our implementation of a LoRa physical layer simulator on GNU Radio, as well as the implementation of some features of LoRaWAN. Third, we show how our simulator enables users to perform large scale simulations. Fourth, we introduce our improvements to an existing LoRaWAN module made with the NS-3 simulator, which is used to evaluate our proposed methods from Chapter 3. Finally, we show the differences between our generated LoRa frames and real LoRa frames, generated by commodity LoRa devices.

5.1 Existing implementations for LoRa and LoRaWAN simulators

The performance of LoRa collision resolution algorithms is typically hard to study. Indeed, in order to study collisions, researchers either need a large number of devices or need to configure the devices to exceed the maximum duty-cycle of 1% in Europe, in order to produce enough collisions. Additional collisions might also occur due to pre-deployed end-devices from other networks or from concurrent technologies on the ISM band. In addition, the CFO, the STO and the noise are often hard to predict or to control, and their randomness make the experiments difficult to realize. Thus, setting up a real test bed is difficult, and simulators are

often used.

We consider in the following two types of simulators: those based on network simulators that allow to simulate LoRa and LoRaWAN, and those based on radio simulators.

5.1.1 Existing network simulators

Several simulator platforms for wireless networks exist, including SimPy [50] and NS-3 [1]. These platforms have extensions that allow the simulation of LoRaWAN networks.

SimPy is a process-based discrete-event simulation framework based on standard Python, released under the MIT license. It is the basis for both LoRaSim and LoRaWANSIM.

- LoRaSim [8] is one of the earliest released simulation tool for LoRa. It uses SimPy to implement a discrete-event simulator that can simulate a network with several end-devices and multiple gateways in a 2-dimensional space.

- LoRaWANSIM [34] extends LoRaSim with features of the MAC layer, such as bidirectional communications. However, LoRaWANSIM is not open-source.

Note that these two versions assume perfect SF orthogonality.

NS-3 is also a discrete-event simulator, that implements a large variety of network protocols. NS-3 uses the GNU GPLv2 license, which guarantees that the extensions of NS-3 are accessible.

- [56] describes an NS-3 module for LoRaWAN networks. The module is capable of dealing with multi-gateway networks and bi-directional traffic. It uses a channel model derived from baseband simulations of a LoRa transceiver over an AWGN channel.

- [54] introduces an NS-3 module for Class A of LoRaWAN. It implements a Carrier-Sense Multiple Access (CSMA) channel access, which outperforms the default ALOHA channel random access of LoRaWAN. The NS-3 module accounts for the capture effect and is capable of evaluating the energy consumption of end-devices.

- [9] and [28] propose an implementation of Class A of LoRaWAN. Their NS-3 module accounts for the pseudo-orthogonality of SFs and supports downlink traffic. The module is also able to evaluate the energy consumption during the simulation.

However, all of the above simulators concentrate on the MAC layer. They provide a model of the physical layer at the frame level, which considers only the parameters to send the packets and the time-on-air of the frames. Such simulation level is not sufficient to meet our needs, as we focus on collisions of LoRa frames at the symbol level.

5.1.2 Existing radio simulators for LoRa

A few researchers and engineers have attempted to reverse-engineer LoRa, since the first attempt in [25].

- Authors in [38] present an open-source LoRa demodulation module for GNU Radio. The module computes the correlation between a reference up-chirp and the received signals to detect the preamble of LoRa frames. Once the LoRa frame is detected, the module is able to demodulate the symbols by detecting the sudden frequency change. Then, the decoder is able to decode the frame with any CR. However, the redundancies added by the module are not considered during the decoding. In addition, due to its implementation of preamble detection and frequency change, the module is not able to handle the CFO and frames in collision.

- Other open-source LoRa modules based on GNU Radio are described in [53] and [30]. The authors reverse-engineered the different stages in LoRa coding and divided them into different implementations. The two modules are thus able to encode and decode a single LoRa frame. However, the gray decoding is not presented in [30].

Most of the existing collision resolution algorithms described in Section 2.3 use sophisticated signal processing techniques, which are often programmed in different environments, such as Matlab and Python. However, some of them are not well described and technical details are missing in the papers. This means that the corresponding performance results are not reproducible. Moreover, the performance of LoRa collision resolution algorithms greatly depends on several physical parameters, including the SNR of the LoRa signals, the presence of a CFO, the presence of a time offset, and the STO. These parameters are not studied consistently in the literature, which makes the existing algorithms hard to compare.

5.2 Our simulator focusing on collision resolution

The main goal of our simulator is to implement and compare various collision resolution protocols, taking into account various parameters of a physical LoRa signal, such as noise strength, CFO, and STO. To the best of our knowledge, there is no framework available that can do this.

The core of our framework, called gr-loraphysim[1], is composed of two parts: a decoder and a node abstraction. The decoder contains an optional LoRa preamble detector, a payload decoder (in which the developer can implement a collision resolution algorithm, including the demodulation and certain aspects of decoding) that retrieves demodulated symbols from a signal, and a LoRa decoder that transforms demodulated symbols into bits. The node abstraction is capable of generating superposed LoRa signals with different parameters from a simulated network.

[1] gr-loraphysim is available at `https://doi.org/10.5281/zenodo.6421804`

In the following sub-sections, we briefly introduce GNU Radio. Then, we describe the decoder and the node abstraction. Finally, we discuss our implementation of existing collision resolution protocols.

5.2.1 Introduction to GNU Radio

GNU Radio is a toolkit that enables to develop signal processing algorithms for radio communications. The basic element in GNU Radio is the block. A block usually contains input and output ports, from which it can respectively receive and send data to other blocks. The flowgraph is the basic data structure in GNU Radio, and represents the connections of the blocks. The data is either a continuous stream (*e.g.*, a signal is often represented as a stream of complex float numbers representing the I and Q components of the signal) or a discrete message containing data with polymorphic type (*i.e.*, a generic structure use to hold a large variety of data such as a number, a string, a tuple, a list or a dictionary). A block is able to attach tags (another type of discrete message) to stream data. The tags can carry some extra information, such as the mark of the arrival of a new frame, the frequency of the transmitted signals, etc.

GNU Radio has an intuitive graphical interface, where blocks are interconnected through links between output and input ports. Figure 5.1 shows an example of a block. There are one input port and two output ports. The input port accepts the streamed data, while the two output ports send streamed data and discrete message data, respectively. In GNU Radio, blue ports use a stream of complex float numbers, while white ports represent messages.

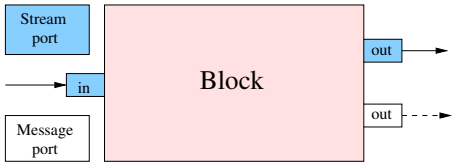


Figure 5.1: A block in GNU Radio, with an input for streamed data, and two outputs (one for streamed data, and one for a discrete message).

Each block can be considered as a state machine with some initial parameters and internal states. A signal processing block consumes the data from its input ports, processes it, and produces data for its output ports. The data are produced by source blocks, which can come from a file, a network, an SDR hardware, etc. In contrast, the sink blocks terminate the data flows. When the data source is an SDR hardware, the data is exactly the signal captured from the radio environment. The data sent to an SDR hardware would be transmitted as a real signal with a certain sample rate.

In GNU radio, the blocks do not require to load all the data at once, which greatly reduces the memory consumption during the processing. In the contrary, it allows to load a small portion of samples, to process them, and to move forward with parts of the samples. Such mechanism is also consistent with the signal

processing in the low-cost hardware, which is memory-limited and time-sensitive.

Figure 5.2 shows an example where the data source is a file (block "File Source") or a connected SDR hardware (block "UHD: USRP Source", in which UHD means that the block is powered by USRP Hardware Driver). The source is connected to a "Throttle" block to limit the data rate of the stream. The output of the "Throttle" block is connected to the input of the "Resampler" block to change the sample rate of the data stream. After the "Resampler" block, the "Decoder" part is one of the core components of our proposed framework focusing on the collision resolution.

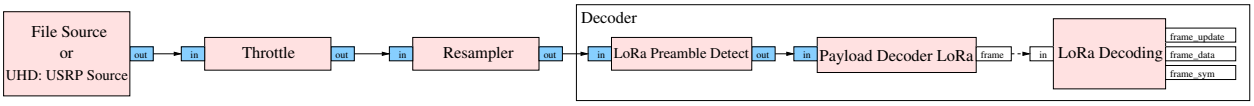


Figure 5.2: Decoding LoRa frames from a signal sample, obtained from a file or from a USRP SDR hardware.

It is worth noting that any GNU Radio block can be substituted by another block with the same input and output, provided that they implement similar features. Such decoupling can help to develop, to evaluate and to improve any algorithm, including the preamble detection algorithm, the payload demodulating algorithm (which is often the core of collision resolution algorithms), and the frame decoding algorithm.

5.2.2 Decoder

Figure 5.2 shows the structure of the decoder part implementing a single LoRa receiver with a given SF, on the right. The entire receiver consists of a preamble detector, a payload decoder (for the data symbols) and a LoRa frame decoder.

Preamble detector

The receiver first takes the samples from a source block. The "LoRa Preamble Detect" block processes the samples in order to find a preamble. The preamble detection algorithm implemented in the block is the one described in Section 4.3. The samples are outputted as-is in the output port. However, once there is a detected frame, the preamble detector attaches a tag containing the corresponding frame information.

The tag is a dictionary implemented by a polymorphic type, which contains a set of key-value mappings. The position of the tag on the samples marks the beginning of the symbols of the payload. The format of the carried data is described in Table 5.1. It contains:

- the ID of the frame detected by the current preamble detector, which auto-increases when a new potential frame appears,
- the length of the detected frame, counted in symbol, when available (note that the length is usually unknown during the preamble detection, because it is usually decoded from the header),

- the SF and the frequency offset of the detected frame.

Each value is an array, because the presence of multiple frames is possible when several frames collide.

Table 5.1: The format of information carried by the streamed sample data after preamble detection.

Key	Value type	Example of values	Description
`i`	Integer array	[1, 2]	The IDs of the detected frames
`s`	Integer array	[23, 18]	The length (in symbols) of the frames
`f`	Integer array	[7, 12]	The SFs of the detected frames
`freq_offset`	Real number array	[4.6, -3.2]	The frequency offsets of the frames

Payload decoder

The samples, combined with the preamble information of the detected frame, are then processed by the "Payload Decoder" block in order to synchronize the receiver with the transmitted symbols. In the example of Figure 5.2, the standard LoRa decoding algorithm is used. The block keeps monitoring the tags passed by the previous blocks (typically from the preamble detector). When a tag is detected and there is no frame it is currently synchronized to, the payload decoder uses the position of the tag as the symbol edge for the synchronization with the frame. The block then starts demodulating the symbols. Finally, when there are eight demodulated symbols (corresponding to the header) or when the entire frame is demodulated (if the frame length in symbol is known), a discrete message is sent to the frame decoding block.

The discrete message transmitted from the "Payload Decoder" block to the "LoRa decoding" block is similar to the aforementioned tags, and contain the frame ID, the frame length and the SF. In addition, the demodulated symbol values are also wrapped into an array. Although the standard LoRa demodulation scheme only takes the index of the strongest peak during each SD, the implementation of other algorithms (whose implementation will be discussed in Subsection 5.2.4) can also produce several possible symbol values for each SD.

Frame decoder

The frame decoding algorithm is based on the description of Section 3.4. When only the eight first symbols are passed as a message to the "LoRa Decoding" block, the block tries to decode first the content of the header. In the explicit mode, the header carries the length of the frame, the presence of a CRC and the CR used to encode the frame. The decoded frame length can then be published as a feedback to notify the "Payload Decoder" block or the preamble detector through the `frame_update` port, in order to determine the end of the given frame. After demodulating all the symbols in the entire frame, this block decodes and outputs the original data from the symbols. Such data is published through the `frame_data` port.

5.2.3 Node abstraction

GNU Radio allows to directly generate data from an internal signal source, as well as from external sources such as a file or an SDR hardware. We thus implemented a LoRa node abstraction in order to simulate different scenarios of large LoRa networks.

Figure 5.3 presents the two node abstractions of our framework, shown as Node 1 and Node 2, which simulates two end-devices in a LoRa network. Node abstractions are also responsible for generating application data. These data can be randomly generated in GNU Radio for test purpose (as in Node 1) or from an external application (as in Node 2 for an example with User Datagram Protocol (UDP)) such as a network simulator. Note that the communication between the external application and GNU Radio can be made through Transport Control Protocol (TCP), UDP or ZeroMQ connections [18]. The generated data are packed as a message and passed to our "LoRa Encoding" block, which performs encoding and modulation, and produces LoRa signals. The encoding implementation is based on the descriptions of Section 3.4.

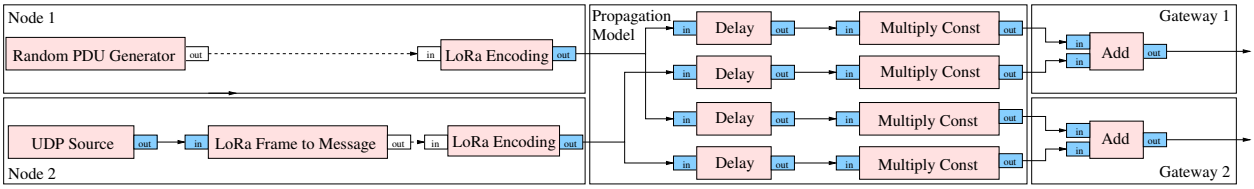


Figure 5.3: Generating signals with two node abstractions and two gateways.

Implementing multiple end-devices. By default, the signals generated by our "LoRa Encoding" block have a normalized amplitude and are perfectly synchronized at the sample level. It is possible to implement a propagation model by delaying the signals (see the "Delay" blocks on Figure 5.3, which can be configured with a given number of samples) and by reducing the strength of the received signal with a given factor (see the "Multiply Const" blocks on Figure 5.3). Then, the signals from the two end-devices are added, and the output is a colliding signal. Note that a very similar setup can be configured to evaluate the impact of low SNR, by adding a strong random noise to a weak signal. Also notice that this delay is also used to implement desynchronized transmissions, required by several existing collision resolution algorithms. In the simulated scenario, the user can remove the "LoRa Preamble Detect" block. In such situation, the frame information is generated by the "LoRa Encoding" block, and then dispatched in the tags, along with the samples of the base band signals.

Let us consider a scenario where we want to simulate a single LoRa gateway receiving two colliding signals, with a relative power difference of 20 dB. From Equation 2.17, we know the relationship between the referenced signal and the interfered signal (as noise), as follows:

$$A_{ref} = A_{int} \cdot 10^{\frac{SIR}{20}}. \tag{5.1}$$

If we set the amplitude of the weaker signal A_{int} to 1, then the amplitude of the stronger signal A_{ref} has to be multiplied by $10^{20/20} = 10$.

Implementing multiple gateways. Multiple gateways can be implemented by adding preamble detector blocks and payload decoder blocks for each gateway. For each gateway, the parameters corresponding to the propagation model are likely to be different, as the signals received at each gateway will experience different channel losses. Therefore, we can set different parameters in the propagation models by calculating them in advance based on the relative distances between the end-devices and gateways. However, it is still difficult to set up a large scale network by hand due to the complexity of constructing the node abstractions of every end-devices to all the gateways. We will present how to automate this setup in Subsection 5.3.

Implementing advanced features. Noise is often considered as an AWGN with a zero mean [16]. In our framework, such noise can be simulated by the "Noise Source" and added to the superposed signals. The relative amplitude is computed according to Equation 5.1 as follows:

$$A_{noise} = A_{ref} \times 10^{-\frac{SNR}{20}}. \tag{5.2}$$

To simulate $SNR = 0$ dB, A_{noise} is set to the same value as A_{ref}.

CFO is not negligible in a real transmission from low-cost hardware. To simulate CFO in the base band signal, a CFO can be added to the 'LoRa Encoding' block as part of the node abstraction. The CFO is translated into an initial frequency offset in the "LoRa Encoding" block.

STO can occur when the re-sampler in the receiver side is not perfectly synchronized with the original signal. To simulate STO in our framework, we can configure the 'LoRa Encoding' block to generate signals with a higher sample rate using the interpolation parameter, and then re-sample the signal with sample-level offsets."

5.2.4 Implementations of collision resolution algorithms

We hereby discuss the implementations of various algorithms, as well as the reasons why some algorithms are not implemented in our framework.

The implemented algorithms are the following:

- Legacy LoRa consumes 2^{SF} samples during SD, multiplies it with a downchirp, performs an FFT on the samples, and finds the strongest FFT bin. The index of this bin is the obtained symbol value.
- CHOIR takes 2^{SF} samples from the synchronized frame during SD. CHOIR adds a zero padding to expand the FFT size to ten times its default value. Then, it stores the strongest FFT bin peaks into ten lists. The symbols are taken from these lists when decoding.

- FTrack requires parameters whose values are not given in the reference article [59]: the number of steps n_{step} between each FFT to extract a track, and the size of an FFT window N_{win}. Our block uses this sliding window on N_{win} samples to calculate the FFTs. The window slides by steps of n_{step} samples. To extract the track of an entire symbol, we thus need to review $2^{SF} + N_{win}$ samples.
- OCT aims at a real-time decoding of LoRa collisions. To do so, OCT divides each symbol duration SD into several segments according to the time offset between the frames. We implemented OCT with a minimal resolution of 1/8-th of SD, that is 0.128 ms for $SF = 7$ and $BW = 125$ kHz. The decoder needs to perform at most as many FFTs within every SD as there are collided frames.
- Our proposed SF-DS, presented in Chapter 4, is also implemented in our simulator, for comparison with the other algorithms and experimentation with real LoRaWAN frames in the future.

The main algorithms that were not implemented are the following.

- CIC was not implemented in our framework, as the authors of CIC already made their code available in both Python and Matlab.
- SCLoRa is not implemented as there is insufficient detail provided in the reference.

All these algorithms have in common the computation of several FFTs. This allows us to compare them depending on how many FFTs they perform, and on the size of these FFTs. The space complexity also mainly depends on the number of samples that a receiver has to review. Table 5.2 summarizes the complexities of each algorithm. We use the following notations: N is the size of each FFT (which depends on the algorithm), n_{sym} is the number of symbols in the collided frames, and n_c is the number of frames in collision. Note that in the case of CIC, the space complexity is $n_{sym} \times N$ according to the code published in [48], but it could be reduced to N by considering only the samples during an SD for each FFT.

Table 5.2: Complexity of the implemented collision resolution algorithms.

Algorithm	FFT size	Time complexity of FFT / symbol	Space complexity of FFT / symbol
LoRa	$N = 2^{SF}$	$\mathcal{O}(N \log N)$	N
CHOIR	$N = 10 \times 2^{SF}$	$\mathcal{O}(N \log N)$	N
CIC	$N = 2^{SF}$	$n_c \times \mathcal{O}(N \log N)$	$n_{sym} \times N$
FTrack	$N = 2^{SF}$	$n_{step} \times \mathcal{O}(N \log N)$	$N + N_{win}$
OCT	$N = 2^{SF}$	$n_c \times \mathcal{O}(N \log N)$	N
SF-DS	$N = 2^{SF}$	$n_{sub-slot} \times \mathcal{O}(N \log N)$	N

SIC-based algorithms for LoRa need to store the samples during the entire superposition (at least $n_{sym} \times 2^{SF}$ samples), so as to do the subtraction and recover a weaker signal. We believe that SIC-based approaches

need to be implemented by storing all the samples and analyzing them several times, in order to be compliant with our framework (and more generally, with the GNU Radio design).

5.3 Simulating large networks with our simulator

Large networks are usually difficult to simulate as they require creating and interconnecting several blocks, which is tedious and error-prone. To facilitate the configurations of large simulations, we developed a script with an optional configuration file to automatically generate a complex network. The script accepts a set of parameters for both end-devices and gateways. The parameters can control the overall simulation time, and the sample rate used in the simulation. The "LoRa Encoding" block is controlled by the parameters such as the duty-cycle, the sample rate, the interval of frames, and the SF to generate LoRa signals from frame data. The frame data are randomly generated, and their lengths are also given as a parameter of the script. A configuration file can also be included, and can store deployment information such as the location of the end-devices or gateways. Table 5.3 summarizes the available parameters of the script.

Table 5.3: Parameters of our script for large-scale simulations.

Parameter	Description
`end-time`	End time of the simulation
`gen-samp-rate`	Sample rate used by the end-devices and the gateway
`duty-cycle`	Duty-cycle required by the ISM regulation
`min-interval`	Minimal interval of frames (after duty-cycle)
`max-interval`	Maximal interval of frames (after duty-cycle)
`decoding-algo`	Algorithm used by the receiver to decode payload
`frame-len`	Length of payload data (in bytes)
`sf`	Spreading factor used by the end-devices and the gateways
`noise-amp`	Absolute value of the noise amplitude
`ini`	Configuration file for complex network simulation

Figure 5.4 shows an example scenario, where multiple end-devices are deployed around a LoRa gateway. The script allows deployment of end-devices with a uniform distribution within a ring of 500 to 1000 meters around the gateway, selection of the log distance model as the path loss model, and computation of noise amplitude equal to the amplitude of the weakest signal, resulting in a SNR of zero. The attenuation and the delay of the signals are computed and set to the node abstraction in the node abstractions of our proposed architecture. The corresponding script is given by Table 5.4.

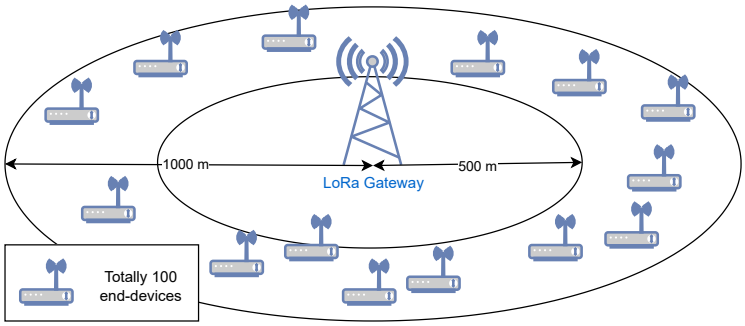


Figure 5.4: Example scenario with several end-devices around a LoRaWAN gateway.

Table 5.4: Script for the example scenario shown on Figure 5.4.

```
[DEFAULT]          [Nodes]                  [Loss Model]          [Noise]
Gateways=1         Distribution=uniform     Model=LogDistance     exist=YES
Nodes=100          Radius=1000              Param1=1              min=0
[Gateway1]         MinRadius=500            Param2=7.7
Algorithm=lora                              Param3=3.76
```

5.4 Improvement of the LoRaWAN module in NS-3

In order to evaluate the algorithm proposed in [61], we improved the NS-3 module of [9] and [28] in order to add a realistic LoRa physical layer. The different features used by different decoding algorithms are also modeled in order to launch the simulations. In this section, we present the main improvements based on this NS-3 module.

5.4.1 Initial implementation

The initial implementation of the NS-3 module described in [9] and [28] mainly concentrates on the LoRaWAN MAC layer, which contains the channel access, the duty-cycle constraint, and the communications between end-devices, gateways and network server. The LoRa physical layer was simply modeled, with a `Send` function accepting the LoRaWAN frame data and the transmission parameters. The authors described several configurations in the LoRa physical layer, including the SF, the frequency of channel and the power for the transmission. A gateway and an end-device have their own `Send` functions, which were defined in `simple-gateway-lora-phy.cc` and `simple-end-device-lora-phy.cc`, respectively. The common part of the two implementations is the computation of the duration of the frame, according to Equation 3.1. This duration was then used as a parameter for the transmission to the channel, along with the parameters from the upper layer. The channel has to find the receiver, to compute the receive time and to attenuate the

transmission power. Figure 5.5 shows the information used between the different layers.

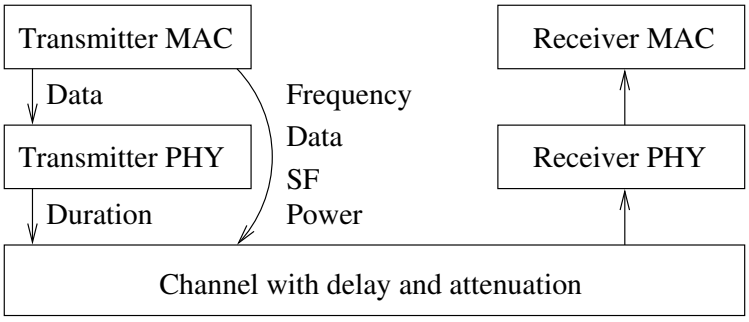


Figure 5.5: Channel modeling, physical layer abstraction and MAC layer abstraction of LoRa and LoRaWAN in the NS-3 module of [9] and [28]. In this initial implementation, the physical layer only calculates and passes the duration of the frame to the channel model, in order to determine whether frames are in collisions or not.

On the receiver side, the physical layer leverages the overlapped time to detect the collisions of frames. It calculates the accumulated power levels based on the SFs of collided frames. A frame with a given SF is considered as lost if one of the accumulated power levels exceeds the corresponding thresholds given in [11]. Otherwise, the frame is passed to the upper layer and forwarded to the network server for further processing. In addition, the module also limits the simultaneous decoding capacity to eight, as explained in [47].

5.4.2 Our improvements on the implementation

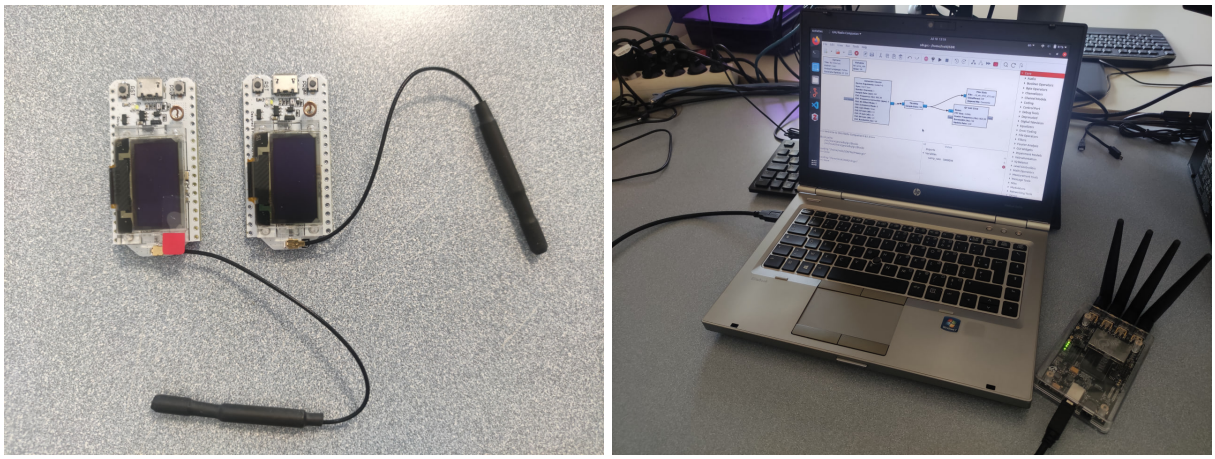
Our main contribution was to implement the LoRa coding, based on the knowledge from existing reverse-engineering works. We added the two files `real-gateway-lora-phy.cc` and `real-end-device-lora-phy.cc` to replace the original unrealistic LoRa physical layer. The encoded frames containing LoRa symbols are passed to the channel. On the receiver side, the physical layer allows different collision resolution algorithms to decode the frame data from the symbols. If a frame is decoded without error, it is passed to the upper layer. Otherwise, the frame is dropped. Indeed, both the "LoRa Encoding" and "LoRa Decoding" blocks of GNU Radio share similar code with these physical implementations on NS-3, which shows that the NS-3 implementation is realistic.

Three collision resolution algorithms are implemented in the simulations we described in [61] to evaluate the performance of the algorithm described in Chapter 6: CHOIR, SIC and GS-MAC. Therefore, the corresponding features had to be modeled. In CHOIR, as the frames are distinguished using the tiny frequency offsets, the simulator can add a random tiny frequency offset as a parameter, associated with an end-device. During the decoding, such frequency offsets are explored to determine whether a frame can be fully decoded. In SIC, after the attenuation, the receive power is leveraged for each frame to confirm if it can

be retrieved from the collision. In GS-MAC, the physical layer computes the symbol values at each sub-slot by combining the symbol values and the time offsets of the different frames in collision. If the corresponding conditions are not met, the physical layer composes the symbols from the different frames, which brings potential uncertainties. The simulation results obtained from this simulator are given in Chapter 6.

5.5 Comparisons of frames generated by our simulator with real frames

Our implementation can work with real hardware. Indeed, the SDR hardware is able to capture the signals around a center frequency, with a given bandwidth. Thus, we leverage our simulator to perform experiments in order to capture real LoRa frames. We used the SDR manufactured by BladeRF [33]. Our goal was to validate the consistency of the frames generated by our simulator with frames sent by a real LoRa end-device from HelTec [20]. The LoRa end-device is equipped with an ESP32 micro-controller and an SX1276 LoRa chip [42, 46], and is able to send LoRa frames with a widely customizable and easy-to-use library [19] provided by HelTec. Figure 5.6 shows the hardware setup of our experiments for the validation of LoRa frames, where Figure 5.6a shows the LoRa end-devices from Heltec, as the senders, and Figure 5.6b shows the receiver side, using the BladeRF SDR and the GNU Radio software.



(a) Transmitters (b) Receiver

Figure 5.6: Experiment setup.

We calculate the length of LoRa frame in terms of symbols and the corresponding payload length according to Equation 3.1. The end-devices are programmed to encode different sizes of data, with different parameters CR values.

5.5.1 Simple frames encoded with SF7, and with CR1 to CR4.

We first used simple frames encoded with SF7, and we changed the CR from 1 to 4. We disabled the CRC of the payload and filled all the symbol blocks up to the corresponding frame lengths, which prevents the encoder from adding padding into the frame. The frame is encoded in the explicit header mode, which consists of five nybbles indicating the frame length, the CR, the existence of CRC, and the Header CRC. We limit the number of symbol blocks to three: a header block and two symbol blocks. According to Equation 3.1, the frame lengths for CR1 to CR4 are always seven bytes. Thus, we set the payload data to `hello\0\0`, which is equal to the sequence `0x68 0x65 0x6c 0x6c 0x6f 0x00 0x00`.

The comparison of the frames generated by our simulator with the frames encoded with the real SX1276 end-device, is given in Table 5.5 (for CR1), Table 5.6 (for CR2), Table 5.7 (for CR3), and Table 5.8 (for CR4). Each difference is highlighted in red. We can observe that except for CR1, all the symbols from our simulator are consistent with the symbols from the real hardware. Most of the symbols from the frame encoded by CR1 are the same as well. The only differences are in the last symbol of each block, which consists of the ECC bits. Regarding that the header block always leverages CR4 to encode the data, we can conclude that the redundancy bits added by SX1276 are different from the redundancy bits added by our simulator, only in case of CR1. However, the different redundancies do not impact the performance of LoRa, because CR1 is only used by LoRa to detect errors, not to correct them.

Table 5.5: Validation test of our simulator for a simple LoRa, with CR1.

Symbol values	Header block: 8 symbols							
Real hardware	12	52	124	0	36	96	48	0
Our simulator	12	52	124	0	36	96	48	0

Block 1: 5 symbols					Block 2: 5 symbols				
53	124	32	70	56	10	17	103	83	125
53	124	32	70	108	10	17	103	83	98

Table 5.6: Validation test of our simulator for a simple LoRa, with CR2.

Symbol values	Header block: 8 symbols							
Real hardware	12	56	100	0	32	28	60	24
Our simulator	12	56	100	0	32	28	60	24

Block 1: 6 symbols						Block 2: 6 symbols					
53	124	32	70	108	49	10	17	103	83	98	15
53	124	32	70	108	49	10	17	103	83	98	15

Table 5.7: Validation test of our simulator for a simple LoRa, with CR3.

Symbol values	Header block: 8 symbols							
Real hardware	112	8	120	0	24	0	48	24
Our simulator	112	8	120	0	24	0	48	24

Block 1: 7 symbols							Block 2: 7 symbols						
53	124	32	70	107	50	15	10	17	103	83	84	105	102
53	124	32	70	107	50	15	10	17	103	83	84	105	102

Table 5.8: Validation test of our simulator for a simple LoRa, with CR4.

Symbol values	Header block: 8 symbols							
Real hardware	12	4	124	124	28	28	12	4
Our simulator	12	4	124	124	28	28	12	4

Block 1: 8 symbols								Block 2: 8 symbols							
53	124	32	70	107	50	15	127	10	17	103	83	84	105	102	12
53	124	32	70	107	50	15	127	10	17	103	83	84	105	102	12

In the following, we thus use the frame encoded by CR3 and SF7 as a reference, in order to explore the various parameters.

5.5.2 Unknown behaviour during padding or with CRC

We then remove one byte from the data, which results in a six bytes payload consisting of `hello\0`, which is not able to fill all the symbol blocks with SF7. Indeed, the removed one byte in the last symbol block, *i.e.*, two nybbles, needs to be padded by the encoder. In our simulator, we choose to pad with 0, and to process the padding as part of the payload data. However, the padded data are not counted in the payload length, encoded by the header block in the explicit header mode. We used SF7, an explicit header, and again no CRC. By comparing the symbols in Table 5.9, we discover that SX1276 might use a different implementation, which is an unknown behavior to us.

We remove a second byte from the payload data, in order to include the payload CRC of two bytes, using CRC-16-CCITT algorithm. The payload data then becomes `hell\0`. In the implementation of our simulator, the CRC is computed before the LoRa coding, and appended to the end of the data. The presence of CRC is also encoded in the header block. We used SF7, CR3, and an explicit header. Table 5.10 shows that there are still differences between the two frames when the CRC is added, although the symbols of the header block and the first payload symbol block are consistent.

Table 5.9: Validation test of our simulator with padding.

Symbol values	Header block: 8 symbols							
Real hardware	124	52	100	0	4	0	8	120
Our simulator	124	52	100	0	4	0	8	120

Block 1: 7 symbols							Block 2: 7 symbols						
53	124	32	70	107	50	15	10	46	119	92	87	107	102
53	124	32	70	107	50	15	10	17	103	83	84	105	102

Table 5.10: Validation test of our simulator with CRC.

Symbol values	Header block: 8 symbols							
Real hardware	108	48	120	0	60	112	4	28
Our simulator	108	48	120	0	60	112	4	28

Block 1: 7 symbols							Block 2: 7 symbols						
53	124	32	70	107	50	15	41	20	13	97	76	88	16
53	124	32	70	107	50	15	57	36	10	97	53	91	14

5.5.3 Nybbles in the header block

The header block is able to encode $SF - 2$ nybbles, which is five nybbles in SF7. In the explicit header mode, the five nybbles are all used to encode the information of the frame, so that there is not any nybble available for the payload data in the header block. However, the number of such nybbles is different from zero when SF is larger than 7. In order to observe the changes from the payload nybbles encoded into the header block, we first explore a frame encoded by SF7 and CR3, in the implicit header mode, without CRC. Table 5.11 shows a frame encoding nine bytes `hello\0\0\0\0` as the payload data in implicit header mode. Indeed, in the implicit header mode, the header block is able to encode 2.5 bytes, while the two remaining blocks are still able to encode 3.5 bytes each (that is, 9.5 bytes in total). There is still half a byte (that is, one nybble) that needs to be padded, which brings different symbols of the entire frame except in the header block. We do not know the reason why the padding changes all symbol values.

Similar differences also occur for a frame encoded by SF8 in the explicit header mode. Such frame with one header block and two symbol blocks is able to encode 8.5 bytes (0.5 ($= (SF - 2 - 5)/2$) bytes in the header block, and four bytes in each block). The difference of symbols from the frame encoding `hello\0\0\0` are shown in Table 5.12. We can observe that less symbols are impacted.

However, when there is only a header block and another block encoded by SF8 in implicit mode, the frame is able to encode three ($= (SF - 2)/2$) bytes in the header, and four bytes in the block. In this case,

Table 5.11: Validation test of our simulator with implicit header mode and padding.

Symbol values	Header block: 8 symbols							
Real hardware	40	112	60	68	108	48	12	0
Our simulator	40	112	60	68	108	48	12	0

Block 1: 7 symbols							Block 2: 7 symbols						
22	39	108	106	81	85	31	44	45	42	21	99	11	87
41	56	99	109	82	84	96	103	79	97	61	23	39	43

Table 5.12: Validation test of our simulator with explicit header mode and padding.

Symbol values	Header block: 8 symbols							
Real hardware	252	112	56	112	12	224	108	52
Our simulator	252	112	56	112	12	224	108	52

Block 1: 7 symbols							Block 2: 7 symbols						
154	126	48	172	106	202	12	61	123	9	211	209	77	177
154	126	48	172	106	202	12	194	123	9	211	222	77	178

we program the end-device and our simulator to generate a frame encoding a seven bytes payload equal to `hello\0\0`. Table 5.13 shows that our simulator is able to reproduce 100% of the symbols from the same frame encoded by the SX1276 end-device.

Table 5.13: Validation test of our simulator with implicit header mode and without padding.

Symbol values	Header block: 8 symbols								Block 1: 7 symbols						
Real hardware	212	240	124	152	208	100	28	0	20	35	78	153	169	82	98
Our simulator	212	240	124	152	208	100	28	0	20	35	78	153	169	82	98

Thus, in order to avoid the padding, we show next the frames encoded by SF9 (see Table 5.14) and SF11 (see Table 5.15) in the explicit header mode. In this mode, $SF-2-5$ is an integral multiple of 2. There are two symbol blocks after the header block. For SF9, we encode a 10-byte payload equal to `hello,nii\0`, with CR3, without CRC, in the explicit header mode. For SF9, the header block is able to encode one byte (two nybbles) from the payload, while the two following blocks encode 4.5 bytes in each. For SF11, we encode a 13-byte payload equal to `hello,world\0`, with CR3, without CRC, in the explicit header mode. The header block encodes two bytes, and 5.5 bytes in each following block. Both results show that there is no difference between the frames generated by our simulator and the real LoRa end-device. Note that the frames encoded with SF11 do not use the low data rate optimization.

Table 5.14: Validation test of our simulator without padding, with SF9.

Symbol values	Header block: 8 symbols							
Real hardware	380	116	56	192	104	12	192	108
Our simulator	380	116	56	192	104	12	192	108

Block 1: 7 symbols							Block 2: 7 symbols						
333	448	103	169	394	314	14	256	83	292	381	339	510	406
333	448	103	169	394	314	14	256	83	292	381	339	510	406

Table 5.15: Validation test of our simulator without padding, with SF11.

Symbol values	Header block: 8 symbols							
Real hardware	1292	396	2008	832	104	1996	240	444
Our simulator	1292	396	2008	832	104	1996	240	444

Block 1: 7 symbols							Block 2: 7 symbols						
595	527	89	874	1421	1294	1012	1572	1132	858	665	1478	139	1559
595	527	89	874	1421	1294	1012	1572	1132	858	665	1478	139	1559

As for the frames encoded with SF10 (see Table 5.16) and SF12 (see Table 5.17)), we further explore them in the implicit header mode (in which $SF-2$ is an integral multiple of 2, representing an integer number of bytes), and with only one symbol block. We used CR3, and no CRC. For each SF, there is only one header block and another block. For SF10, the 9-byte payload is `hello,ni\0`. For SF12, the 12-byte payload is `hello,nii\0\0`. The results show no difference between the frames generated by our simulator and the real LoRa end-device. Note that the frames encoded with SF12 do not use the low data rate optimization.

Table 5.16: Validation test of our simulator without padding, with SF10.

Symbol values	Header block: 8 symbols							
Real hardware	808	1008	384	612	848	404	124	1020
Our simulator	808	1008	384	612	848	404	124	1020

Block 1: 7 symbols						
506	864	474	725	890	71	281
506	864	474	725	890	71	281

Table 5.17: Validation test of our simulator without padding, with SF12.

Symbol values	Header block: 8 symbols							
Real hardware	2856	3084	1660	2404	3408	1620	480	4080
Our simulator	2856	3084	1660	2404	3408	1620	480	4080

Block 1: 7 symbols						
1534	1191	2486	2821	2905	66	1746
1534	1191	2486	2821	2905	66	1746

5.5.4 Frames with low data rate

Since the header block is encoded in the low data rate mode, we will show at last the frames encoded with SF11 and SF12, with the low data rate optimization enabled.

The library [19] provided by Heltec does not include the low data rate optimization. However, we added such functionality in [60] based on the data-sheet of SX1276 [46], which allows to explicitly enable or disable the low data rate optimization. In the low data rate mode, each block after the header block is able to encode $(SF-2)/2$ bytes, instead of $SF/2$ bytes. Therefore, we reuse the payload data from SF9 for SF11, and from SF10 for SF12. The payload for SF11 is `hello,nii\0`, and is `hello,nii\0\0` for SF12. Table 5.18 and Table 5.19 show the results for SF11 and SF12, respectively. Our simulator is able to reproduce the same frames as those generated by the SX1276 chip in the LoRa end-device manufactured by Heltec.

Table 5.18: Validation test of our simulator without padding, with SF11 and the low data rate mode.

Symbol values	Header block: 8 symbols							
Real hardware	1392	436	36	832	104	48	140	124
Our simulator	1392	436	36	832	104	48	140	124

Block 1: 7 symbols							Block 2: 7 symbols						
332	60	356	852	1460	1284	1004	764	1452	1828	1408	1708	544	616
332	60	356	852	1460	1284	1004	764	1452	1828	1408	1708	544	616

5.5.5 Summary of the validation tests

Although our simulator is not always able to generate consistent frames, particularly when padding is involved, it can still generate consistent frames in a large number of conditions. These validations demonstrate that our physical layer simulator can simulate realistic LoRa end-devices and decode frames from commodity nodes, *i.e.*, the end-devices in LoRa. We believe that the implementation issues described here may assist

Table 5.19: Validation test of our simulator without padding, with SF12 and the low data rate mode.

Symbol values	Header block: 8 symbols							
Real hardware	808	1008	384	612	848	404	124	1020
Our simulator	808	1008	384	612	848	404	124	1020

Block 1: 7 symbols						
506	864	474	725	890	71	281
506	864	474	725	890	71	281

researchers in reverse-engineering other parts of the LoRa coding algorithm.

Chapter 6

Simulation results

In this chapter, we present the simulation results of all our contributions.

In Section 6.1, we show the gain of the uncertainties resolution using LoRa coding mechanism (described in Chapter 3 and in [61]), on several existing collision resolution algorithms. In Section 6.2, we study the impact of several advanced parameters on existing collision resolution algorithms, using our GNU Radio simulator (described in Chapter 5 and in [63]). These parameters are the duty-cycle and the SNR, and are often ignored in the description of existing papers. In Section 6.3, we study the performance of our main contribution SF-DS (described in Chapter 4 and in [62, 64]).

6.1 Performance evaluation of the uncertainties resolution using LoRa coding mechanism

In this section, we evaluate the gain of our proposed uncertainties resolution algorithm using LoRa coding, by comparing it to several existing collision resolution algorithms, through simulation. More precisely, we compare the performance of several existing collision resolution algorithms with and without our proposed algorithm.

First, we present the parameter settings used in our simulations. Then, we show the performance of each algorithm.

6.1.1 Parameter settings

We used the NS-3 simulator presented in Section 5.4. The network topology consists of an NS, a single gateway, and several end-devices uniformly distributed within one kilometer around the gateway, as shown in Figure 6.1. The simulation results are averaged over 100 repetitions.

Regional settings and channel model. We considered the European regional setting of LoRaWAN for all protocols. The bandwidth is set to 125 kHz. All the devices use the same channel at 868.1 MHz in order to simulate a heavy traffic, *i.e.*, with a large probability of colliding frames. Our NS-3 simulator assumes that the channel does not produce frame errors or symbol errors, thus the symbols are correctly demodulated by all algorithms (at least when there is no collision). In other words, we consider a perfect channel model, without noise. We assumed a log-distance path loss model along with a time delay model configured as the propagation model in the simulator. The power loss can be represented as follows:

$$Loss = L_0 + 10n \log_{10} \frac{d}{d_0}, \tag{6.1}$$

where $n = 3.76$, $L_0 = 7.7$ and $d_0 = 1$.

Gateways. The gateway is simulated as an SX1301 hardware [47] with eight demodulation paths, which allows decoding up to eight frames simultaneously. All the demodulation paths are set to decode the frames with the same SF.

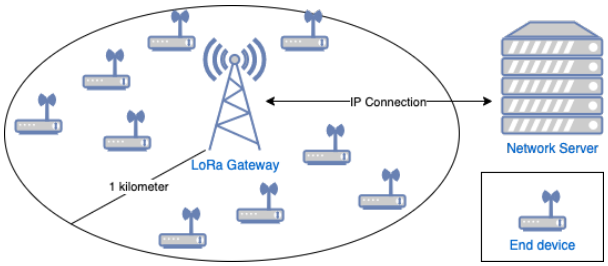


Figure 6.1: Architecture of the simulated LoRaWAN network with several end-devices, one gateway and one NS.

End-devices and traffic. The simulation lasts for one hour. We varied the number of end-devices from 400 to 2000 for SF7, and from 200 to 1000 for SF12. The CR for LoRa frame encoding is set to 1, which results in the shortest frame length, but in the lowest performance improvement for our algorithm. Each end-device sends only one frame with 10 bytes of random payload, with a random delay from the beginning of the simulation. After applying LoRa coding, each frame contains 28 symbols for SF7, and 38 symbols for SF12. The frames do not require an acknowledgement.

Settings for our algorithm. We do not limit the decoding time nor the number of possible frames produced by the uncertainties for our proposed algorithm.

Settings for the existing collision resolution algorithms. The collision resolution algorithms are only applied at the gateway. We implemented CHOIR, SIC, and GS-MAC. Their internal parameters in the

simulations are presented as follows:

- In CHOIR, end-devices are configured with uniformly distributed frequency offsets to simulate hardware imperfection. These frequency offsets are fixed throughout the simulation. We apply our algorithm when the collided frames have a relative frequency offset smaller than the given resolution, which is one tenth of a symbol.
- In SIC, we consider that a large difference in signal power is sufficient to extract the frame with the strongest signal. In [5], the authors have shown that the error rate dramatically increases when three or four frames are received simultaneously with similar power. So, we applied our algorithm only when there are several strong frames, all received with a similar power, or when there is a single strong frame and a group of weaker frames, all these weaker frames being received with similar power. The thresholds to define the similarities are set to -6 dB for SF7 and to -20 dB for SF12, which are coherent with the thresholds proposed in [51].
- In GS-MAC, the number of virtual sub-slots is set to 4, as described in [13]. The end-devices randomly choose a virtual sub-slot for the transmission, following a uniform distribution.

Our objective here is not to compare CHOIR, SIC, and GS-MAC, as each collision resolution algorithm benefits from different deployment settings. Instead, we focus on the benefits of our uncertainties resolution algorithm when it is applied to any collision resolution algorithm.

6.1.2 Frame error rate

In our simulations, we did not consider the SER, as the channel model in the simulator does not produce errors. Thus, we evaluate the decoding capabilities in terms of FER. In the simulation, frame errors come either from collisions that the algorithms could not resolve, or by exceeding the limited number of demodulation paths of the gateways (which is a hardware constraint of the SX1301).

Figure 6.2 shows the average FER for CHOIR, with and without our proposed algorithm, as a function of the number of end-devices in the network. SF7 is shown on the left, and SF12 is shown on the right. In CHOIR, the indistinguishable frames in collision are those which have the same tiny frequency offsets, with a granularity of one tenth of a symbol. The gain of our proposed algorithm originates from the recovering of these frames. The results show that our proposed algorithm improves the decoding capacities for unconfirmed traffic by 13% with SF7, and by 11% with SF12.

Figure 6.3 shows the average FER for SIC, with and without our proposed algorithm, as a function of the number of end-devices in the network. SF7 is shown on the left, and SF12 is shown on the right. According to the hypothesis from [5] and to the propagation model, the uncertainties are from the frames sent by end-devices at similar distances to the gateway. The gain of our proposed algorithm comes from the

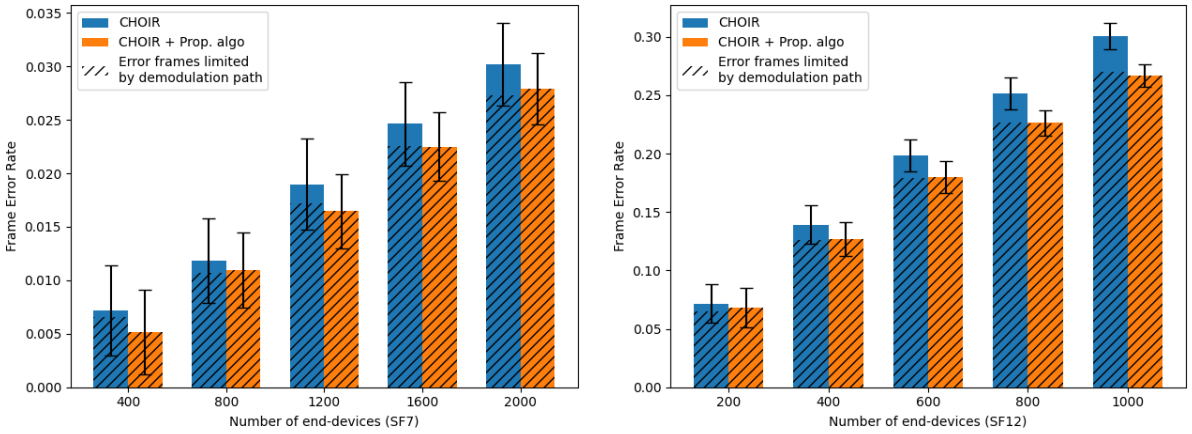


Figure 6.2: Simulation results of CHOIR before and after applying our frame recovering algorithm. SF7 is shown on the left, and SF12 is shown on the right.

recovering of the frames with the strongest and the second strongest power levels. The results show that our proposed algorithm improves the decoding capabilities for unconfirmed traffic by 35% with SF7 and by 47% with SF12.

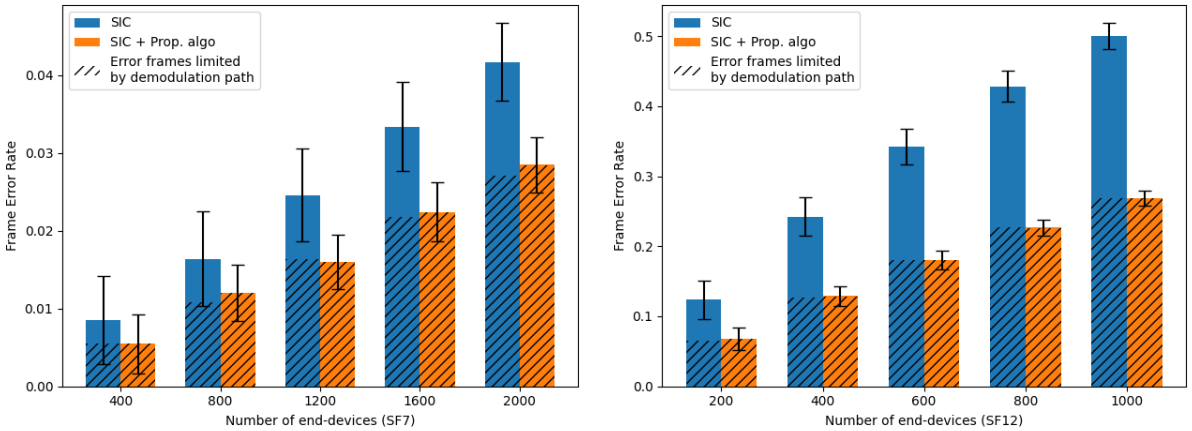


Figure 6.3: Simulation results of SIC before and after applying our frame recovering algorithm. SF7 is shown on the left, and SF12 is shown on the right.

Figure 6.4 shows the average FER for GS-MAC, with and without our proposed algorithm, as a function of the number of end-devices in the network. SF7 is shown on the left, and SF12 is shown on the right. The recovering of frames sent in the same virtual sub-slot and the uncertainties caused by the repeated symbols gives a significant gain to GS-MAC, when our proposed algorithm is used. The results show that our proposed algorithm improves the decoding capabilities for unconfirmed traffic by 28% with SF7, and by 19% with SF12.

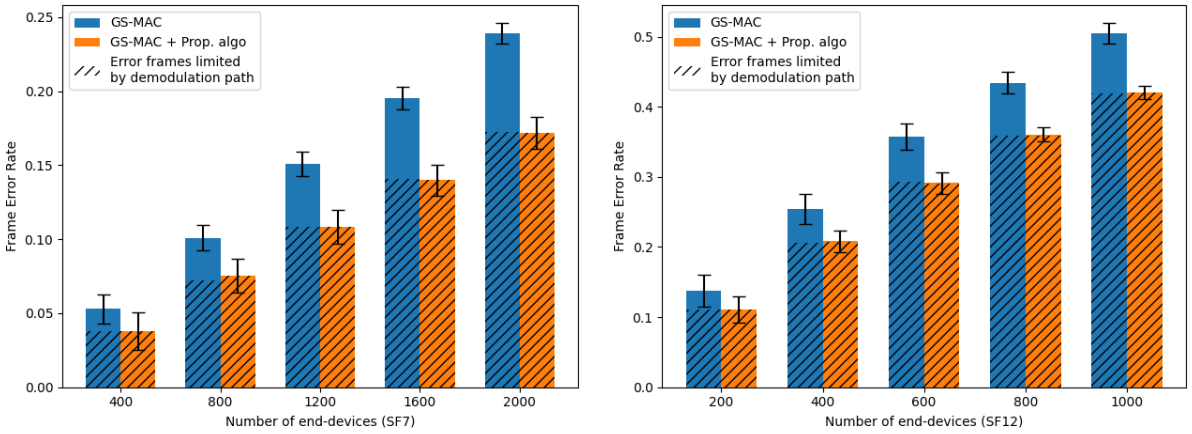


Figure 6.4: Simulation results of GS-MAC before and after applying our frame recovering algorithm. SF7 is shown on the left, and SF12 is shown on the right.

Overall, when applying our algorithm, the remaining errors on frames are primarily due to the limited number of demodulation paths, as shown by the dashed areas in Figures 6.2, 6.3 and 6.4. Our algorithm reduces most of the uncertainties for SF7 and SF12, which brings fewer frame errors. With SF7, compared to the original CHOIR, SIC, and GS-MAC, the gain with our proposed algorithm can reach, respectively, 13%, 35%, and 28%. With SF12, the gain for each algorithm reaches up to 11%, 47%, and 19%, respectively.

As our simulation scenario deals with unconfirmed traffic, we did not study the results in terms of delay or energy efficiency, as they are more impacted by the retransmissions of confirmed traffic than by the decoding capabilities of the protocols. In the case of confirmed traffic, reducing the FER also reduces the number of retransmissions, which should also yield to a significant decrease in terms of both the delay and the energy consumption.

However, note that our algorithm requires the symbols to be correct, which is the case in this simulation setup thanks to the perfect channel model. In a real deployment, the proposed algorithm would likely fail when the symbol demodulation is erroneous. In order to cope with this in SF-DS, we added a tolerance in case of failed decoding, which brings some extra uncertainties. However, these extra uncertainties are likely to include the correct symbols. We will show the results after applying our algorithm in a more realistic channel model later, in Section 6.3.

6.2 Performance evaluation of existing collision resolution algorithms based on GNU Radio

In this section, we evaluate existing collision resolution algorithms under our GNU Radio simulator, which finely simulates the physical layer of LoRa. By integrating several existing collision resolution algorithms in this simulator, we are able to compare these algorithms together in a single environment. Moreover, our simulator allows to study the impact of useful network parameters that are sometimes missing from the original papers: this is typically the case for the SNR.

We implemented the following algorithms: legacy LoRa, CHOIR, FTrack and OCT. In the simulation, the signals are generated from several node abstractions, as described in Subsection 5.2.3. We aim at comparing the ideal performances of the implemented collision resolution algorithms, without the impact of the preamble detection. Note that to remove the preamble detection, we considered that the superposed signals carry the required information indicating the beginning of the payload, along with the sample data.

In addition, some early articles presenting collision resolution algorithms, such as [59], do not describe into great details the actual physical demodulation. Moreover, the lack of available source code sometimes prevents us from evaluating them precisely. Therefore, we evaluate their throughput without considering SER, and we expect the symbol errors to be partially corrected using the channel coding, or leaving a small amount of bit errors in the decoded data.

In the following, we focus on two factors: the duty-cycle (which is a parameter of the MAC layer) and the SNR (which is a parameter of the physical layer).

6.2.1 Impact of the duty-cycle

We first evaluated the performance of different decoding algorithms with different duty-cycle constraints, but without noise (that is, in ideal conditions). The duty-cycle limits the intervals between several transmissions of a single end-device, which results into different traffic loads.

The network consists of one gateway and a varying numbers of end-devices. All the end-devices have the same relative distance to the gateway, which is the worst-case for the algorithms based on power level. For CHOIR, the window size is set to 10 times the FFT window of legacy LoRa. For FTrack, we set the size of the sliding window to the entire symbol duration, and n_{step} to 8 (which produces $M/8$ FFT results for one SD). For OCT, variable window sizes are used, depending on the relative delays between the frames.

Each end-device periodically sends frames of 20 random bytes, encoded with $CR = 1$. The time on air of the frames is 57.3 ms for SF7 and 1.34 s for SF12. The duty-cycle is either 1% or 10%. In other words, after each transmission, an end-device needs to wait for either 99 times, or for 9 times, the time on air of the frame to initiate another transmission. In addition to this implicit waiting time due to the duty-cycle,

we add a random waiting time between each transmission in order to desynchronize the end-devices. For 1% duty-cycle, the maximum waiting time is set to 100 times the time on air, which is equal to 5.7 s for SF7, and 135 s for SF12. For 10% duty-cycle, the maximum waiting time is set to 10 times the time on air, which is equal to 0.57 s for SF7, and 13.5 s for SF12.

We request each end-device to send five transmissions. In order to ensure collisions, we generated a heavy load by using a relatively small simulation time of 57 s (with SF7) or 1350 s (with SF12) for 1% duty-cycle, and of 5.7 s (with SF7) and 135 s (with SF12) for 10% duty-cycle. We vary the number of end-devices from 5 to 100, with a step of 5. The results are averaged over 20 simulations.

Figure 6.5 shows the throughput based on the number of bits of correct frames with SF7 (on the left) and SF12 (on the right). The results are displayed for four algorithms and two duty-cycles (1% or 10%). Our results show that the setup with 1% duty-cycle does not generate enough collisions: collision resolution algorithms do not bring much improvement to the performance of legacy LoRa. Note that legacy LoRa does not even reach its saturation with 100 end-devices at 1% duty-cycle. For 10% duty-cycle, the saturation is reached at about 30 end-devices for legacy LoRa. Note that both SF7 and SF12 have the same saturation tendency because the duty-cycle and the intervals between frames are set according to the time on air of the frames. It can be observed that FTrack performs the best in all setups, which is not consistent with the results from [48]. This comes from the fact that [48] evaluates FTrack with collided frames of different power levels, while the original FTrack paper [59] mentions that they only used FTrack to decode frames with similar power levels.

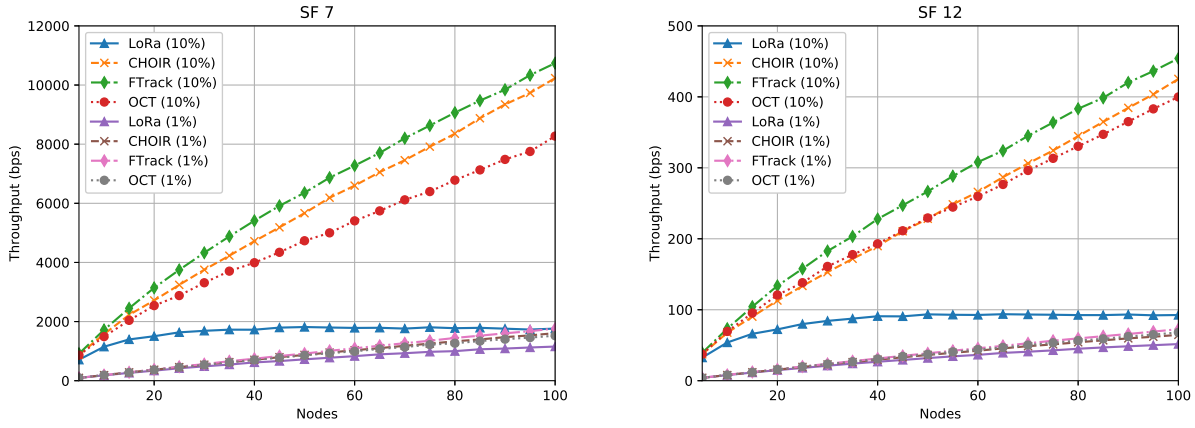


Figure 6.5: Study of the throughput as a function of the number of end-devices, with 1% or 10% duty-cycle, and with SF7 (on the left) and SF12 (on the right).

6.2.2 Impact of the SNR

We then evaluated the performance of different decoding algorithms with a fixed duty-cycle of 10% (in order to generate a high load), but with different SNRs. We deployed 50 end-devices in the network, which ensures that many collisions will occur. Our target is to compare the performance of decoding algorithms under realistic conditions, including when the signals are weaker than the noise (i.e., $SNR < 0$). Note that we also studied the influence of the sliding window size in FTrack: we considered a full window (that is, $N_{win} = 2^{SF}$) and a half window (that is, $N_{win} = 2^{SF}/2$). It is important to note that the impact of the SNR on the decoding algorithms was only known in the literature for legacy LoRa and for the CIC algorithm, but not for CHOIR, FTrack and OCT.

The end-devices are randomly deployed in a ring between 500 m and 1000 m around the gateway, as shown in Fig. 5.4, but with different numbers of end-devices in the network. The relative receiving power levels are calculated using a log-distance loss model, as described in Equation 6.1 with the same parameters ($n = 3.76$, $L_0 = 7.7$ and $d_0 = 1$). We varied the relative strength of the noise so that the SNR varies from -10 dB to 20 dB.

Figure 6.6 shows the throughput with SF7 (on the left) or SF12 (on the right), for four algorithms, 50 end-devices, and 10% duty-cycle. The SNR varies from -10 dB to 20 dB. Note that -10 dB is already below the rejection limit of SF7 , while 20 dB is a fairly good SNR. Our results show that FTrack with full FFT window performs the best with SF7, while OCT reaches the best performance with SF12. The performance of legacy LoRa appears to be slightly impacted by the SNR. The performance of CHOIR is influenced by the noise, as it can disturb the detection of the tiny frequency offsets. With SF7, throughput degradation can be observed for negative SNRs, for CHOIR, for FTrack with full window, and for OCT. With SF12, such degradation appears for CHOIR and FTrack for $SNR = 10\ dB$ and below, and for OCT for $SNR = 5\ dB$ and below. Finally, note that FTrack with half window has relatively poor performance for all noise levels. This is because the incomplete FFT window brings more uncertainties and errors, and FTrack does not contain a solution to reduce these uncertainties. Thus, FTrack with half window results in more bit errors, and thus lower throughput.

6.3 Performance evaluation of SF-DS

We implemented SF-DS in our physical layer simulator based on GNU Radio. This section presents the core results of the performance evaluation of SF-DS.

We first evaluated SF-DS in a simple scenario to show the SER and the theoretical throughput of SF-DS compared to legacy LoRa. Then, we evaluated SF-DS in a more complex scenario with periodic transmissions, and we compare SF-DS with the other algorithms from the literature. Finally, we study the impact of the

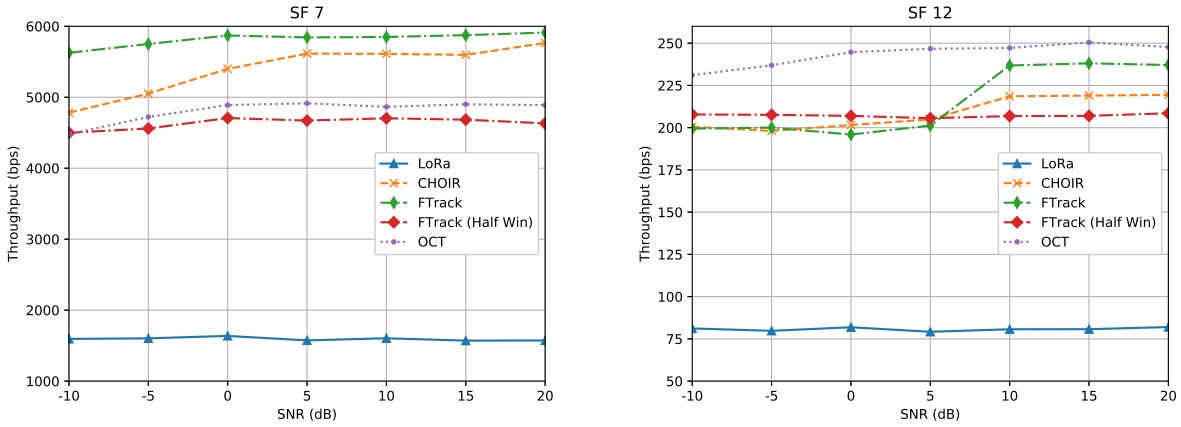


Figure 6.6: Study of the throughput as a function of the SNR, for 50 end-devices and a duty-cycle of 10%, and with SF7 (on the left) and SF12 (on the right).

bandwidth, the SNR, and the number of virtual sub-slots, still in this complex scenario, in order to show the advantages and the shortcomings of SF-DS.

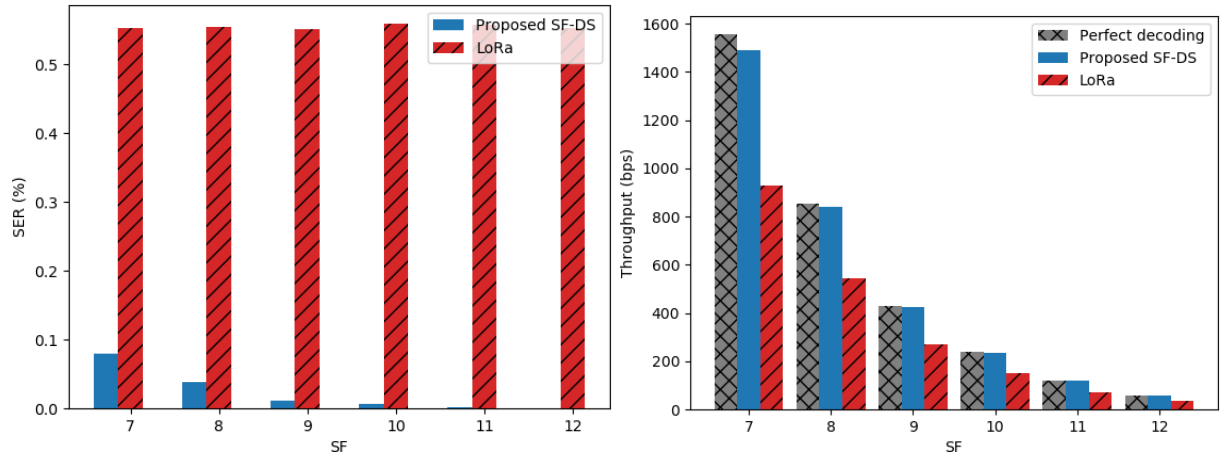
In both simple and complex scenarios, each frame encodes 22 bytes of randomly generated data, with $CR = 1$, $BW = 125$ kHz and SF varying from 7 to 12. The low data rate mode is enabled for SF11 and SF12. These configurations result into a payload of 43, 38, 38, 33, 33 and 33 symbols for SF7 to SF12, respectively. By default, SF-DS uses 4 virtual sub-slots during the demodulation. The simulation results are averaged over 100 repetitions.

6.3.1 Simple scenario: two colliding frames with the same power

In the simple scenario, there are only two frames received with the same power, and they are always in collision. The second frame has a random delay between 0 and the entire duration of the first frame. There is no noise in this scenario.

Figure 6.7 shows the SER and the throughput of LoRa, SF-DS, and a perfect decoding scheme, as a function of the SF. As the second frame arrives after the beginning of the first frame, LoRa can often demodulate the beginning of the first frame. However, when the preamble or the payload of the second frame collides with the first frame, the capture effect of LoRa cannot be applied due to the similarity of the power of the two frames, and LoRa is unable to decode the symbols of any frame. This leads to a SER of more than 50% for LoRa, and a maximum throughput of about 900 bps for SF7. With SF-DS, symbol errors arise only when there are two superposed symbols with similar values (namely, ± 5). Indeed, in this case, it is hard to differentiate the peaks of the FFT due to the imperfection of the FFT. Since each symbol encodes SF bits, the probability of having two superposed symbols with similar values decreases as SF increases,

which quickly reduces the SER. In terms of throughput, it can be seen that SF-DS is close to the maximum throughput obtained with a perfect collision resolution scheme.



(a) SER (b) Throughput

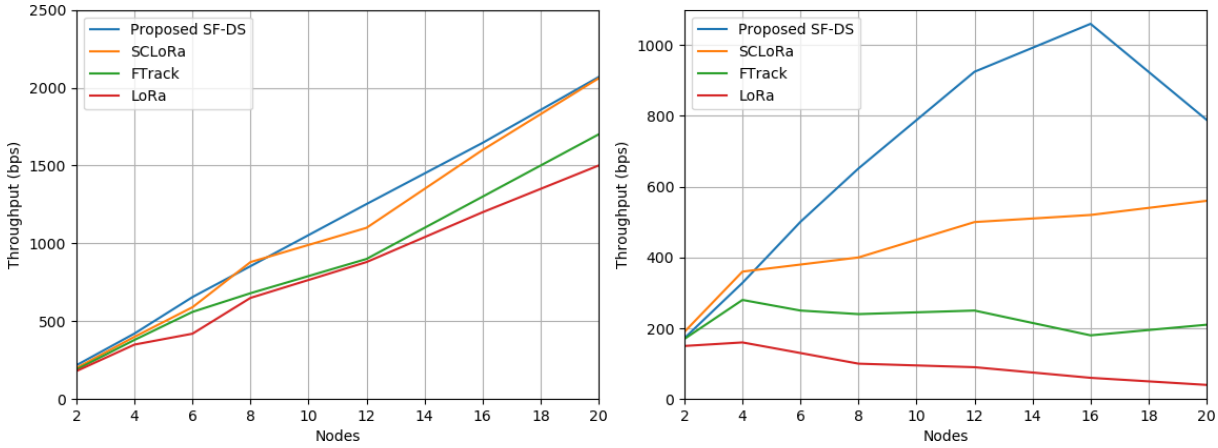
Figure 6.7: When only two frames collide, SF-DS has a very low SER. This yields to a throughput which is close to the maximum throughput achievable by a perfect decoding scheme.

6.3.2 Complex scenario: periodic transmissions of several end-devices

We then focus on a more realistic scenario. We used the same setup as [22], in order to stay consistent in the comparison. There are between 2 and 20 end-devices. All transmissions are received with a similar power. The SNR is set to 20 dB. Before sending a frame, each end-device chooses a random delay varying between 1000 ms and 2200 ms. The end-devices keep sending until the end of the simulation. The duration of each frame is about 103 ms for SF8, and 1480 ms for SF12. Note that the risk of collision is higher with SF12. We assumed that there is no duty-cycle limitations in order to simulate a heavy traffic. Again, SF-DS uses 4 virtual sub-slots during the demodulation.

Figure 6.8 shows the results of the simulation for SF8 and SF12, and for four protocols: LoRa, SCLoRa, FTrack and SF-DS. To stay consistent, we used the same setup of the simulations in [22]. The results for SCLoRa and FTrack are taken directly from [22], as the implementation of SCLoRa is not available. For SF8, the network is not yet saturated because there are few collisions. Thus, the performance of SF-DS is similar to the one of SCLoRa, which is currently the best known collision resolution protocol for LoRa. For SF12, SF-DS is able to achieve a much higher throughput than SCLoRa: the gain is 103% at 16 end-devices. This huge gain comes from three main factors: (i) the virtual sub-slots which helped to identify individual frequencies even when symbols were slightly desynchronized, (ii) the use of a tolerance of ± 5 values when identifying imperfect FFT peaks, and (iii) the leverage of redundancy bits from the Hamming code further

improved symbol decoding through the use of our mechanism proposed in Chapter 4.



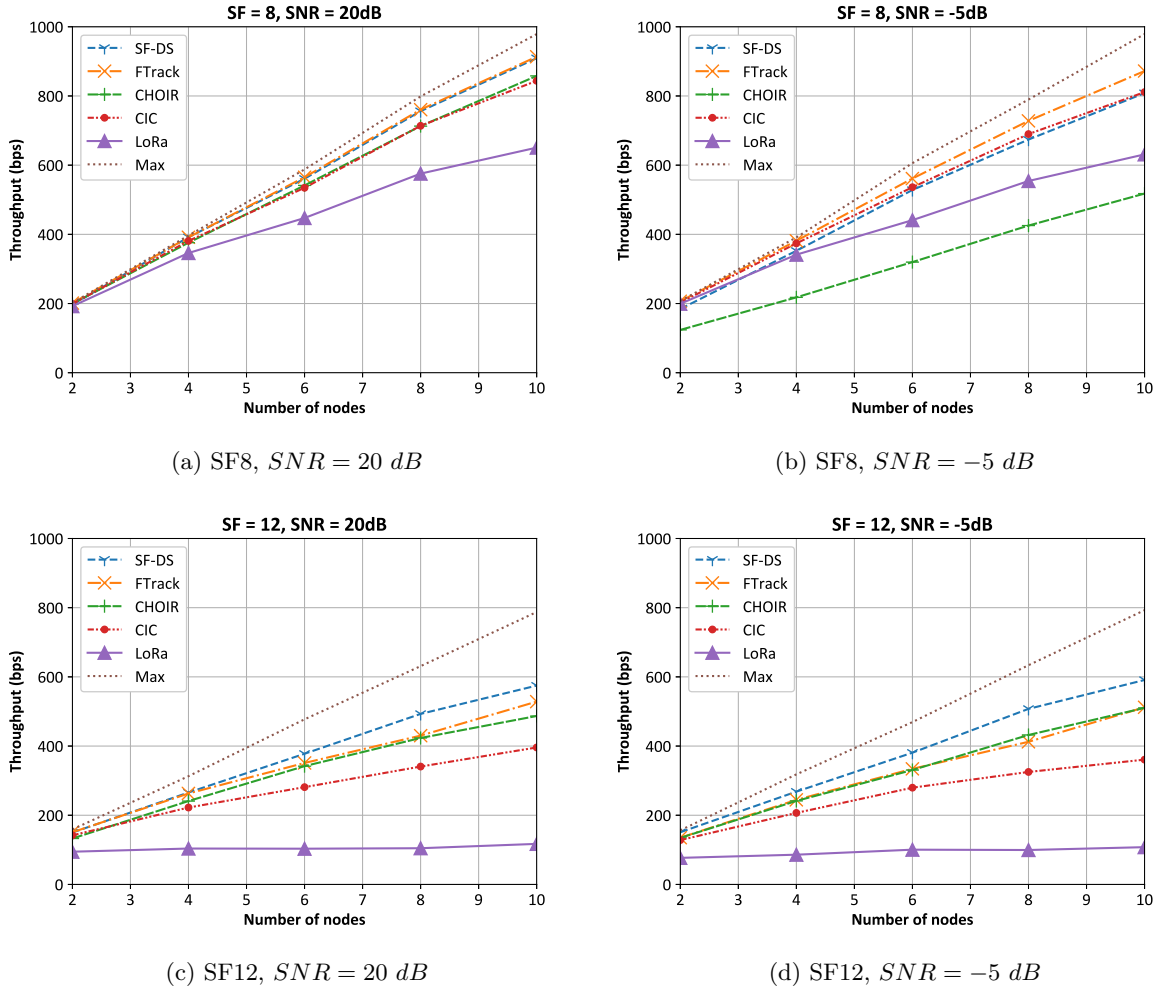
(a) SF8 (b) SF12

Figure 6.8: Our proposed SF-DS has slightly better performance than SCLoRa for SF8, but has much better performance than all algorithms for SF12, especially when the number of end-devices is large.

We performed a comparison of SF-DS with another set of algorithms, based on the simulator described in Subsection 5.2.4. This allowed us to include CIC from [48] in our simulations. For this new set of simulations, we generate and store superposed signals, in order to apply each algorithm to the same signals, for consistency and reproducibility. Each decoding algorithm operates on the same sample files, without preamble detection. We modified the implementation of CIC, which is available online, in order to remove its preamble detection and avoid the impact of non-detected frames. We used SF8 and SF12, and we varied the SNR between high (20 dB) and low (-5 dB). During each simulation, we deploy two to ten end-devices uniformly around a gateway, and the end-devices periodically transmit packets with random intervals between 1000 and 2200 ms. These packets consist of a random payload of 22 bytes and, after encoding with $CR = 1$, each frame contains 38 payload symbols for SF8 and 28 payload symbols for SF12.

Figure 6.9 shows the network throughput. Note that the plot labelled Max represents the ideal throughput if all of the transmitted frames were decoded without error. When SF8 is used, the network is not yet at its maximum capacity with ten end-devices. Although some frames are lost due to collisions, adding more end-devices increases the network throughput. In the high SNR scenario, all collision decoding algorithms perform better than basic LoRa. However, in the low SNR scenario, the performance of CHOIR is worse than LoRa because the small frequency offsets are masked by noise and cannot be detected by the gateway. Under low SNR, SF-DS experiences a slight decrease in performance as the noise may cause incomplete samples and padding in each FFT, shifting the FFT bins and reducing the accuracy of the FFT peak detection. It is worth noting that all collision resolution algorithms using FFTs shorter than the symbol length suffer from spectrum leakage, which affects the accuracy of the detection of the FFT peaks. When using SF12,

LoRa maintains a relatively constant throughput as the number of end-devices increases due to its ability to capture only one frame at a time. However, other collision resolution algorithms perform similarly to SF8 and do not exhibit a significant decrease in throughput. The longer symbol length of SF12 (typically 16 times longer than with SF8) increases the power levels on FFT bins, which helps to distinguish LoRa peaks from noise. It is noteworthy that SF-DS achieves the highest performance for SF12 in both high and low SNR scenarios. These results are coherent with the results shown in both Figure 6.6 and Figure 6.8.



(a) SF8, $SNR = 20\ dB$

(b) SF8, $SNR = -5\ dB$

(c) SF12, $SNR = 20\ dB$

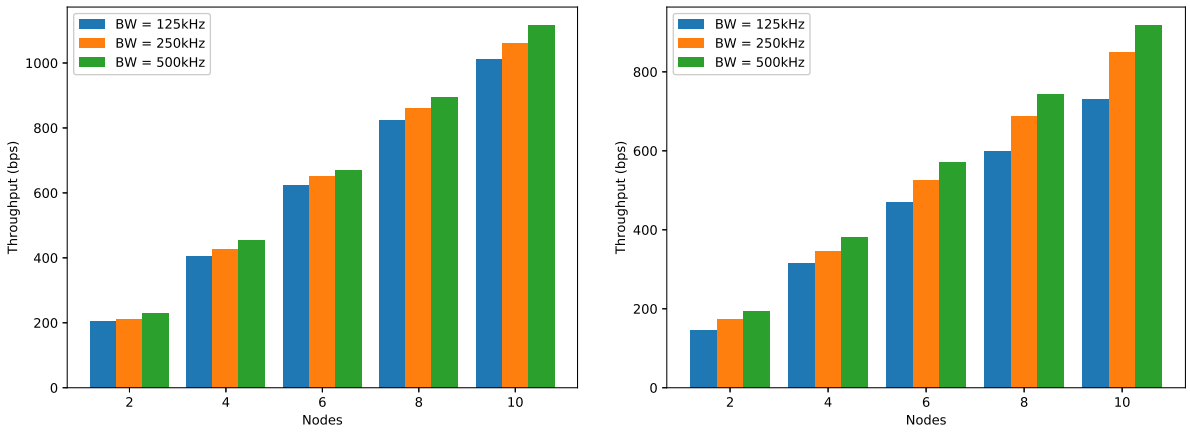
(d) SF12, $SNR = -5\ dB$

Figure 6.9: Throughput of decoding algorithm under different SNRs. The sending interval saturates the network with SF12. The SNR also changes the relative performance among the algorithms.

6.3.3 Impact of the bandwidth

By referring to Equation 2.1 and Equation 2.5, we can observe that the characteristics of the LoRa chirp remain unchanged when the bandwidth varies, and only the symbol duration is affected. As the bandwidth increases, the duration of symbols and frames decreases, resulting in a lower number of collisions. This makes the frames easier to decode when the bandwidth is large. However, the generated signal occupies more frequency resources.

Figure 6.10 shows our simulation results for SF-DS using three bandwidths (125 kHz, 250 kHz and 500 kHz, which are usable for the commodity LoRa SX1276 hardware), and with the same settings as the complex scenario from Subsection 6.3.2. Note that the frame duration for 250 kHz is 50% of the frame duration for 125 kHz, and the frame duration for 500 kHz is 25% of the frame duration for 125 kHz. We can see that the throughput increases by increasing the bandwidth, because of a lower number of collisions, as well as collisions that are easier to resolve. The relative gain of SF-DS when increasing the bandwidth is larger with SF12 than with SF8. The reason is that the shorter duration of the frames with SF8 alleviates the traffic load, and reduces the opportunity of collisions.



(a) SF8

(b) SF12

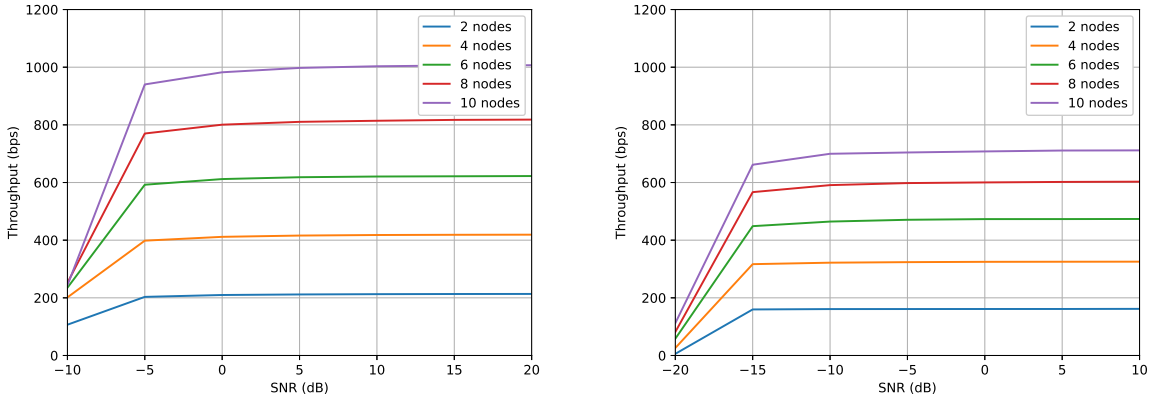
Figure 6.10: Throughput using SF-DS with different bandwidths. SF-DS remains stable over all bandwidths. The throughput is only impacted by the frame duration.

6.3.4 Impact of the SNR

As described in Subsection 2.1.2, a constant frequency over an entire SD contributes to a single FFT peak, which helps make the peak appear stronger than the noise. However, our proposition shortens the length of the samples used in FFT by dividing them into virtual sub-slots, which reduces the decoding performance when the SNR is low. In order to study the effect of the SNR, we generated the signals of superposed random

LoRa frames using the parameters from the complex scenario described in Subsection 6.3.2. Then, we added to the superposed signals an AWGN according to the desired SNR.

Figure 6.11 shows the results with SF8 and SF12, for two to ten end-devices. The SNR varies from -10 dB to 20 dB for SF8, and from -20 dB to 10 dB for SF12, with 5 dB as a step. With SF8, we can see that the throughput increases quickly when the SNR increases from -10 dB to -5 dB, and then slowly increases when the SNR increases from -5 dB to 20 dB. With SF12, the throughput increases quickly when the SNR increases from -20 dB to -15 dB, and then remains relatively stable when the SNR increases from -15 dB to 10 dB. According to Table 2.1, the SNR thresholds of LoRa for SF8 and SF12 are -9 dB and -20 dB respectively. We can conclude that our SF-DS has similar anti-noise performance as LoRa. However, due to the virtual sub-slots, the performance of SF-DS starts to slightly decrease a few dB before reaching the SNR thresholds.



(a) $SF = 8$ (b) $SF = 12$

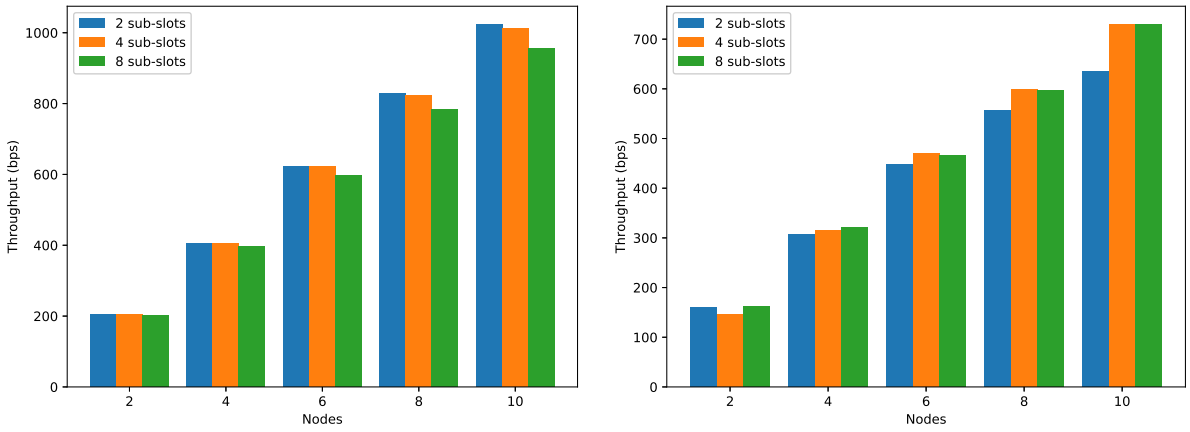
Figure 6.11: Throughput using SF-DS with different SNRs. The performance is similar to LoRa, despite of the slight degradation around the SNR thresholds.

6.3.5 Impact of the number of virtual sub-slots

The number of virtual sub-slots used in SF-DS has two effects on its decoding capabilities. First, it limits the number of frames that can be decoded without leveraging the LoRa coding proposed in [61]. Second, it affects the number of samples in each virtual sub-slot, as the total number of samples in SD is fixed by the sample rate. For instance, for SF7, the number of samples in SD is 128 (regardless of the bandwidth used): if the receiver uses four virtual sub-slots, then each virtual sub-slot contains 32 samples.

Figure 6.12 illustrates the throughput achieved by our proposed SF-DS decoder when using two, four, and eight virtual sub-slots. In the case of SF8, increasing the number of virtual sub-slots leads to a decrease

in throughput, as the total number of samples during each SD is limited to only 256 samples. However, with SF12, increasing the number of virtual sub-slots results in an improvement in the overall throughput when there are more frames in the network. This is because even with eight virtual sub-slots, the number of samples in each virtual sub-slot is 512 (which equals to $2^{SF}/8$) for SF12, which is still large enough for SF-DS to maintain a sufficiently high resolution.



(a) $SF = 8$ (b) $SF = 12$

Figure 6.12: Throughput using SF-DS with a varying numbers of virtual sub-slots. The throughput is mainly impacted by the number of samples in each virtual sub-slot with SF8, and by the number of usable virtual sub-slots for the collided frames with SF12.

Figure 6.13 shows the SER as a function of the number of virtual sub-slots, in the simple scenario described in 6.3.1, but with a varying number of virtual sub-slots equal to 2, 4 and 8. We can observe that the SER increases with the number of virtual sub-slots, which is caused by the imperfect FFT as described in Subsection 4.1.2. When the SF increases, the number of samples during a symbol also increases. Therefore, there are more samples in each virtual sub-slot, which helps to reduce the SER. With 8 virtual sub-slots, the SER is much worse than with 2 or 4 virtual sub-slots, while the performance using 2 virtual sub-slots and 4 virtual sub-slots becomes closer by increasing SF. However, recall that the number of virtual sub-slots in SF-DS affects the amount of uncertainties, which come mostly from the frames in the same virtual sub-slot. Less virtual sub-slots bring more uncertainties when there are superposed frames. Therefore, we believe that using 4 virtual sub-slots is a good trade-off. Indeed, the SER is similar when using 2 virtual sub-slots and 4 virtual sub-slots with SF12. The SER using 2 virtual sub-slots has a gain of around 30% compared to the SER using 4 virtual sub-slots with SF8, but the number of virtual sub-slots is reduced by half. By increasing the numbers of collided LoRa frames, the receiver is likely to decode more frames from the different sub-slots. Considering that there are at most ten end-devices during the simulations, 4 virtual

sub-slots seems a reasonable option. Although it is possible to set different numbers of virtual sub-slots for different numbers of end-devices, we keep using four sub-slots for the consistencies of the results.

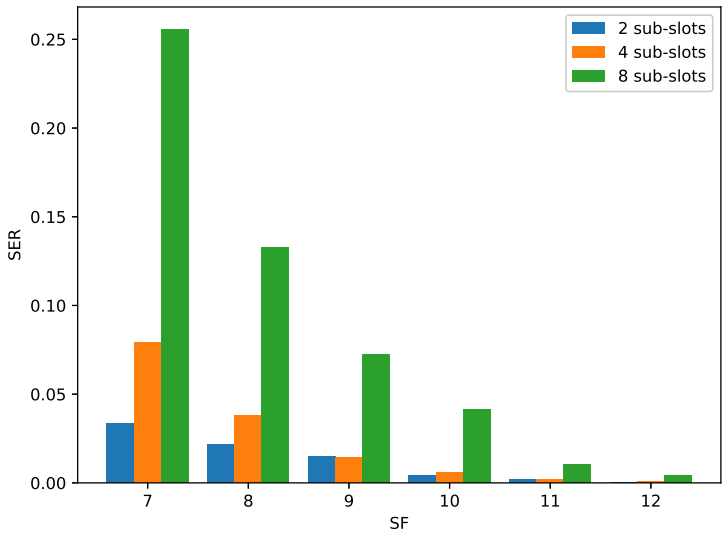


Figure 6.13: SER using SF-DS with a varying numbers of virtual sub-slots, in the simple scenario.

Chapter 7

Conclusion

LPWAN technologies are generally used in several applications, including environmental monitoring. Collisions in LoRa become more common when the number of active end-devices increases, which negatively impacts network performance. Note that collisions in LoRa further reduce the throughput which is already very small in LoRa. Therefore, some collision resolution algorithms have been proposed in order to decode collided frames. They exploit the characteristics of the signals of LoRa frames: they leverage the difference of power from the received frames (including the capture effect), they track frames based on their time offsets, or they combine both. However, the full description of the LoRa physical layer is not available, which hinders the design of such algorithms. Authors in [25, 38, 3, 53, 30] explored the LoRa physical layer and provided details on different parts, including the channel coding used in LoRa and the CSS modulation. These works make possible deeper and further research on collision resolution.

In this thesis, we proposed a new decoding scheme for collided LoRa frame, exploiting the features of LoRa frames, as well as an algorithm based on LoRa coding and designed to improve the performance of any collision resolution algorithm.

In this chapter, we first summarize our contributions from our four related publications [64, 61, 63, 62]. Then, we give a brief discussion about the perspectives based on our current work.

7.1 Contributions

In Chapter 2 of this thesis, we investigated the current existing collision resolution algorithms. We first provided a basic understanding of both the LoRa physical layer and the LoRaWAN standard. Then, we categorized the major approaches for decoding collided LoRa signals and explained their key principles and differences. This categorization was described in our overview article [64]. Additionally, the paper [64] discusses how each category can be integrated with MAC protocols. We also classified the scheduling and

MAC layer protocols compatible with LoRa. Finally, we discuss open research issues, including future challenges and the expansion towards more complex network topologies, energy efficiency, mobility issues, and satellite-terrestrial communications.

In Chapter 3, we discussed how uncertainties can occur in collision resolution algorithms, using the GS-MAC algorithm [13] as an example. While the causes of uncertainties might change from one collision resolution algorithm to another, they always result in several possibilities for a given symbol value. We proposed an algorithm to exploit the Hamming coding and the interleaving used in LoRa coding, in order to reduce (and, often, to remove) these uncertainties. This algorithm was presented in [61]. Note that this generic algorithm can be utilized for decoding colliding LoRa signals, as well as non-colliding LoRa signals. Thanks to this algorithm, additional colliding frames can be recovered, which can prevent some retransmissions and ultimately improve the overall network performance.

In Chapter 4, we proposed a novel decoding scheme called SF-DS inspired by GS-MAC. SF-DS brings the following improvements over GS-MAC: (i) SF-DS does not require the transmitters to be synchronized, and is thus able to operate with legacy LoRaWAN devices, (ii) SF-DS uses virtual sub-slots in order to regroup and track signals sent at different times, (iii) SF-DS identifies ambiguous FFT peaks and takes into account the uncertainty caused by the Hamming window of the FFT, and (iv) SF-DS includes the algorithm presented in [61], in order to improve the frame decoding rate. SF-DS was presented in [62].

In Chapter 5, we discuss our open-source simulator, which can be used to study existing collision resolution algorithms in a consistent environment. It uses the GNU Radio framework, and is designed to be extensible, allowing a simple inclusion of new algorithms and features. Thus, this simulator enables homogeneous comparisons between algorithms using either real USRP hardware or simulations. We also provide a technical discussion on implementation issues of existing algorithms that were not addressed in the descriptions of the original articles, such as CFO correction, STO correction, and computational complexity. We utilize the framework to compare existing algorithms in a common scenario, including an analysis of the memory requirements and time complexity of the implementation of the collision resolution algorithms. The simulator was presented in [63].

In Chapter 6, we regrouped most of the simulation results of our contributions. We attempted to perform extensive simulations of the main parameters of our algorithms.

7.2 Perspectives

We list below some of the perspectives of our work.

7.2.1 Cooperation of multiple gateways

To the best of our knowledge, all existing collision resolution algorithms focus on a single gateway. When collided frames are received by several gateways, each gateway attempts to recover the frames independently. However, by having the gateways collaborate, more decoding opportunities could be obtained. With a multi-gateway collision resolution algorithm, each gateway could store its own observation on the received collided signals (including the signal phases, the power differences, the time offsets and the frequency offsets). By combining the demodulated symbols from different gateways, it is likely that uncertainties would be easier to solve, as most collision features are receiver-dependent.

7.2.2 Integration of physical layer and MAC layer in simulators

Our current physical layer simulator based on GNU Radio is able to generate and decode LoRa frames, corresponding to a set of signal samples. We can also set the duty-cycle in the simulator during the frame generation. However, we have not implemented most parts of the LoRaWAN protocol stack. The implementations of these features exist in some of the NS-3 LoRaWAN simulators, such as the one described in [9]. In order to extend the capacities of our simulator, we believe that it would be interesting to make our LoRa physical layer GNU Radio simulator interoperable with NS-3. We have made some progress in this direction, but the inter-operability has not yet been achieved.

7.2.3 Dynamic selection of algorithms and the corresponding hyper-parameters

Each collision resolution algorithm has its own advantages and unfeasible cases. Although we have proposed a generic algorithm in Chapter 3 to resolve some of them based on the uncertainties, the performance is still limited by the remaining frames. An intelligent receiver may be able to select one or more of the collision resolution algorithms according to the numbers of collided LoRa frames and the characteristics of the superposed LoRa signals.

In addition, we have shown the influences of different hyper-parameters in FTrack and the SF-DS in Chapter 6. We have also given an analysis of the influence of the number of virtual sub-slots on the robustness to the noise in Chapter 4, which also impacts the number of uncertainties after the decoding. The receiver would also benefit from the dynamic selection of the hyper-parameters, according to the different channel conditions (*e.g.*, SNR) and the numbers of collided LoRa frames that it detects.

7.2.4 Speeding up our LoRa coding algorithm by hardware acceleration

Our algorithm for frame recovery using LoRa coding, described in Chapter 3, may require a large number of computations, as it needs to validate each possible frame, based on the uncertainties. We noticed that these

computations can be parallelized. Thus, we plan to speed-up our LoRa coding algorithm by replacing the serial processing by a parallel processing.

- Recall that the symbol values are demodulated one by one. When all the symbols in a block are demodulated, the possibilities in this block can then be processed by a decoder. When there is only a single frame, the receiver can decode the blocks in parallel. This allows the receiver to obtain the frame data quickly. When there are collided frames with uncertainties in the symbols, the receiver might need to perform the deinterleaving up to ${N_u}^{CR+4}$ times to remove the uncertainties in each block and to retrieve the frame. The possible blocks can again be processed in parallel.

- Besides, note that the ALOHA random access mechanism is likely to generate colliding frames whose symbol edges are not aligned. Unlike the worst case presented in Section 3.1, the symbol blocks of each frame might start at different positions. Thus, the time instants to start decoding the blocks in different frames might be different. The receiver is able to allocate the decoding tasks to different decoders, in order to decode the collided frames, with or without uncertainties, in parallel.

Our algorithm can be implemented on NVIDIA Compute Unified Device Architecture (CUDA), which is a parallel computing platform and programming model designed for general-purpose computing on NVIDIA Graphics Processing Units (GPUs). It was first introduced in 2006 as a programming model for NVIDIA GPUs. It was initially designed to provide developers with a way to harness the massive parallel computing power of GPUs for scientific simulations and other computationally intensive tasks. There are also embedded devices built with NVIDIA GPU for IoT scenario. We have already made some progress on the CUDA implementation. We believe that we can also implement our algorithm on Field Programmable Gate Arrays (FPGA) in order to accelerate the decoding and the validation of the possible symbol blocks as well.

7.2.5 Evaluating the imperfect orthogonality

The authors of [11] showed that the orthogonality of different SFs is not perfect. Similarly, the authors of [39] showed that the orthogonality of uplink and downlink frames is not perfect. Such imperfect orthogonality can cause collisions in some cases, or uncertainties in symbols. Currently, all collision resolution algorithms are mostly considered to process the collisions of frames with the same SF, and with superposed up-chirps. Studying how to improve the imperfect orthogonality of SF or up-chirps/down-chirps is an interesting way to improve the performance of these collision resolution algorithms.

7.2.6 Improving the ADR

The allocation of modulation parameters is important for the network performance, as collisions largely depend on the modulation parameters such as the SF. Thus, it is important to work on the ADR mechanism,

which is the mechanism of LoRaWAN that allocates the SF. It would be interesting to improve this ADR mechanism by taking into account possibilities of collision resolution during the parameter allocation.

Indeed, while studying several agricultural monitoring scenarios, and especially those of the ConnecSenS project, we realized that LoRaWAN networks can address the requirements of most of the applications. The more challenging applications are those with a high spatial density (from 400 to 1200 end-devices per square kilometer) or with a high measurement frequency (in order to reach the saturation of the 1% duty-cycle). Such scenarios include Farm 4.0 (such also called precision agriculture), where the end-devices are deployed in fields, meadows, in livestock buildings, in warehouses, in grain silos, or even in agricultural robots (such as tractors and harvesters). Such scenarios also include dense monitoring campaigns, typically in the setup step, where a high-scale deployment is intended to accurately characterize the environment.

The main technical drawback in these scenarios is the limited throughput of LoRaWAN. It is also likely that the deployment of an agricultural wireless sensor network will be performed by the users, typically farmers, and thus it will be beneficial for them if an automatic mechanism such as the ADR can perform the allocation of the SFs on their behalf.

The example we focused on is the experimental farm of Montoldre, France, owned by INRAE, with parcels covering an area of about 3000m×1500m. A satellite view of this farm is shown on Figure 7.1. We simulated a gateway in the main building of the farm, and deployed several end-devices with a varying density. Then, we observed how the ADR allocated the SF to end-devices. Figure 7.2 shows the SF allocation with end-devices every 10 m, and Figure 7.3 shows the SF allocation with end-devices every 30 m. It can be observed that a large proportion of end-devices have SF7, and few of them have a different SF. Our initial simulations showed that allocating a fixed to all end-devices could yield better results than letting the ADR allocate the SF. For instance, the uplink reception rate for end-devices every 30 m is 65% with the ADR, 90% with SF10, and 96% with SF12. Globally, our initial results show that the ADR yields a good throughput, but a relatively low reception rate and a high energy consumption.

Since the legacy ADR mechanism has several drawbacks, we would like to tackle them, and especially we would like:

- to improve the very slow convergence time of the ADR,
- to take into account the traffic load per SF to allocate the SFs,
- to make the ADR less optimistic, by increasing the uplink reception rate it yields,
- to make the ADR more stable to changes in channel condition,
- to make the ADR more energy efficient.

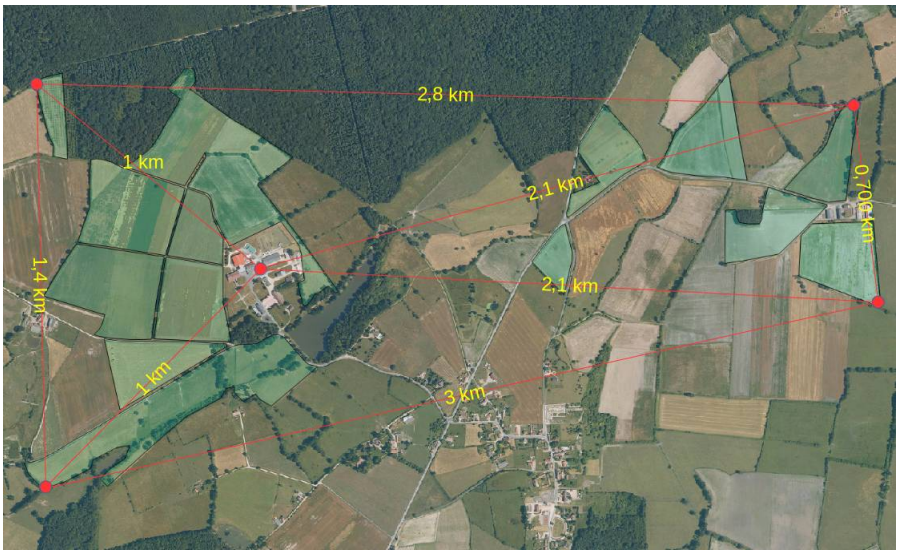


Figure 7.1: A satellite view of the Montoldre farm. The green areas indicate agricultural parcels that are part of the farm. The buildings of the farm are in gray, and is surrounded by the parcels of the left part of the map.

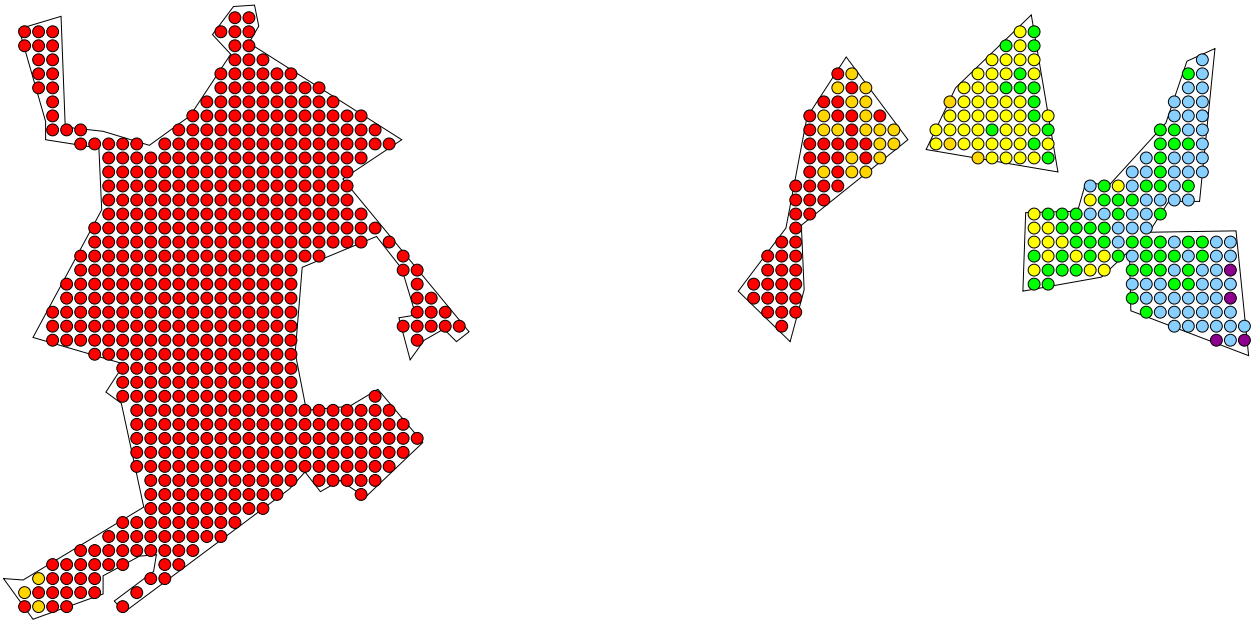


Figure 7.2: SF allocation achieved by the ADR for the Montoldre farm, with an end-device every 10 m. The colors indicate the SF: red is for SF7, orange is for SF8, yellow is for SF9, green is for SF10, blue is for SF11, and purple is for SF12.

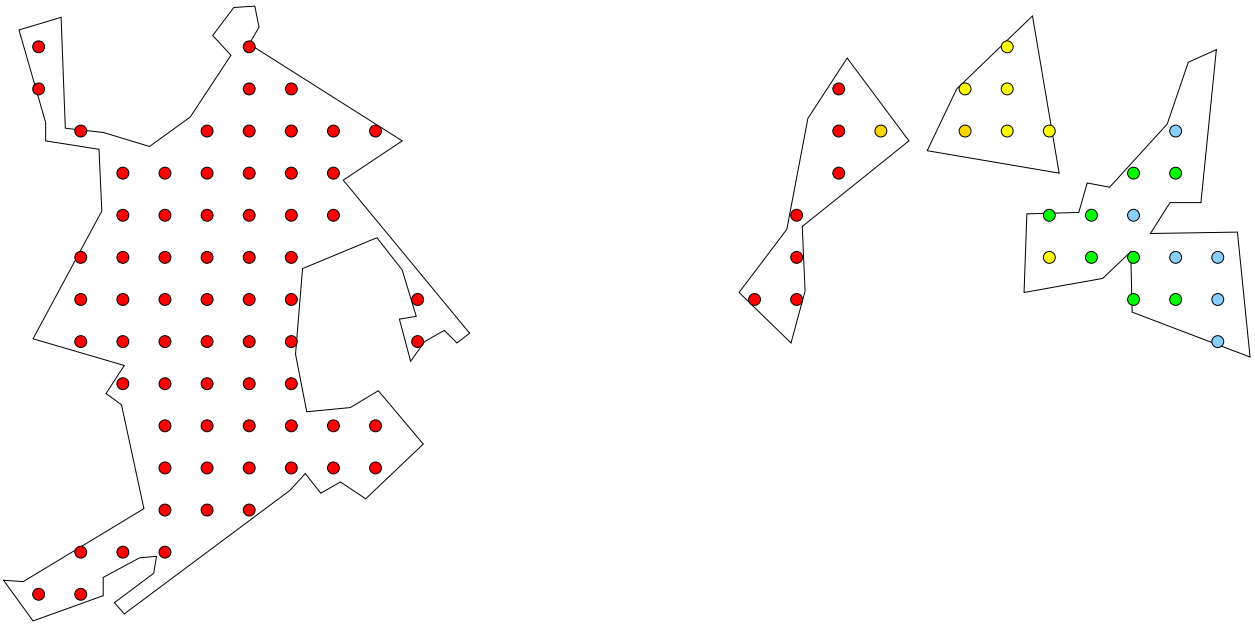


Figure 7.3: SF allocation achieved by the ADR for the Montoldre farm, with an end-device every 30 m. The colors indicate the SF: red is for SF7, orange is for SF8, yellow is for SF9, green is for SF10, blue is for SF11, and purple is for SF12.

7.2.7 Extension to new LoRa-based modulations

Finally, new LoRa-based modulations have been developed by Semtech. They include the use of LoRa in the 2.4 GHz ISM band [21], and the Long-Range Frequency Hopping Spread Spectrum (LR-FHSS) modulation [7] used to communicate from terrestrial end-devices to low Earth orbit satellites. Adapting our collision resolution algorithms to LoRa 2.4 GHz or to LR-FHSS is a very promising idea.

Publications

Related publications:

[N1] **Weixuan Xiao**, Nancy El Rachkidy, and Alexandre Guitton. Signaux radio superposés : problème et solution. In *La Minute Recherche, Université Clermont Auvergne*, 2023.

[C1] **Weixuan Xiao**, Gil De Sousa, Nancy El Rachkidy, and Alexandre Guitton. An open-source GNU radio framework for LoRa physical layer and collision resolution. In *IEEE VTC Fall (Vehicular Technology Conference)*, 2022.

[J1] **Weixuan Xiao**, Megumi Kaneko, Nancy El Rachkidy, and Alexandre Guitton. Integrating LoRa Collision Decoding and MAC Protocols for Enabling IoT Massive Connectivity. *IEEE Internet of Things Magazine*, 2022.

[C2] **Weixuan Xiao**, Nancy El Rachkidy, and Alexandre Guitton. SF-DS: A slot-free decoding scheme for collided lora transmissions. In *IEEE VTC Spring (Vehicular Technology Conference)*, 2022.

[C3] **Weixuan Xiao**, Nancy El Rachkidy, and Alexandre Guitton. Recovering Colliding LoRa Frames from Uncertainties Using LoRa Coding. In Proceedings of the *IEEE 46th Conference on Local Computer Networks (LCN)*, short paper, 2021.

Unrelated publications:

[J2] Dong Ding, Abdellatif Ouahsine, **Weixuan Xiao** and Peng Du. Numerical study of ballast-flight caused by dropping snow/ice blocks in high-speed railways using Discontinuous Deformation Analysis (DDA). *Transportation Geotechnics*, 2021.

[J3] Dong Ding, Abdellatif Ouahsine, **Weixuan Xiao** and Peng Du. CFD/DEM coupled approach for the stability of caisson-type breakwater subjected to violent wave impact. *Ocean Engineering*, 2020.

List of Acronyms

CSS Chirp Spread Spectrum 20

CUDA Compute Unified Device Architecture 138

CSMA Carrier-Sense Multiple Access 100

DFT Discrete Fourier Transform 30

DR Data Rate 43

ECC Error Correction Coding 13

ERDF European Regional Development Fund 3

FDMA Frequency-Division Multiple Access 20

FER Frame Error Rate 40

FFT Fast Fourier Transform 11

FPGA Field Programmable Gate Arrays 138

FSK Frequency-Shift Keying 20

ICS Information and Communication Systems 22

INRAE Institut National de Recherche pour l'Agriculture, l'alimentation et l'Environnement 21

Bibliography

[1] NS-3. https://www.nsnam.org/. accessed Feb. 2023.

[2] F. Adelantado et al. "Understanding the limits of LoRaWAN". In: *IEEE Communications Magazine* (2017). DOI: 10.1109/MCOM.2017.1600613.

[3] O. Afisiadis, A. Burg, and A. Balatsoukas-Stimming. "Coded LoRa Frame Error Rate Analysis". en. In: *arXiv:1911.10245 [eess]* (2019). arXiv: 1911.10245. URL: http://arxiv.org/abs/1911.10245 (visited on 04/29/2021).

[4] M. A. M. Almuhaya et al. "A Survey on LoRaWAN Technology: Recent Trends, Opportunities, Simulation Tools and Future Directions". In: *Electronics* (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11010164. URL: https://www.mdpi.com/2079-9292/11/1/164.

[5] M. A. Ben Temim et al. "A Novel Approach to Process the Multiple Reception of Non-Orthogonal LoRa-Like Signals". en. In: *Proceedings of IEEE International Conference on Communications*. 2020. DOI: 10.1109/ICC40277.2020.9148783. (Visited on 08/20/2020).

[6] C. Bernier, F. Dehmas, and N. Deparis. "Low Complexity LoRa Frame Synchronization for Ultra-Low Power Software-Defined Radios". In: *IEEE Transactions on Communications* (2020). DOI: 10.1109/TCOMM.2020.2974464.

[7] G. Boquet et al. "LR-FHSS: Overview and Performance Analysis". In: *IEEE Communications Magazine* (2021).

[8] M. C. Bor et al. "Do LoRa Low-Power Wide-Area Networks Scale?" In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 2016. DOI: 10.1145/2988287.2989163.

[9] M. Capuzzo, D. Magrin, and A. Zanella. "Confirmed Traffic in LoRaWAN: Pitfalls and Countermeasures". In: *17th Annual Mediterranean Ad Hoc Networking Workshop*. 2018. DOI: 10.23919/MedHocNet.2018.8407095.

[10] M. Centenaro et al. "Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios". In: *IEEE Wireless Communications* (2015). DOI: 10.1109/MWC.2016.7721743.

[11] Daniele Croce et al. “Impact of LoRa Imperfect Orthogonality: Analysis of Link-Level Performance”. In: *IEEE Communications Letters* (2018). ISSN: 1558-2558. DOI: 10.1109/LCOMM.2018.2797057.

[12] N. El Rachkidy, A. Guitton, and M. Kaneko. “Decoding Superposed LoRa Signals”. In: *IEEE Conference on Local Computer Networks*. 2018. DOI: 10.1109/LCN.2018.8638253.

[13] N. El Rachkidy, A. Guitton, and M. Kaneko. “Generalized Slotted MAC Protocol Exploiting LoRa Signal Collisions”. In: *Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*. 2020.

[14] R. Eletreby et al. “Empowering Low-Power Wide Area Networks in Urban Settings”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017.

[15] G. Ferré and A. Giremus. “LoRa Physical Layer Principle and Performance Analysis”. In: *25th IEEE International Conference on Electronics Circuits and Systems*. 2018.

[16] O. Georgiou and U. Raza. “Low Power Wide Area Network Analysis: Can LoRa Scale?” In: *IEEE Wireless Communications Letters* (2017).

[17] R. Ghanaatian et al. “LoRa Digital Receiver Analysis and Implementation”. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. 2019. DOI: 10.1109/ICASSP.2019.8683504.

[18] GNU Radio. https://www.gnuradio.org. accessed Feb. 2022.

[19] HelTec. https://github.com/HelTecAutomation/Heltec_ESP32. accessed 2023-04-10.

[20] HelTec WiFi LoRa 32 V2. https://heltec.org/project/wifi-lora-32/. accessed Feb. 2023.

[21] G. Hochet Derévianckine et al. “Opportunities and Challenges of LoRa 2.4 GHz”. In: *IEEE Communications Magazine* (2023).

[22] B. Hu et al. “SCLoRa: Leveraging Multi-Dimensionality in Decoding Collided LoRa Transmissions”. In: *Proceedings of the IEEE 28th International Conference on Network Protocols*. 2020. DOI: 10.1109/ICNP49622.2020.9259397.

[23] Ingenu. http://www.ingenu.com. accessed 2020-10-01.

[24] NB-IoT. https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/. accessed 2023-04-07.

[25] K. Knight. “Reversing LoRa: Exploring Next-Generation Wireless”. In: *GRCon*. 2016.

[26] B. Laporte-Fauret et al. “An Enhanced LoRa-Like Receiver for the Simultaneous Reception of Two Interfering Signals”. In: *Proceedings of Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. 2019. DOI: 10.1109/PIMRC.2019.8904258.

[27] LoRa Alliance Technical Committee. *LoRaWAN 1.1 Specification*. Standard V1.1. LoRa Alliance, 2017.

[28] D. Magrin, M. Capuzzo, and A. Zanella. "A Thorough Study of LoRaWAN Performance Under Different Parameter Settings". In: *IEEE Internet of Things Journal* (2020). ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2946487.

[29] J. M. Marais, A. M. Abu-Mahfouz, and G. P. Hancke. "A Review of LoRaWAN Simulators: Design Requirements and Limitations". In: *Proceedings of International Multidisciplinary Information Technology and Engineering Conference*. 2019. DOI: 10.1109/IMITEC45504.2019.9015882.

[30] A. Marquet, N. Montavont, and G. Z. Papadopoulos. "Towards an SDR Implementation of LoRa: Reverse-engineering, Demodulation Strategies and Assessment over Rayleigh Channel". en. In: *IEEE Computer Communications* (2020). ISSN: 0140-3664. DOI: 10.1016/j.comcom.2020.02.034. URL: http://www.sciencedirect.com/science/article/pii/S0140366419314665 (visited on 09/21/2020).

[31] L. Moiroux-Arvis et al. "ConnecSenS, a Versatile IoT Platform for Environment Monitoring: Bring Water to Cloud". In: *Sensors* (2023). ISSN: 1424-8220. DOI: 10.3390/s23062896. URL: https://www.mdpi.com/1424-8220/23/6/2896.

[32] U. Noreen, L. Clavier, and A. Bounceur. "LoRa-like CSS-based PHY layer, Capture Effect and Serial Interference Cancellation". In: *24th European Wireless Conference*. 2018.

[33] Nuand. https://www.nuand.com/bladerf-2-0-micro/. accessed 2023-04-10.

[34] A. Pop et al. "Does Bidirectional Traffic do More Harm than Good in LoRaWAN based LPWA Networks?" In: *Proceedings of IEEE Global Communications Conference*. IEEE. 2017.

[35] S. Popli, R.K. Jha, and S. Jain. "A Survey on Energy Efficient Narrowband Internet of Things (NBIoT): Architecture, Application and Challenges". In: *IEEE Access* (2019). DOI: 10.1109/ACCESS.2018.2881533.

[36] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing*. Pearson, 2006.

[37] U. Raza, P. Kulkarni, and M. Sooriyabandara. "Low Power Wide Area Networks: An Overview". In: *IEEE communications surveys & tutorials* (2017).

[38] P. Robyns et al. "A Multi-Channel Software Decoder for the LoRa Modulation Scheme". In: *Proceedings of International Conference on Internet of Things, Big Data and Security*. 2018. DOI: 10.5220/0006668400410051.

[39] R. Saroui et al. "Uplink and Downlink are Not Orthogonal in LoRaWAN!" In: *IEEE Vehicular Technology Conference Fall*. 2022. DOI: 10.1109/VTC2022-Fall57202.2022.10012754.

[40] O. B. A. Seller and N.s Sornin. *Low Power Long Range Transmitter*. EP2763321A1, 2013.

[41] Semtech Corporation. *An In-depth Look at LoRaWAN Class B Devices*. Technical Paper. Semtech, 2019. URL: https://lora-developers.semtech.com/uploads/documents/files/LoRaWAN_Class_B_Devices_In_Depth_Downloadable.pdf.

[42] Semtech Corporation. *AN1200.13 SX1272/3/6/7/8: LoRa Modem Designer's Guide*. Application note. Semtech, 2013.

[43] Semtech Corporation. *AN1200.22 LoRa Modulation Basics*. Application note Revision 2. Semtech, 2015. URL: http://www.semtech.com/uploads/documents/an1200.22.pdf.

[44] Semtech Corporation. *LoRaWAN v1.1 Regional Parameters*. Tech. rep. Revision A. Semtech, 2017. URL: https://lora-alliance.org/sites/default/files/2018-05/lorawan-regional-parameters-v1.1ra.pdf.

[45] Semtech Corporation. *SX1272/73 - 860 MHz to 1020 MHz Low Power Long Range Transceiver*. Application note. Semtech, 2015.

[46] Semtech Corporation. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver*. Application note. Semtech, 2017.

[47] Semtech Corporation. *SX1301 Datasheet v2.4 ed.* Application note. Semtech, 2017.

[48] M. O. Shahid et al. "Concurrent Interference Cancellation: Decoding Multi-packet Collisions in LoRa". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2021.

[49] Sigfox. http://www.sigfox.com. accessed 2020-10-01.

[50] SimPy. https://gitlab.com/team-simpy/simpy/. accessed Feb. 2023.

[51] R. B. Sorensen et al. "Analysis of LoRaWAN Uplink with Multiple Demodulating Paths and Capture Effect". In: *IEEE International Conference on Communications*. 2019. DOI: 10.1109/ICC.2019.8761142.

[52] R. B. Sørensen et al. "Analysis of Latency and MAC-layer Performance for Class A LoRaWAN". In: *IEEE Wireless Communications Letters* (2017). DOI: 10.1109/LWC.2017.2716932.

[53] Joachim Tapparel et al. "An Open-Source LoRa Physical Layer Prototype on GNU Radio". en. In: *arXiv:2002.08208 [eess]* (2020). arXiv: 2002.08208. URL: http://arxiv.org/abs/2002.08208 (visited on 08/20/2020).

[54] T. To and A. Duda. "Simulation of LoRa in NS-3: Improving LoRa Performance with CSMA". en. In: *2018 IEEE International Conference on Communications*. IEEE, 2018. ISBN: 978-1-5386-3180-5. DOI: 10.1109/ICC.2018.8422800. URL: https://ieeexplore.ieee.org/document/8422800/ (visited on 05/25/2020).

[55] S. Tong, Z. Xu, and J. Wang. "CoLoRa: Enabling Multi-Packet Reception in LoRa". In: *IEEE Conference on Computer Communications*. 2020. DOI: 10.1109/INFOCOM41043.2020.9155509.

[56] F. Van den Abeele et al. "Scalability Analysis of Large-scale LoRaWAN Networks in NS-3". In: *IEEE Internet of Things Journal* (2017).

[57] Z. Wang et al. “Online Concurrent Transmissions at LoRa Gateway”. In: *IEEE Conference on Computer Communications*. 2020. DOI: 10.1109/INFOCOM41043.2020.9155433.

[58] A. Waret et al. “LoRa Throughput Analysis With Imperfect Spreading Factor Orthogonality”. In: *IEEE Wireless Communications Letters* (2019).

[59] X. Xia, Y. Zheng, and T. Gu. “FTrack: Parallel Decoding for LoRa Transmissions”. en. In: *Proceedings of the ACM 17th Conference on Embedded Networked Sensor Systems*. 2019. ISBN: 978-1-4503-6950-3. DOI: 10.1145/3356250.3360024. URL: https://dl.acm.org/doi/10.1145/3356250.3360024 (visited on 04/16/2021).

[60] W. Xiao. https://github.com/Inokinoki/Heltec_ESP32. accessed 2023-04-10.

[61] W. Xiao, N. El Rachkidy, and A. Guitton. “Recovering Colliding LoRa Frames from Uncertainties Using LoRa Coding”. In: *Proceedings of the IEEE 46th Conference on Local Computer Networks*. 2021. DOI: 10.1109/LCN52139.2021.9524949.

[62] W. Xiao, N. El Rachkidy, and A. Guitton. “SF-DS: A Slot-Free Decoding Scheme for Collided LoRa Transmissions”. In: *IEEE Vehicular Technology Conference Spring*. 2022.

[63] W. Xiao et al. “An Open-Source GNU Radio Framework for LoRa Physical Layer and Collision Resolution”. In: *IEEE Vehicular Technology Conference Fall*. 2022.

[64] W. Xiao et al. “Integrating LoRa Collision Decoding and MAC Protocols for Enabling IoT Massive Connectivity”. In: *IEEE IoT Magazine* (2022).

[65] Z. Xu et al. “FlipLoRa: Resolving Collisions with Up-Down Quasi-Orthogonality”. In: *17th Annual IEEE International Conference on Sensing, Communication, and Networking*. 2020. DOI: 10.1109/SECON48991.2020.9158432.

Appendix A

Résumé étendu de la thèse

A.1 Introduction

Les réseaux sans fil LPWAN sont conçus pour permettre à des dispositifs terminaux sans fil de communiquer sur de longues distances, avec une faible consommation d'énergie. Les LPWAN ont de nombreuses applications dans le domaine des villes intelligentes, du transport intelligent et de la logistique [10, 2], ainsi que dans le domaine de l'industrie du futur (passage à l'industrie 4.0 ou 5.0), de la surveillance des infrastructures, de l'agriculture de précision, du suivi des animaux ou encore de la surveillance environnementale [37].

Technologies LPWAN. Plusieurs technologies concurrentes existent dans les LPWAN, et notamment : Sigfox [49], Ingenu [23], NB-IoT [24], et LoRa [43]. Bien que toutes ces technologies soient conçues pour fournir une connectivité longue portée à faible consommation d'énergie, il existe des différences et des compromis clés entre elles, par exemple en terme de débit ou de bandes de fréquences utilisées.

Sigfox utilise une modulation sans fil Ultra Narrow Band sur la bande de fréquences sous-GHz de l'ISM, et permet d'atteindre un débit binaire allant jusqu'à 100 bps. Les protocoles des couches supérieures ne sont pas disponibles publiquement car le modèle économique de cette entreprise est d'être un opérateur de service IoT [10]. La technologie prend en charge une communication bi-directionnelle. Cependant, la charge utile des paquets transmis par Sigfox est limitée à seulement 12 octets, et le nombre de paquets qu'un nœud peut transmettre par jour est lui aussi limité.

Ingenu est une technologie LPWAN appartenant à On-Ramp Wireless, une entreprise qui a été à l'avant-garde de la norme 802.15.4k [10]. La couche physique exploite la technologie brevetée RPMA. Cette technologie fonctionne dans la bande ISM à 2.4 GHz. Le choix de cette bande rend l'environnement radio complexe, car il y a des interférences potentielles venant de nombreuses autres technologies, comme le Bluetooth et le Wi-Fi [4]. RPMA utilise donc une large bande passante et une grande puissance de transmission (dans les

limites de la réglementation) [4], au détriment de la consommation d'énergie.

NB-IoT est une technologie introduite par 3GPP [35]. Elle exploite l'infrastructure existante du réseau mobile pour transmettre les données des applications LPWAN par des canaux sous licence. NB-IoT utilise la technique d'accès multiple par répartition en fréquence simple (FDMA) pour les transmissions montantes, qui limite le débit binaire à 20 kbps. L'OFDMA et le QPSK sont utilisés dans la liaison descendante [4], ce qui permet d'atteindre un débit binaire allant jusqu'à 200 kbps.

LoRa est une technologie LPWAN brevetée par Semtech. Elle permet deux modulations différentes : (i) une modulation CSS avec un débit faible, de 300 bps à 5500 bps, mais avec une grande robustesse, et (ii) une modulation FSK avec un débit de transmission plus élevé allant jusqu'à 50 kbps. Les utilisateurs de LoRa peuvent ajuster un grand nombre de paramètres, notamment la modulation et le débit binaire, afin d'optimiser les performances du réseau pour une application spécifique. De plus, LoRa fonctionne dans les bandes ISM libres, ce qui permet un déploiement facile des dispositifs terminaux. Finalement, la LoRa Alliance défend la norme Long-Range Wide Area Network (LoRaWAN) [27], qui définit le protocole MAC et la topologie de réseau pour un LPWAN utilisant la couche physique LoRa. LoRaWAN garantit que les dispositifs terminaux LoRa puissent communiquer sur un réseau pour répondre à différentes exigences et contraintes. LoRaWAN permet de déployer des réseaux publics ou privés. Tous ces éléments font de LoRa et LoRaWAN des technologies très populaires.

Contexte : le projet ConnecSenS. ConnecSenS [31] est un projet soutenu par la région Auvergne-Rhône-Alpes (France) qui vise à comprendre les changements mondiaux en utilisant une approche de développement durable. Ce projet se concentre sur la modélisation des écosystèmes et des territoires dynamiques à plusieurs échelles, avec une plateforme IoT pour la surveillance environnementale en Auvergne.

Dans le projet ConnecSenS, un capteur sans fil appelé SoLo a été développé afin de surveiller l'environnement pendant plusieurs mois sans changement de batterie et sans récupération d'énergie, en se basant sur LoRa et LoRaWAN. Ainsi, la consommation d'énergie et le taux de livraison des trames sont cruciaux pour les dispositifs terminaux SoLo.

Cette thèse est réalisée dans le cadre de ConnecSenS, avec la cible d'améliorer les capacités de communication de LoRa, afin d'améliorer les capacités des dispositifs terminaux SoLo. Le projet est financé par le CPER, avec pour partenaires institutionnels : le CNRS, l'UCA et l'INRAE.

Le travail de cette thèse a été réalisé au sein de l'axe SIC du LIMOS, à l'UCA, à Clermont-Ferrand (France). La thèse a été dirigée par Professeur Alexandre GUITTON de l'UCA, et supervisée par Docteur Nancy EL RACHKIDY de l'UCA et Docteur Gil DE SOUSA de l'INRAE.

Problématique de la thèse. Dans les réseaux LoRa et LoRaWAN, des collisions peuvent se produire en raison du mécanisme d'accès au médium aléatoire nommé ALOHA utilisé dans LoRaWAN, notamment dans

le cas de réseaux denses. LoRa ne peut pas récupérer les collisions lorsque plusieurs dispositifs terminaux transmettent des trames simultanément en utilisant des paramètres similaires. Ainsi, les trames qui entrent en collision doivent être retransmises, ce qui réduit le débit du réseau et augmente la consommation d'énergie.

Plusieurs chercheurs ont proposé d'améliorer les performances des réseaux LoRaWAN en optimisant la sélection des paramètres, par exemple en modifiant le mécanisme ADR de LoRaWAN. Cependant, des collisions peuvent encore se produire dans des réseaux denses.

Nous avons décidé de suivre une autre approche, basée sur la résolution des collisions, car le débit indicatif de LoRa est loin de la capacité du canal donnée par le théorème de Shannon-Hartley. L'objectif de cette thèse est de réduire le nombre de retransmissions en essayant de détecter et de résoudre les collisions à la réception. De cette manière, le débit est amélioré et la consommation d'énergie est réduite. Avec la résolution de collisions, nous pensons pouvoir améliorer le passage à l'échelle des réseaux LoRaWAN, et ainsi les rendre plus attractifs pour un plus grand nombre d'applications IoT.

Dans cette thèse, nous nous concentrons sur la résolution des collisions dans un réseau LoRaWAN, avec la modulation LoRa uniquement (c'est-à-dire, en ignorant la modulation FSK).

Plan du résumé de la thèse et contributions. Le plan de ce résumé est le suivant. La section A.2 présente la modulation LoRa, le protocole LoRaWAN, et plusieurs algorithmes existants permettant de résoudre les collisions. La section A.3 présente notre première contribution, qui est un algorithme permettant de réduire les incertitudes survenant lors des collisions au moyen d'informations issues du codage LoRa. La section A.4 présente notre deuxième contribution, qui est le schéma de décodage de trames LoRa en collision appelé SF-DS. Il s'agit de la contribution principale de la thèse. SF-DS utilise un algorithme de détection de préambule capable de détecter plusieurs préambules LoRa en collision. SF-DS est ensuite capable de démoduler plusieurs cas de trames en collision, en calculant les fréquences plusieurs fois pendant chaque symbole LoRa. La section A.5 présente notre troisième contribution, qui est un simulateur implémenté sous GNU Radio et permettant de comparer des algorithmes de couches physiques pour LoRa. La section A.6 donne de nombreux résultats de simulation, et montre que SF-DS surpasse les autres protocoles de la littérature dans une grande variété de scénarios. La section A.7 conclut nos travaux de thèse.

A.2 État de l'art

LoRa est une modulation robuste qui permet des communications à longue portée, et constitue ainsi un élément clé des réseaux LPWAN. LoRaWAN définit les couches supérieures d'une pile de protocoles basée sur LoRa. LoRaWAN est largement utilisée dans les applications LPWAN actuelles, car elle permet le déploiement rapide et flexible de réseaux à faible coût, y compris dans des endroits éloignés. Dans cette partie, nous décrivons d'abord LoRa, puis LoRaWAN, puis plusieurs protocoles conçus pour résoudre les

collisions LoRa.

A.2.1 LoRa

Modulation. LoRa utilise une modulation CSS : le modulateur LoRa encode les données à l'aide d'une série de chirps linéaires, également appelés symboles. Chaque chirp est composé d'une variation linéaire de la fréquence dans le temps, au sein d'une certaine BW. La durée d'un chirp est appelée SD, et est égale à $2^{SF}/BW$, dans lequel SF est le facteur d'étalement. Le choix de la valeur du SF permet de réaliser un compromis entre débit, robustesse et économie d'énergie.

Les up-chirps et les down-chirps représentent des chirps linéaires avec une fréquence croissante et décroissante, respectivement. Dans la bande de base, c'est-à-dire lorsque la fréquence du symbole est centrée sur 0 kHz, la fréquence des chirps varie dans $[-\frac{BW}{2}; \frac{BW}{2}]$. Un up-chirp normalisé est un up-chirp dont la fréquence initiale commence à la valeur minimale, qui est $-\frac{BW}{2}$. Un down-chirp normalisé est un down-chirp dont la fréquence initiale commence à la valeur maximale, qui est $\frac{BW}{2}$.

Pour encoder une valeur de symbole $m \in [0; M-1]$ dans un up-chirp (respectivement un down-chirp), un décalage est appliqué à la fréquence initiale du chirp. La fréquence commence à augmenter (respectivement à diminuer) à partir de la fréquence initiale décalée. Lorsque la fréquence atteint $BW/2$ (respectivement $-BW/2$), la fréquence revient à la valeur minimale $-BW/2$ (respectivement la valeur maximale $BW/2$) afin de rester dans BW. Nous notons par $\tau_m = \frac{m}{M}SD$ le moment à l'intérieur d'un chirp où la fréquence atteint la frontière de BW. La fréquence instantanée $f_u^m(t)$ d'un up-chirp encodant la valeur de symbole m, et la fréquence instantanée d'un down-chirp encodant la valeur de symbole m, sont ainsi définies par :

$$f_u^m(t) = \begin{cases} \frac{M}{SD^2}(t+\tau_m), t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{M}{SD^2}(t+\tau_m) - \frac{M}{SD}, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}), \end{cases} \tag{A.1}$$

$$f_d^m(t) = \begin{cases} -\frac{M}{SD^2}(t-\tau_m), t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ -\frac{M}{SD^2}(t-\tau_m) + \frac{M}{SD}, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{A.2}$$

Nous notons f_u^0 la fréquence du up-chirp normalisé, et f_d^0 la fréquence du down-chirp normalisé.

Une trame LoRa peut être soit en liaison montante[1] ou en liaison descendante[2]. Une trame en liaison montante LoRa se compose d'un préambule, d'un en-tête optionnel et de données utiles, comme illustré sur la Figure A.1. Le préambule est constitué de plusieurs up-chirps normalisés (généralement huit), de deux up-chirps appelés mot de synchronisation qui servent d'identifiant de réseau, et de 2.25 down-chirps représentant le SFD. L'en-tête optionnel et les données utiles suivent. Un trame en liaison descendante LoRa est similaire, à l'exception que les up-chirps et down-chirps dans la trame sont inversés.

[1]Les liaisons montantes sont utilisées dans LoRaWAN pour les trames envoyées par un dispositif terminal vers une passerelle.
[2]Les liaisons descendantes sont utilisées dans LoRaWAN pour les trames envoyées par une passerelle vers un dispositif.

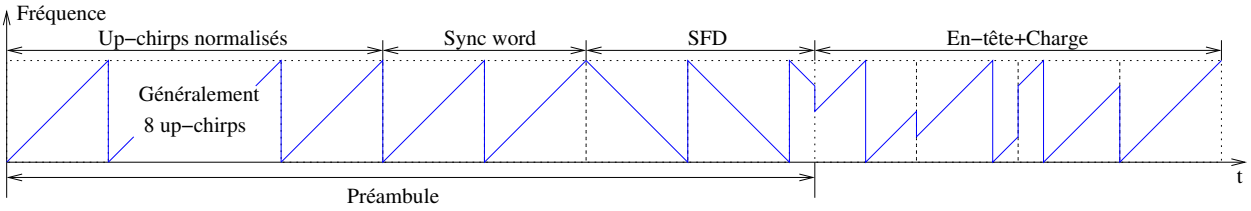


Figure A.1: Un exemple de trame LoRa en liaison montante, avec huit up-chirps normalisés, deux up-chirps représentant le mot de synchronisation, 2.25 down-chirps pour le SFD, et quatre up-chirps pour l'en-tête et les données utiles.

Démodulation. Pour démoduler les symboles d'une trame, un récepteur LoRa doit être synchronisé avec les symboles transmis. Cette synchronisation est réalisée en détectant les symboles répétés du préambule, jusqu'à la détection de l'inversion des chirps du SFD. Le récepteur vérifie ensuite que le mot de synchronisation correspond à la valeur de symbole attendue. Ensuite, le récepteur peut commencer à décoder l'en-tête et les données utiles.

La démodulation de l'en-tête et de la charge utile commence en multipliant chaque symbole reçu par un chirp conjugué normalisé : pour les trames en liaison montante, chaque up-chirp reçu est multiplié par un down-chirp normalisé, et pour les trames en liaison descendante, chaque down-chirp reçu est multiplié par un up-chirp normalisé. Une telle multiplication de signaux sous forme complexe peut être interprétée comme une addition des fréquences des chirps. Cela élimine la variation temporelle de la fréquence dans le chirp d'origine, et est appelé déchirpement. En effet, après le déchirpement, les fréquences instantanées du up-chirp décrites dans l'équation A.1 deviennent les suivantes :

$$f_u^m(t) + f_d(t) = \begin{cases} \frac{M}{SD^2}\tau_m, t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{M}{SD^2}\tau_m - BW, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{A.3}$$

Il es important de noter que t disparaît ici par rapport à l'équation A.3, et $f_u^m(t)+f_d(t)$ donne deux fréquences constantes séparées par BW.

Finalement, le démodulateur doit extraire la valeur du symbole à partir des deux composantes de fréquence obtenues, pour chaque symbole. Pour mettre en évidence la relation entre ces deux composantes de fréquence et la valeur du symbole, réécrivons l'équation A.3 avec la valeur de symbole attendue m plutôt qu'avec τ_m, comme suit :

$$f_u^m(t) + f_d(t) = \begin{cases} \frac{m}{M}BW, t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m), \\ \frac{m}{M}BW - BW, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}). \end{cases} \tag{A.4}$$

Les deux composantes de fréquence dépendent uniquement de la valeur du symbole m et de la largeur de bande BW, ce qui signifie que m devient facile à obtenir, en théorie. Cependant, l'estimation de m

en pratique nécessite plusieurs échantillons du signal LoRa. Le récepteur collecte plusieurs échantillons avec un taux d'échantillonnage $f_s = \frac{1}{T_s}$, où T_s est le temps d'échantillonnage. Ensuite, le démodulateur effectue une DFT de n points pendant chaque période SD pour extraire les deux composantes de fréquence. La DFT prend n échantillons du signal dans le domaine temporel, et les transforme en n éléments dans le domaine fréquentiel, correspondant à l'amplitude de chaque composante de fréquence. En pratique, le démodulateur peut exploiter l'algorithme FFT, qui est une implémentation rapide de la DFT, avec une complexité temporelle de $O(n \log n)$. Il est important de noter que si le démodulateur utilise BW comme taux d'échantillonnage f_s, le nombre d'échantillons n pendant chaque période SD est donné comme suit :

$$n = SD \times f_s = SD \times BW = \frac{M}{BW} BW = M \tag{A.5}$$

Par conséquent, le résultat de la FFT sur chaque symbole contient M éléments, où chaque élément représente une composante de fréquence avec un intervalle de BW/M.

Afin de déduire la valeur du symbole m à partir des résultats de la FFT d'un symbole, considérons d'abord la composante de fréquence positive de l'équation A.4, qui varie entre 0 et $BW/2$. Après la FFT, le premier élément représente la distribution de l'énergie de la fréquence 0, le deuxième élément représente la distribution de l'énergie de la fréquence $\frac{1}{M}BW$, et ainsi de suite. La distribution de l'énergie dépend des niveaux de puissance des signaux et de la synchronisation. L'indice de chaque élément correspond donc à la valeur du symbole m. Dans ce cas, le démodulateur peut simplement considérer l'indice de la composante de fréquence maximale dans la bande passante BW comme la valeur du symbole démodulé m pendant chaque période SD. En bref, le démodulateur peut prendre l'indice du pic de FFT le plus fort comme estimation de la valeur du symbole pendant chaque période SD.

Selon le théorème d'échantillonnage de Nyquist-Shannon [36], les fréquences prises en compte avec un taux d'échantillonnage de $f_s = BW$ sont limitées à la plage $[-BW/2, BW/2]$, qui est symétrique autour de la fréquence 0. Ce taux d'échantillonnage produit ce qu'on appelle de l'aliasing, ce qui limite la plus haute fréquence absolue à une valeur modulo BW.

Dans le cas de la démodulation LoRa, l'aliasing rend les deux cas de l'équation A.3 identiques dans le domaine des fréquences limitées en bande, comme illustré dans l'équation A.6.

$$\begin{aligned} Aliasing(f_u^m(t) + f_d(t)) &= \frac{m}{M} BW \mod BW \\ &= \begin{cases} \frac{m}{M} BW, m \in [0, \frac{M}{2}), \\ \frac{m}{M} BW - BW \mod BW = \frac{m}{M} BW, m \in [\frac{M}{2}, m). \end{cases} \end{aligned} \tag{A.6}$$

Par conséquent, si $f_s = BW$ et que la synchronisation est parfaite, le démodulateur n'observe qu'un seul pic de la FFT pendant la période SD donnée, ce qui lui permet d'obtenir la valeur du symbole m. Les valeurs de symboles démodulées pendant chaque période SD peuvent ensuite être transmises au décodeur afin de décoder la trame.

A.2.2 LoRaWAN

La topologie utilisée par LoRaWAN comporte des dispositifs terminaux, des passerelles, un serveur réseau (NS), un serveur d'association (JS) et des serveurs applicatifs. Les dispositifs terminaux envoient des trames LoRa en liaison montante à toutes les passerelles accessibles. Les passerelles restent à l'écoute sur tous les canaux et pour tous les SF. Lorsqu'une passerelle détecte le préambule d'une trame montante, elle alloue un démodulateur dont le rôle est de démoduler et décoder la trame. Ensuite, chaque passerelle envoie un paquet contenant la trame vers le NS en utilisant le réseau IP. Le NS supprime les trames en double et transfère le paquet vers le serveur d'application correspondant à l'identifiant de l'émetteur, encore une fois en utilisant le réseau IP. Les paquets de liaison descendante sont envoyés par le serveur d'application par l'intermédiaire du NS s'il s'agit de paquets de données, ou par le NS lui-même s'il s'agit de paquets de contrôle. Le NS choisit une passerelle et transfère le paquet vers celle-ci. Ensuite, la passerelle utilise le réseau LoRa pour envoyer la trame en liaison descendante vers le dispositif. Le JS est utilisé par le NS lorsque de nouveaux dispositifs terminaux rejoignent le réseau.

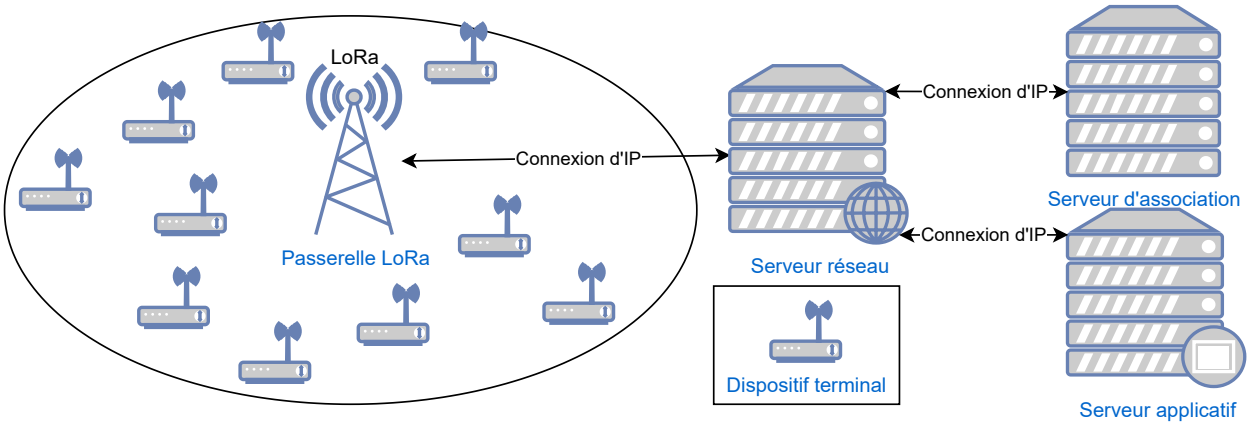


Figure A.2: Un réseau LoRa typique basé sur l'architecture LoRaWAN, avec une seule passerelle.

LoRaWAN définit trois classes de dispositifs terminaux : la classe A, la classe B et la classe C. La classe A est obligatoire pour tous les dispositifs terminaux, tandis que les classes B et C sont optionnelles.

La Classe A est dédiée aux communications à faible consommation d'énergie. Un mécanisme d'accès au medium aléatoire ALOHA est utilisé : les dispositifs terminaux peuvent transmettre à tout moment. Pour ce faire, le dispositif terminal choisit aléatoirement un canal et transmet sa trame. Ensuite, le dispositif terminal attend une éventuelle réponse en ouvrant deux fenêtres de réception courtes appelées RX1 et RX2, respectivement une et deux secondes après la fin de la transmission, comme illustré sur la figure A.3.

La classe B permet des liaisons descendantes avec un délai garanti, et est mise en œuvre à l'aide d'un mécanisme basé sur des balises. Les passerelles LoRa émettent périodiquement et de manière synchrone des balises. Lorsqu'un dispositif terminal reçoit une balise, il configure de courtes fenêtres de liaison descendante

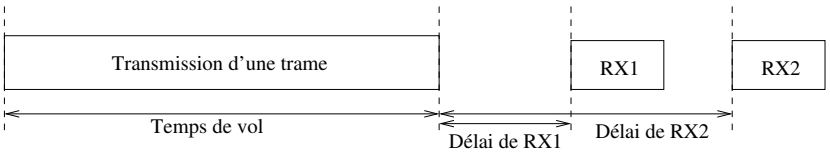


Figure A.3: Fenêtres de réception RX1 et RX2 en classe A de LoRaWAN.

appelées "ping slots". Ces fenêtres sont ouvertes en plus des fenêtres de réception de la classe A.

La classe C permet des liaisons descendantes rapides en supposant que les dispositifs terminaux n'ont pas de contraintes d'énergie. Les dispositifs terminaux peuvent ainsi rester à l'écoute des canaux pour recevoir des trames de liaison descendante à tout moment.

Ces trois classes sont conçues pour s'adapter à différents scénarios d'application. Il faut noter que la classe A est celle qui consomme la plus petite quantité d'énergie et est donc de loin la classe la plus couramment utilisée dans LoRaWAN.

A.2.3 Collisions dans LoRa

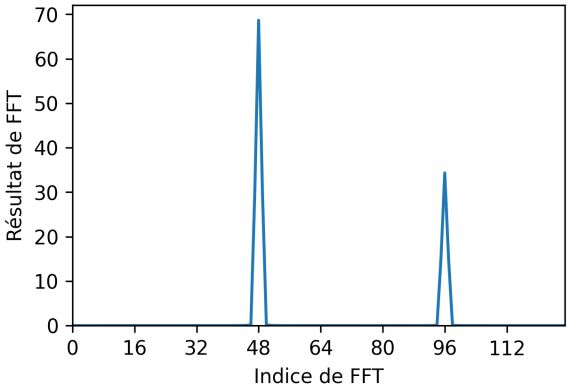
Gestion des collisions dans le LoRa natif. Les communications LoRa sur différents SF étaient initialement considérées comme orthogonales [42], ce qui signifie qu'on pensait qu'elles ne provoquaient pas d'interférences les unes avec les autres. Cela était dû au fait que les signaux avec différentes SF ont des durées de symbole différentes : lors de la démodulation d'un signal superposé selon une durée de symbole de référence, la composante temporelle du signal avec la mauvaise durée de symbole n'est pas supprimée, et donc son énergie est répartie dans plusieurs bandes de la FFT. Cependant, les auteurs de [11] ont montré que cette orthogonalité est bonne mais pas parfaite, et elle est maintenant appelée quasi-orthogonalité. Lorsqu'un signal LoRa interférant avec un SF SF_{int} est beaucoup plus fort que le signal souhaité avec un SF SF_{ref}, avec $SF_{int} \neq SF_{ref}$, le démodulateur peut se tromper sur le pic de la FFT et peut ainsi démoduler la mauvaise valeur de symbole. Les auteurs de [11] ont donné les seuils de différence de puissance, pour chaque paire de SF, à la fois par simulation et par expérimentation. Cette différence de puissance varie de -7dB à -25dB pour des signaux LoRa superposés, de SF6 à SF12.

LoRa peut exploiter l'effet de capture pour démoduler et décoder correctement le signal LoRa le plus fort, même s'il est superposé à un autre signal sur le même SF et sur le même canal. Deux cas peuvent entraîner l'effet de capture lors des collisions de trames LoRa :

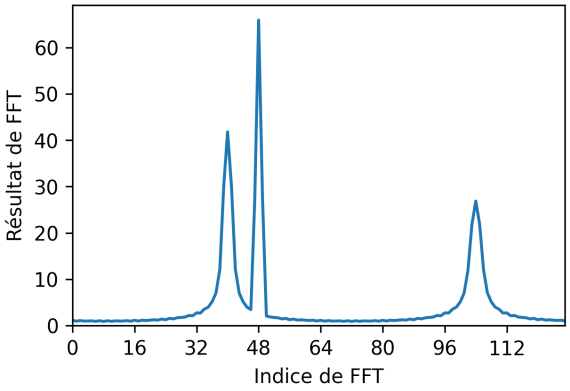
- *Différence de niveaux de puissance :* La figure A.4(a) montre les deux pics produits pendant une durée de symbole de deux trames parfaitement synchronisées, l'une reçue avec une forte puissance, et l'autre reçue avec une faible puissance : le symbole 48 est reçu avec une puissance élevée, tandis que le symbole 96 est reçu avec une faible puissance. Le démodulateur produit le pic le plus fort, c'est-à-dire le pic de

valeur 48. En raison de la différence de puissance entre les deux trames, il est probable que tous les autres symboles de la trame forte seront capturés, ce qui illustre l'effet de capture de LoRa.

- *Différence de puissance dues à la désynchronisation :* La figure A.4(b) montre les trois pics produits pendant une durée de symbole de deux trames légèrement désynchronisées, le récepteur étant synchronisé sur la première trame : le symbole 48 est le symbole de la première trame, le symbole 32 est la fin d'un symbole de la deuxième trame, et le symbole 96 est le début du symbole suivant de la deuxième trame. Les valeurs correspondant aux symboles de la deuxième trame sont légèrement décalés en raison du retard de la deuxième trame. Bien que ces deux trames soient considérées comme reçues avec une puissance similaire, le décalage temporel entre les trames rend l'amplitude des pics de la deuxième trame plus faibles. Le démodulateur produit le pic le plus fort, c'est-à-dire la valeur 48, et le symbole correct est décodé. Une fois de plus, il est probable que tous les symboles de la première trame seront capturés.



(a) Symbole 48 et symbole 96 de deux trames synchronisées. La trame avec le symbole 48 est reçue avec deux fois la puissance de l'autre trame.



(b) Symbole 48 de la trame synchronisée, et symbole 32 suivi du symbole 96 d'une trame désynchronisée. Deux trames sont reçues avec le même niveau de puissance.

Figure A.4: Piques de FFT de deux trames superposées avec SF7.

En raison de la nature de l'accès aléatoire ALOHA, les trames sont rarement synchronisées. Par conséquent, à la fois le niveau de puissance et la désynchronisation aident le décodeur à capturer une trame, si possible. Cependant, étant donné que les trames les plus faibles sont perdues, elles doivent être éventuellement retransmises.

Algorithmes de résolution de collision existants. Les up-chirps et down-chirps sont orthogonaux, même s'ils utilisent le même SF. Cependant, il a été démontré dans [39] que les liaisons montantes et descendantes ne sont pas orthogonales, en raison du fait que les trames de liaisons montantes (respectivement,

descendantes) contiennent quelques down-chirps (respectivement, up-chirps), qui interfèrent avec la plupart des chirps de liaisons descendantes (respectivement, montantes). À notre connaissance, FlipLoRa [65] est le seul protocole qui utilise cette orthogonalité pour résoudre les collisions.

Dans [64], nous avons proposé une catégorisation des algorithmes de résolution de collisions pour LoRa, qui comprend plusieurs algorithmes issus de l'état de l'art. Notre catégorisation définit deux principaux types d'algorithmes de résolution de collisions : ceux basés sur la puissance et ceux basés sur le temps. L'approche basée sur la puissance est principalement axée sur l'effet de capture des trames avec des niveaux de puissance différents, tandis que l'approche basée sur le temps exploite la désynchronisation entre les trames en collision pour les distinguer.

SIC a été proposé pour la première fois dans [32] afin de tirer parti de l'effet de capture et d'améliorer les performances du réseau. Les auteurs ont réalisé une analyse du SIC basée sur l'effet de capture de LoRa. Dans [26], les auteurs mettent en œuvre le SIC sur du matériel réel pour décoder deux trames superposées : le récepteur décode de manière récursive la trame avec le signal le plus fort, reproduit le signal correspondant, et le supprime des signaux superposés reçus afin de décoder les signaux plus faibles.

CHOIR [14] utilise de petites déviations de fréquence pour distinguer les trames. Les auteurs montrent que ces petites déviations proviennent des imperfections matérielles des émetteurs et sont stables pendant toute la durée de la trame. Afin de les capturer, les auteurs appliquent la FFT sur une fenêtre large, dont la taille est dix fois supérieure à la taille originale.

FTrack [59] utilise le décalage temporel entre les trames en collision pour les identifier. Lorsque le récepteur effectue l'opération de déchirpement, chaque symbole devient une fréquence constante sur SD, appelée piste par les auteurs. Étant donné que la fréquence des symboles d'une trame ne change qu'aux frontières de symboles, FTrack utilise ces frontières pour associer les symboles à la trame correcte.

OCT [57] utilise également le décalage temporel pour associer les symboles aux émetteurs. Le récepteur divise le temps en segments, où chaque segment est délimité par les frontières des symboles des trames LoRa en collision. En d'autres termes, il ne peut y avoir de frontière de symbole d'aucune trame pendant un segment. Ensuite, OCT effectue une FFT sur chaque segment. Il enregistre tous les pics de chaque FFT et détermine les symboles envoyés en fonction des symboles qui se répètent sur tous les segments correspondants. Lorsque les trames en collision sont quasi-synchronisées, l'approche précédente ne peut pas être utilisée et OCT utilise plutôt le niveau de puissance des symboles pour associer les symboles à la trame correcte.

CIC [48] est similaire à OCT et utilise également le décalage temporel pour diviser chaque SD en segments, chaque segment étant soumis à une FFT. CIC calcule l'intersection de l'ensemble des symboles des segments contigus pour extraire le symbole correct.

SCLoRa [22] exploite à la fois le décalage temporel et la différence de puissance pour attribuer les symboles démodulés à la bonne trame. SCLoRa utilise des fenêtres glissantes pour identifier les frontières des symboles de chaque trame à partir du signal superposé et pour trouver le meilleur niveau de puissance d'un symbole.

La complexité de SCLoRa dépend du nombre et de la taille de ces fenêtres glissantes. À notre connaissance, SCLoRa est actuellement le protocole de résolution de collision offrant les meilleures performances.

GS-MAC est un protocole MAC à slots temporels : le temps est divisé en slots, et chaque slot est ensuite divisé en sous-slots d'une durée de SD/M_s, où SD représente la durée d'un symbole et M_s est le nombre de sous-slots par durée de symbole. Les dispositifs terminaux ne peuvent démarrer la transmission d'une trame qu'au début d'un sous-slot. GS-MAC utilise ces sous-slots pour suivre correctement les symboles des différentes trames en collision. En général, GS-MAC a des chances de décoder les trames qui sont envoyées seules dans leur propre sous-slot. Si plusieurs trames sont envoyées dans le même sous-slot, elles sont toutes indistinguables. Cependant, de telles trames n'affectent pas le décodage des trames des autres sous-slots.

A.3 Contribution 1 : Récupération de trames LoRa en utilisant le codage LoRa

Bien que plusieurs algorithmes aient été proposés pour décoder des trames en collision même dans des conditions spécifiques, il existe toujours des conditions dans lesquelles les trames sont indistinguables en raison d'incertitudes sur la valeur de certains symboles.

Par conséquent, nous avons proposé un nouvel algorithme de résolution de collisions basé sur le codage LoRa. Cet algorithme vise à réduire les incertitudes générées par la collision de plusieurs signaux LoRa, pour n'importe quel algorithme de résolution de collision. Cet algorithme est publié dans [61].

A.3.1 Codage de LoRa

Le schéma de codage LoRa comprend quatre opérations qui transforment le flux de bits généré par l'application en valeurs encodées en symboles : (1) L'émetteur ajoute d'abord de la redondance en utilisant un codage de Hamming afin de pouvoir détecter ou corriger les erreurs. (2) Ensuite, il effectue un whitening des données. (3) Puis, il applique un entrelacement diagonal, dont le rôle est de répartir les bits sur plusieurs symboles. (4) Enfin, les symboles sont encodés en utilisant un schéma de décodage de Gray. Le schéma de décodage LoRa effectue les opérations inverses, dans l'ordre inverse, afin de transformer les valeurs reçues en le flux de bits d'origine.

Codage de Hamming et détection ou correction d'erreurs. La redondance est ajoutée par l'émetteur par le codage ECC. Pour ce faire, le codage LoRa agit sur chaque demi-octet, composé de 4 bits, de la trame MAC. À l'étape d'encodage, un paramètre appelé CR, variant de 1 à 4, contrôle le nombre de bits redondants à ajouter à chaque demi-octet. Les bits redondants sont dérivés de chaque demi-octet en utilisant le codage de Hamming et concaténés avec lui pour générer un mot de code. Plusieurs CR sont possibles : CR1=4/5,

CR2=4/6, CR3=4/7 et CR4=4/8, ce qui donne des mots de code de 5 à 8 bits. Du côté du récepteur, les demi-octets peuvent être facilement récupérés à partir des mots de code pour obtenir les données d'origine. Les erreurs d'un seul bit sur chaque mot de code peuvent être détectées avec CR1 et CR2, et peuvent être corrigées avec CR3 et CR4.

Whitening. L'opération de whitening (ou de-whitening) est réalisée en effectuant une opération XOR avec une séquence spécifique sur chaque mot de code. Cette opération est appliquée pour éviter de longues séquences continues composées uniquement de 0, ou uniquement de 1.

Entrelacement. LoRa utilise un entrelacement pour répartir les bits encodés dans plusieurs symboles, de sorte qu'une erreur sur quelques bits consécutifs se traduise par une erreur d'un bit sur plusieurs mots de code, ce qui est plus facile à récupérer grâce à la redondance. L'entrelacement utilisé dans LoRa prend en entrée SF mots de code, ayant chacun $CR+4$ bits. Ces bits sont organisés dans une matrice de $SF \times (CR+4)$ bits, où chaque mot de code est une ligne. L'entrelacement génère une matrice de $(CR+4) \times SF$ bits en sortie, où chaque ligne représente SF bits, utilisés comme valeur de symbole à l'étape suivante. La matrice de sortie est remplie ligne par ligne. La i-ème ligne de sortie (avec $i \in [1; CR+4]$) correspond à la $(CR+4+1-i)$-ème colonne de la matrice d'entrée. Les bits sont remplis dans la direction ascendante (avec retour à la ligne), en commençant par le bit à la ligne $(i-2) \mod SF+1$. La matrice est nommée bloc dans la suite de cette section.

A.3.2 Contexte et incertitudes

Supposons que deux trames entrent en collision. En raison du mécanisme d'accès aléatoire ALOHA, il est très probable que les signaux LoRa soient désynchronisés.

Dans le cas où le décalage temporel entre les trames n'est pas un multiple de la durée du symbole, comme illustré sur la figure A.5, on peut observer que les frontières des symboles dans chaque trame ne sont pas alignées.

Pour illustrer l'effet du décalage de la composante de fréquence, supposons que le décalage temporel est donné par $\tau_\sigma = \frac{\sigma}{M} SD$. Dans ce cas, la première équation dans l'équation A.1 devient :

$$\begin{aligned} f_u^m(t+\tau_\sigma) &= \frac{M}{SD^2}(t+\tau_\sigma+\tau_m) \\ &= \frac{M}{SD^2}(t+\frac{m}{M}SD) + \frac{M}{SD^2}\frac{\sigma}{M}SD \\ &= f_u^m(t) + \frac{\sigma}{SD}. \end{aligned} \tag{A.7}$$

La valeur de symbole estimée de la fréquence correspondante devient $m' = m + \sigma$ au lieu de m. La même dérivation s'applique également au deuxième cas de l'équation A.1.

Par exemple, supposons que deux trames f_1 et f_2 sont envoyées par deux émetteurs et sont désynchronisées de $SD/2$. Les valeurs des symboles de f_2 observées par le récepteur sont décalées de $\frac{2^{SF}}{2}$. Pendant chaque symbole de f_1, le récepteur va observer une seule valeur pour f_1 et deux valeurs pour f_2. Par exemple, pendant le symbole 1 de f_1, le récepteur détectera la valeur correcte pour f_1, ainsi que deux valeurs pour f_2 correspondant aux symboles 1 et 2.

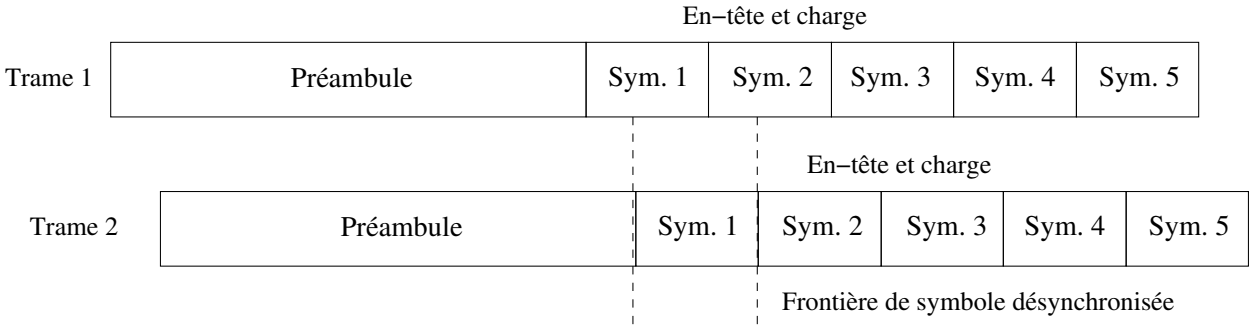


Figure A.5: Cas 1 (symboles désynchronisés) : Exemple de deux trames désynchronisées en collision. Le trame 2 arrive plus tard que le trame 1, avec un retard d'environ $SD/3$.

Dans le cas où le décalage temporel entre les trames est un multiple de la durée du symbole, comme illustré sur la figure A.6, On peut observer que les frontières des symboles dans chaque trame sont alignées. Il convient de noter que le cas où les trames sont complètement synchronisées dès le début des trames est similaire à ce deuxième cas. Le récepteur, en utilisant un algorithme de résolution de collision, va détecter deux valeurs possibles pour chaque symbole, l'une correspondant au symbole de f_1, et l'autre correspondant au symbole de f_2. Cela peut générer des incertitudes dans l'ensemble de la trame, si le récepteur n'est pas en mesure d'associer chacun des deux symboles à la bonne trame.

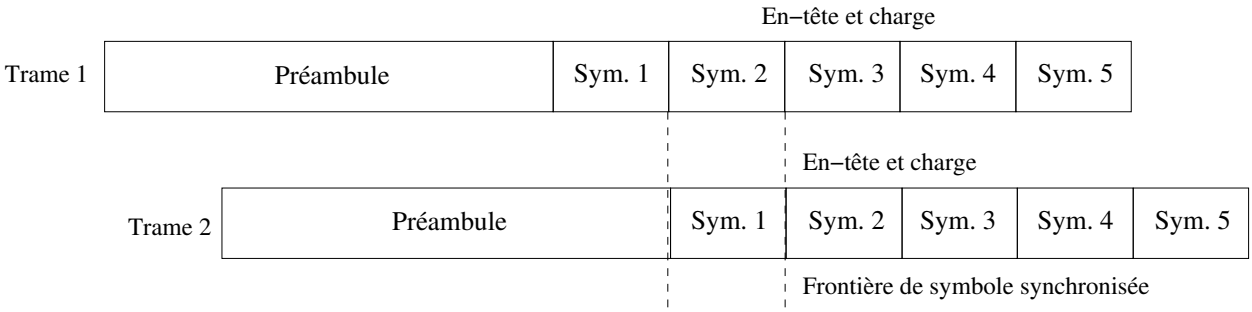


Figure A.6: Cas 2 (symboles synchronisés) : Exemple de deux trames désynchronisées en collision, avec des frontières de symboles qui se chevauchent. Le trame 2 arrive plus tard que le trame 1, avec un décalage temporel égal à SD.

Dans les deux cas, en fonction de l'algorithme utilisé et de ses caractéristiques, le récepteur peut être en

mesure d'associer les différentes valeurs de symboles aux trames correspondantes, c'est-à-dire, le récepteur peut être en mesure de distinguer les trames. Par exemple, SIC est capable de distinguer les trames en ignorant les symboles de la trame la plus faible. CHOIR peut résoudre conditionnellement la collision en utilisant les petits décalages de fréquences cachés dans les symboles de f_1 et f_2 (typiquement, si les deux émetteurs ont de petits décalages de fréquences différents).

Cependant, certaines trames restent indistinguables lorsqu'elles ont des caractéristiques similaires. Par exemple, dans SIC, cela se produit lorsque les trames sont reçues avec une puissance similaire et avec des symboles synchronisés. Dans CHOIR, cela se produit lorsque les deux émetteurs ont un petit décalage de fréquences similaire. Dans GS-MAC, cela se produit lorsque les trames sont envoyés dans le même slot et sous-slot.

La synchronisation des trames n'est pas le seul facteur qui apporte des incertitudes. En effet, les algorithmes basés sur le temps ne peuvent pas associer correctement les symboles aux trames en collision lorsque leurs frontières de symboles sont presque synchronisées, même si le début des trames n'est pas synchronisé. Les algorithmes basés sur la puissance ne sont pas capables de capturer une trame lorsque plusieurs trames sont reçues avec une puissance similaire. Le bruit ou des interférences courtes peuvent également apporter des incertitudes sur un petit nombre de symboles, rendant les trames indécodables.

A.3.3 Algorithme proposé

L'algorithme que nous proposons permet de résoudre de nombreux cas d'incertitudes sur les symboles. Notre algorithme est basé sur le codage LoRa, et se compose de deux étapes : Tout d'abord, toutes les séquences de codage possibles sont générées et validées en utilisant les bits redondants contrôlés par CR ; ensuite, toutes les trames possibles sont générées et validées en utilisant le CRC.

La première étape de notre algorithme se concentre sur chaque bloc de symboles indépendamment. S'il y a des incertitudes dans un bloc de symboles, c'est-à-dire plusieurs valeurs de symboles possibles, le décodeur génère tous les blocs potentiels et tente de les désentrelacer. Ensuite, il calcule les bits redondants attendus, appelés *compECC*, à partir des bits de données et les compare avec les bits redondants de la trame LoRa reçue, appelés $refECC$. Les bits redondants encodés $refECC$ doivent correspondre à *compECC* pour tous les mots de code dans un bloc potentiel. Si les bits diffèrent, ce bloc de symboles candidat est rejeté.

La deuxième étape de notre algorithme prend en entrée les blocs validés. Il est important de noter que chaque bloc occupe une position dans la trame LoRa à laquelle il appartient. Cette position ne change pas pendant l'entrelacement ou le désentrelacement. Ainsi, après la validation des mots de code, les données encodées dans les mots de code à chaque position peuvent être récupérées et assemblées pour obtenir les données possibles pour la couche supérieure, qui sont les trames MAC. Le CRC des trames MAC possibles, encodé dans les deux derniers octets de la trame MAC, peut être calculé à partir des données dans les trames

MAC. Nous pouvons réduire davantage le nombre de trames MAC possibles en comparant le CRC calculé à partir des données avec le CRC récupéré des trames MAC.

A.4 Contribution 2 : SF-DS

La contribution essentielle de cette thèse est appelée SF-DS [64]. SF-DS est une nouvelle méthode de décodage proposée pour améliorer les performances des réseaux LoRaWAN en cas de collisions.

L'algorithme SF-DS s'inspire de l'algorithme GS-MAC[13]. Cependant, SF-DS élimine les principales limitations de GS-MAC, qui sont : (1) le besoin d'un oracle pour déterminer les fréquences instantanées, et (2) le mécanisme de slot. Étant libre des slots, SF-DS est compatible avec les dispositifs terminaux LoRaWAN puisqu'il peut gérer des trames transmises à n'importe quel instant, contrairement à GS-MAC. De plus, SF-DS peut bénéficier des spécificités du schéma de codage LoRa pour résoudre des incertitudes supplémentaires[61], comme expliqué dans la section A.3.

Pour supprimer les slots et sous-slots, SF-DS introduit la notion d'intervalle et de sous-slot virtuel. Un intervalle est une division de la période SD en M_s valeurs. Les intervalles sont synchronisés avec une trame de référence, qui est la première trame détectée, mais pas avec les autres trames en collision. Un sous-slot virtuel est un regroupement de deux intervalles : les intervalles $2.M_s - 1$ et 0 correspondent au sous-slot virtuel 1 (semblable au sous-slot 1 de GS-MAC), les intervalles 1 et 2 correspondent au sous-slot virtuel 2 (semblable au sous-slot 2 de GS-MAC), et ainsi de suite.

Décodage avec un canal parfait. Dans SF-DS, le récepteur vérifie constamment les préambules. Le premier préambule détecté devient le préambule de référence, et permet au récepteur de se synchroniser avec la trame. Lorsque de nouveaux préambules sont détectés, le récepteur stocke le délai relatif entre le nouveau préambule et le préambule de référence. Le délai relatif de chaque trame est traduit en l'un des $2.M_s$ intervalles, puis en l'un des sous-slot virtuel. Lorsqu'un symbole s_{dec} est obtenu à la frontière d'un sous-slot virtuel, le symbole d'origine s_{org} peut être calculé en fonction du délai temporel relatif d de la trame et de la frontière d'un sous-slot virtuel, en utilisant $s_{org} = s_{dec} - (2^{SF}.d/SD)$.

La figure A.7 montre deux exemples de trames retardées. En haut, le retard est égal à $d = (1/4).(SD/M_s)$, tandis qu'en bas, le retard est $d = (3/4).(SD/M_s)$, les deux avec $M_s = 4$ sous-slots virtuels. Dans les deux cas, le retard correspond à un demi-intervalle, ce qui est le pire des cas pour SF-DS. Les sous-figures de gauche montrent les FFT pour le sous-slot virtuel 1, et les sous-figures de droite montrent les FFT pour le sous-slot virtuel 2. Pour les sous-figures supérieures, le retard est de $SD/16 \in [0; SD/8)$, de sorte que la deuxième trame est considérée comme étant reçue dans l'intervalle 0. Ainsi, notre algorithme effectue les FFT comme si les deux trames étaient complètement synchronisées. Les pics de FFT dans le sous-slot virtuel 1 sont 64 et 104, ce qui correspond à un symbole original de la deuxième trame égal à $64 - 8 = 56$

ou $104 - 8 = 96$ (qui est la valeur correcte), en raison du retard d. Pour les sous-figures inférieures, le retard est de $3.SD/16 \in [SD/8; SD/4)$, de sorte que la nouvelle trame est considérée comme étant reçue dans l'intervalle 1. Ainsi, notre algorithme effectue les FFT comme si les deux trames étaient envoyées sur différents sous-slots virtuels. Les pics de FFT dans le sous-slot virtuel 2 sont 96 et 120, ce qui correspond à un symbole original de la deuxième trame égal à $96-24 = 72$ ou $120-24 = 96$ (qui est la valeur correcte). Dans les deux cas, la valeur correcte du symbole a été récupérée avec succès. Les pics de FFT d'une trame retardée sont généralement plus faibles que les pics de FFT de la trame de référence, en raison du retard. Cela peut être observé sur le deuxième pic de la sous-figure supérieure gauche de la figure A.7. En effet, même si la deuxième trame a été transmise peu de temps après la première trame, elle est toujours considérée comme étant transmise dans le même sous-slot virtuel en raison du mécanisme d'arrondi du retard, introduit pour supprimer la synchronisation. Ce pic de FFT plus faible est facilement influencé par les lobes latéraux de la première trame, et est donc susceptible d'être incorrectement décodé. Afin de faire face à cela, nous utilisons également les résultats de FFT du sous-slot virtuel suivant (affiché sur la sous-figure supérieure droite de la figure A.7), car ces résultats sont moins influencés par ce retard et sont donc plus forts. Par exemple, le démodulateur pourrait être incapable de déterminer si le pic de la trame retardée dans le sous-slot virtuel 1 est 104 (qui est la valeur correcte), mais obtiendrait la valeur correcte pendant le sous-slot virtuel 2.

Décodage avec un canal imparfait. Concentrons-nous à présent sur le décalage temporel subi par une trame LoRa. Rappelons que le taux d'échantillonnage d'un récepteur LoRa est de $F_s = BW$, c'est-à-dire que le récepteur prend F_s échantillons par seconde. Le temps d'échantillonnage est donc de $\frac{1}{F_s}$ seconde, ce qui est également la résolution temporelle avec le taux d'échantillonnage donné. Considérons qu'il y a une seule trame LoRa. Étant donné que le récepteur LoRa doit se synchroniser avec les frontières des symboles de la trame LoRa, le décalage temporel jusqu'à la frontière du symbole peut être divisé en deux parties : le décalage temporel entier et le décalage temporel fractionnel, comme le montre la figure A.8 sur un exemple simplifié (avec un taux d'échantillonnage très faible). Le décalage temporel entier est proportionnel au temps d'échantillonnage et peut être compensé après avoir été détecté pendant le préambule. Cependant, le récepteur n'est pas en mesure de compenser le décalage temporel fractionnel en raison de la résolution temporelle limitée. Cette partie fractionnelle cause un décalage temporel fractionnel. Dans ce qui suit, nous notons par T_{off} le décalage de temps entier, et par t_{off} le décalage de temps fractionnel. Ainsi, le décalage de temps total est $\tau_{sigma} = T_{off} + t_{off}$. Dans ce qui suit, nous discutons de la façon dont SF-DS vise à résoudre les décalages de temps entiers et fractionnels, pendant le préambule.

Lorsqu'un récepteur reçoit la première trame LoRa, il n'est pas encore synchronisé avec une trame, comme illustré sur la figure A.8. Pour une meilleure synchronisation, le récepteur doit utiliser un taux d'échantillonnage plus élevé (typiquement $k.BW$, où $k > 1$ est un facteur de sur-échantillonnage), afin de réduire les erreurs de synchronisation dues à l'intervalle d'échantillonnage large. Le récepteur doit donc

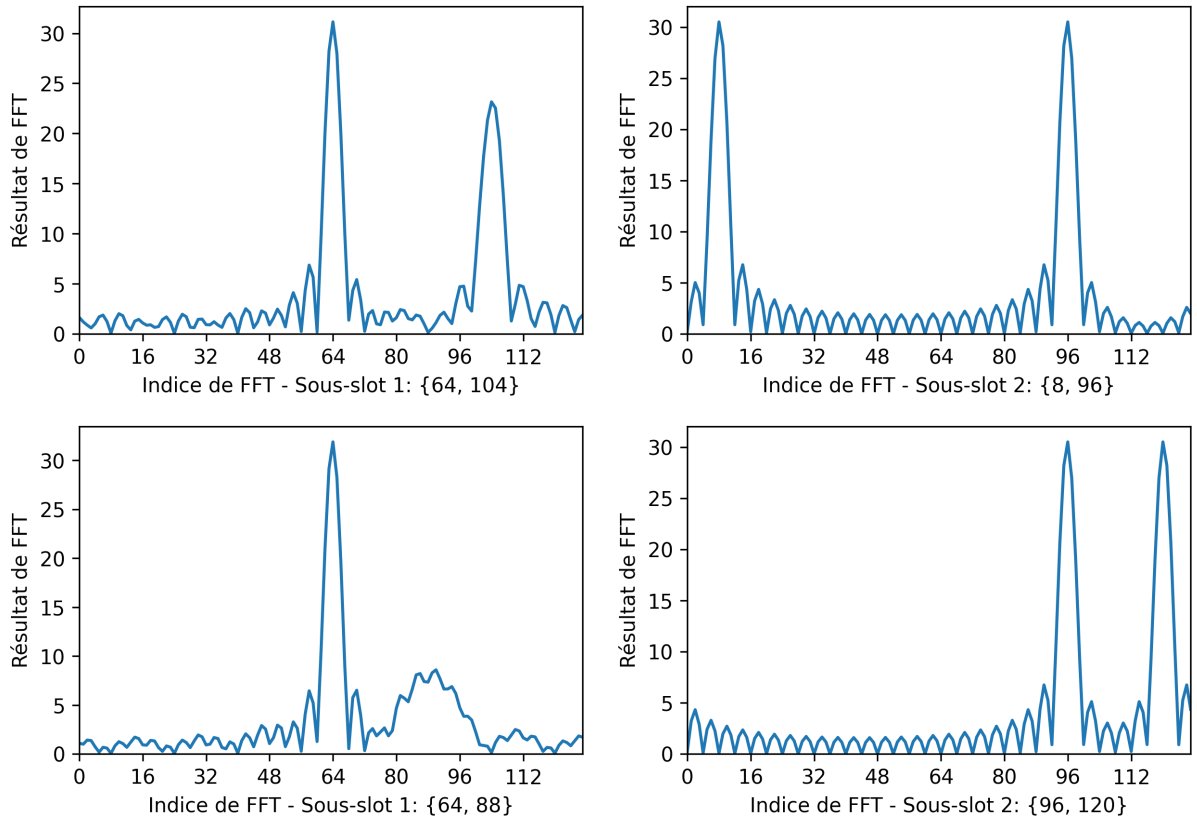


Figure A.7: Résultats de la FFT dans le sous-slot virtuel 1 (en haut à gauche et en bas à gauche) et le sous-slot virtuel 2 (en haut à droite et en bas à droite) pour deux symboles qui sont désynchronisés d'un quart de sous-slot virtuel (en haut à gauche et en haut à droite) ou de trois quarts de sous-slot virtuel (en bas à gauche et en bas à droite). Les valeurs du premier symbole sont 48, et la valeur du deuxième symbole, qui est retardé, est 96. SF vaut 7, $BW = 125$ kHz, et il y a quatre sous-slots virtuels (correspondant à huit intervalles).

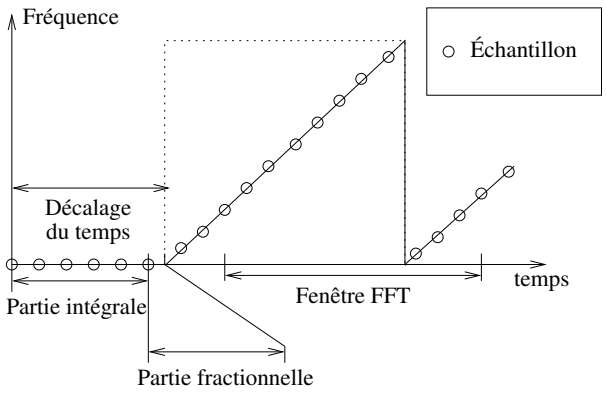


Figure A.8: LoRa utilise plusieurs échantillons pour calculer les symboles. Le décalage temporel entre le symbole déduit des échantillons observés et le symbole réel peut être divisé en deux parties : un décalage temporel entier et un décalage temporel fractionnel.

décaler la fenêtre FFT dans SD, avec moins de $k.M$ échantillons, où M est le nombre minimum d'échantillons pour décoder un symbole après la synchronisation, en utilisant un taux d'échantillonnage égal à BW. Ce décalage permet au récepteur de se synchroniser avec les frontières de symboles, c'est-à-dire d'utiliser les frontières de symboles comme référence. En effet, le décalage de temps τ_{sigma} modifie la valeur du symbole de σ, selon l'équation A.7 (si nous ne considérons pas les décalages de fréquences). Par exemple, si la valeur de symbole démodulée au début du préambule est 48 (alors qu'elle devrait être 0), le récepteur doit avancer la fenêtre FFT de 48 échantillons pour compenser le décalage de temps et s'aligner sur la véritable frontière de symbole.

Lorsqu'un récepteur détecte une nouvelle trame en collision avec la trame actuelle, il n'est plus en mesure d'ajuster la fenêtre FFT pour la deuxième trame en raison de la synchronisation avec la première trame uniquement. Dans SF-DS, le décalage de temps entier est utilisé pour allouer les trames entrantes au sous-slot virtuel correspondant. La stratégie consiste à allouer la trame au sous-slot virtuel le plus proche. Le calcul de l'index du sous-slot virtuel le plus proche i, dont l'index varie de 0 à $M_s - 1$, peut être décrit comme suit :

$$i = \left\lfloor (T_{off} + \frac{1}{2}\frac{SD}{M_s})\frac{M_s}{SD} \right\rfloor \mod M_s. \tag{A.8}$$

De plus, SF-DS a besoin de connaître le début des trames entrantes en termes de symboles. De même, le décalage de temps entier aide à calculer le nombre d'échantillons et le nombre de symboles entre les préambules ΔN_s, comme suit:

$$\Delta N_s = \left\lfloor (T_{off} + \frac{1}{2}\frac{SD}{M_s})\frac{1}{SD} \right\rfloor. \tag{A.9}$$

Alors que le décalage temporel fractionnel a peu d'influence lorsque le récepteur est synchronisé avec une seule trame (comme c'est le cas avec le LoRa traditionnel), le fait que SF-DS ne peut pas se synchroniser avec toutes les trames en collision augmente l'impact du STO sur toutes les trames, sauf la première. Rappelons

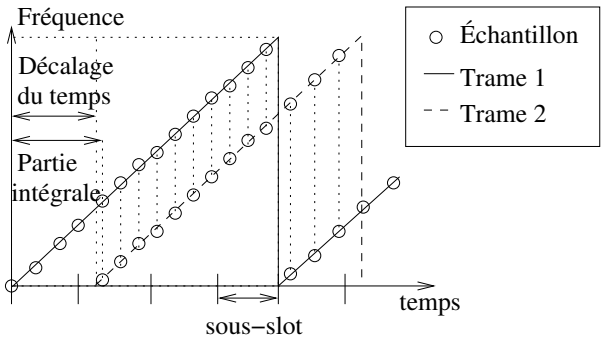


Figure A.9: Puisque SF-DS est synchronisé avec la première trame reçue, le STO de toutes les autres trames en collision ne peut pas être ignoré.

que la démodulation utilise une fréquence d'échantillonnage égale à BW, qui est généralement inférieure à la fréquence d'échantillonnage lors de la détection du préambule. En effet, les STO possibles des trames non synchronisées deviennent plus grands en raison de la fréquence d'échantillonnage plus faible après la synchronisation, ce qui entraîne des intervalles plus longs entre les échantillons. Un tel STO peut changer la valeur du symbole d'au plus un, à condition que $|t_{off}| > 0.5$.

Dans SF-DS, nous ne prenons en compte le STO qu'en arrondissant la partie entière T_{off}. En effet, cette considération donne la propriété que $|t_{off}| \leq \frac{1}{2k.F_s}$ pour la trame synchronisée, car les trames sont synchronisées en utilisant une fréquence d'échantillonnage supérieure $k.F_s$. Pendant ce temps, le décalage temporel pour les trames en collision suivantes dans SF-DS est limité à $|t_{off}| \leq \frac{1}{2F_s}$, car le récepteur utilise une fréquence d'échantillonnage inférieure F_s pour le décodage. Tout $|t_{off}| > 0,5$ contribue à une valeur ± 1 pour T_{off}.

A.5 Contribution 3 : Simulateur de couche physique LoRa

Les simulateurs LoRa et LoRaWAN permettent aux chercheurs d'étudier les performances des applications et des dispositifs terminaux IoT basés sur LoRa sans avoir besoin de matériel physique, étant donné que ce matériel peut être coûteux et qu'une modification à LoRa peut être complexe à mettre en place. Il existe plusieurs simulateurs avec différents niveaux de simulation et de granularité, comme détaillé dans [29].

Pour les besoins de cette thèse, la couche physique LoRa doit être modélisée avec précision. Il faut noter que le développement fin de la couche physique est difficile en raison du fait que la modulation LoRa est propriétaire. Mais étant donné que plusieurs chercheurs ont fait de la rétro-ingénierie sur la modulation, la plupart des détails de la modulation LoRa sont connus.

Dans cette section, nous détaillons notre implémentation d'un simulateur de couche physique LoRa, pouvant intégrer des éléments de LoRaWAN.

A.5.1 Simulateur GNU Radio

GNU Radio est un outil qui permet de développer des algorithmes de traitement du signal pour les communications radio. L'élément de base dans GNU Radio est le bloc. Un bloc contient généralement des ports d'entrée et de sortie, à travers lesquels il peut respectivement recevoir et envoyer des données à d'autres blocs. Le graphe de flux est la structure de données de base dans GNU Radio et représente les connexions entre les blocs. Les données peuvent être un flux continu (par exemple, un signal est souvent représenté sous la forme d'un flux de nombres flottants complexes représentant les composantes I et Q du signal), ou un message discret contenant des données de type polymorphe (c'est-à-dire une structure générique utilisée pour contenir une grande variété de données telles qu'un nombre, une chaîne de caractères, un tuple, une liste ou un dictionnaire). Un bloc peut attacher des balises (un autre type de message discret) aux données en flux. Les balises peuvent contenir des informations supplémentaires, telles que le marquage de l'arrivée d'une nouvelle trame, la fréquence des signaux transmis, etc.

GNU Radio dispose d'une interface graphique intuitive où les blocs sont interconnectés via des liens entre les ports de sortie et d'entrée. Le port d'entrée accepte les données en flux, tandis que les deux ports de sortie envoient respectivement des données en flux et des messages discrets. Par exemple, dans GNU Radio, les ports bleus utilisent un flux de nombres flottants complexes, tandis que les ports blancs représentent des messages.

La figure A.10 montre un exemple où la source de données est un fichier (bloc "File Source") ou un matériel SDR connecté (bloc "UHD: USRP Source", où UHD signifie que le bloc est alimenté par le pilote de matériel USRP). La source est connectée à un bloc "Throttle" pour limiter le débit des données du flux. La sortie du bloc "Throttle" est connectée à l'entrée du bloc "Resampler" pour modifier le taux d'échantillonnage du flux de données. Après le bloc "Resampler", la partie "Decoder" est l'un des composants principaux du simulateur que nous avons proposé pour implémenter les algorithmes de résolution de collisions.

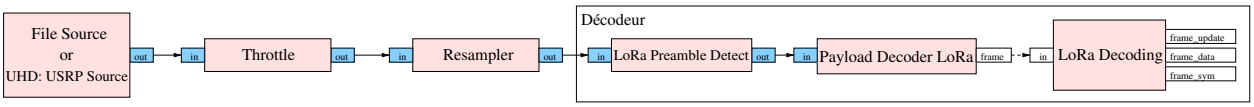


Figure A.10: Décodage des trames LoRa à partir d'un échantillon de signal, obtenu à partir d'un fichier ou d'un matériel USRP SDR.

Dans notre implémentation, le récepteur commence par prendre les échantillons à partir d'un bloc source. Le bloc "Détection du préambule LoRa" traite les échantillons afin de trouver un préambule. Les échantillons sont transmis en sortie. Cependant, dès qu'une trame est détectée, le détecteur de préambule attache une étiquette contenant les informations de la trame correspondante. L'étiquette est un dictionnaire implémenté par un type polymorphe, qui contient un ensemble de paires clé-valeur. La position de l'étiquette sur les échantillons marque le début des symboles de la charge utile. Il contient :

- l'identifiant de la trame détectée par le détecteur de préambule actuel, qui s'incrémente automatiquement lorsqu'une nouvelle trame potentielle apparaît,

- la longueur de la trame détectée, exprimée en symboles[3],

- le SF et le décalage de fréquence de la trame détectée.

Chaque valeur est un tableau, car la présence de plusieurs trames est possible lorsque plusieurs trames entrent en collision.

Les échantillons, combinés aux informations de préambule de la trame détectée, sont ensuite traités par le bloc "Décodeur de charge utile" afin de synchroniser le récepteur avec les symboles transmis. Dans l'exemple de la figure A.10, l'algorithme de décodage standard LoRa est utilisé. Le bloc surveille en permanence les étiquettes transmises par les blocs précédents (c'est-à-dire, par le détecteur de préambule). Lorsqu'une étiquette est détectée et qu'aucune trame n'est actuellement synchronisée, le décodeur de charge utile utilise la position de l'étiquette comme frontière de symbole pour la synchronisation avec la trame. Le bloc commence alors à démoduler les symboles. Enfin, lorsque huit symboles démodulés sont disponibles (correspondant à l'en-tête) ou lorsque la trame entière est démodulée (si la longueur de la trame en symboles est connue), un message discret est envoyé au bloc de décodage de trame. Ce message discret est similaire aux étiquettes mentionnées précédemment, et contient l'ID de la trame, la longueur de la trame et le SF. De plus, les valeurs des symboles démodulés sont également regroupées dans un tableau. Bien que le schéma de démodulation standard LoRa ne prenne en compte que l'indice du pic le plus fort pendant chaque SD, la mise en œuvre d'autres algorithmes peut également produire plusieurs valeurs de symboles possibles pour chaque SD. Lorsque seuls les huit premiers symboles sont transmis en tant que message au bloc "Décodage LoRa", le bloc tente d'abord de décoder le contenu de l'en-tête. En mode explicite, l'en-tête contient la longueur de la trame, la présence d'un CRC et le CR utilisé pour coder la trame. La longueur de la trame décodée peut ensuite être publiée en tant que rétroaction pour informer le bloc "Décodeur de charge utile" ou détecteur de préambule, via le port `frame_update`, afin de déterminer la fin de la trame donnée. Après avoir démodulé tous les symboles de la trame entière, ce bloc décode et produit les données d'origine à partir des symboles. Ces données sont publiées via le port `frame_data`.

Nous avons implémenté plusieurs algorithmes dans le simulateur : LoRa, CHOIR, FTrack et OCT, selon les descriptions faites dans les articles correspondants, ainsi que SF-DS. Toutes ces implémentations aident à comparer SF-DS avec les autres algorithmes dans les mêmes scénarios et les mêmes conditions.

[3]Il convient de noter que la longueur est généralement inconnue pendant la détection du préambule, car elle est généralement décodée à partir de l'en-tête.

A.5.2 Abstraction du nœud

GNU Radio permet de générer directement des données à partir d'une source de signal interne ou à partir de sources externes telles qu'un fichier ou un matériel SDR. Nous avons donc mis en œuvre une abstraction de nœuds LoRa afin de simuler différents scénarios de grands réseaux LoRa.

La figure A.11 présente les deux abstractions de nœud de notre implémentation, représentées par Nœud 1 et Nœud 2, et qui simulent deux dispositifs terminaux dans un réseau LoRa. Les abstractions de nœuds sont également utilisées pour générer les données applicatives. Ces données peuvent être générées de manière aléatoire dans GNU Radio à des fins de test (comme dans Nœud 1) ou à partir d'une application externe (comme dans Nœud 2 pour un exemple avec UDP), telles qu'un simulateur de réseau. Il faut noter que la communication entre l'application externe et GNU Radio peut se faire via des connexions TCP, UDP ou ZeroMQ [18]. Les données générées sont encapsulées dans un message et transmises à notre bloc "LoRa Encoding", qui effectue le codage et la modulation, et produit des signaux LoRa.

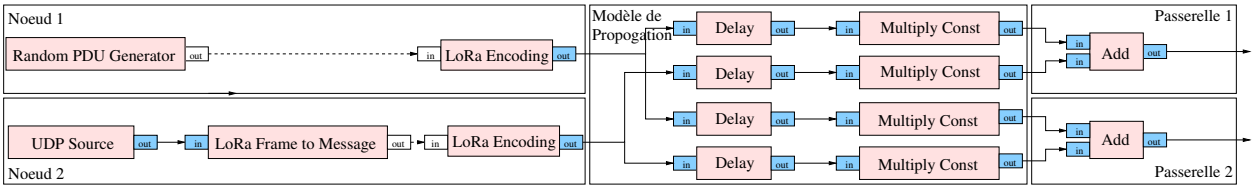


Figure A.11: Génération de signaux avec deux abstractions de nœuds et deux passerelles.

Réalisation de plusieurs dispositifs terminaux. Par défaut, les signaux générés par notre bloc "LoRa Encoding" ont une amplitude normalisée et sont parfaitement synchronisés au niveau de l'échantillon. Il est possible de mettre en œuvre un modèle de propagation en retardant les signaux (voir les blocs "Delay" sur la figure A.11, qui peuvent être configurés avec un nombre donné d'échantillons) et en réduisant la puissance du signal reçu avec un facteur donné (voir les blocs "Multiply Const" sur la figure A.11). Ensuite, les signaux des deux dispositifs terminaux sont ajoutés, et le résultat est un signal en collision. Il est important de noter qu'une configuration très similaire peut être mise en place pour évaluer l'impact d'un faible rapport signal sur bruit (SNR), en ajoutant un bruit aléatoire important à un signal faible. Il faut également noter que ce retard est également utilisé pour mettre en œuvre des transmissions désynchronisées, requises par plusieurs algorithmes de résolution de collisions existants. Dans le scénario simulé, l'utilisateur peut supprimer le bloc "LoRa Preamble Detect". Dans une telle situation, les informations de trame sont générées par le bloc "LoRa Encoding" et ensuite transmises dans les étiquettes, avec les échantillons des signaux de la bande de base.

Mise en œuvre de plusieurs passerelles. Plusieurs passerelles peuvent être mises en œuvre en ajoutant des blocs de détection de préambule et de décodage de charge utile pour chaque passerelle. Pour chaque passerelle, les paramètres correspondant au modèle de propagation sont susceptibles d'être différents, car les signaux reçus par chaque passerelle subiront des pertes de canal différentes. Par conséquent, nous pouvons définir différents paramètres dans les modèles de propagation en les calculant à l'avance en fonction des distances relatives entre les dispositifs terminaux et les passerelles. Cependant, il reste difficile de mettre en place un réseau à grande échelle manuellement en raison de la complexité de la construction des abstractions de nœuds de chaque dispositif terminal vers toutes les passerelles.

Mise en œuvre de fonctionnalités avancées. Le bruit est souvent considéré comme un bruit blanc gaussien additif avec une moyenne nulle [16]. Dans notre simulateur, un tel bruit peut être simulé à l'aide de la source de bruit "Noise Source", et ajouté aux signaux superposés. L'amplitude relative est calculée comme suit :

$$A_{bruit} = A_{ref} \times 10^{-\frac{SNR}{20}}. \tag{A.10}$$

Pour simuler un SNR de 0 dB, A_{bruit} est réglé sur la même valeur que A_{ref}.

Le problème de fréquence porteuse non nulle, aussi appelé CFO, n'est pas négligeable dans une transmission réelle générée par du matériel bon marché. Pour simuler le CFO dans le signal de bande de base, un CFO peut être ajouté au bloc "LoRa Encoding" en tant que partie de l'abstraction de nœud. Le CFO est traduit en un décalage de fréquence initial dans le bloc "LoRa Encoding".

Le décalage temporel de symbole, appelé STO, peut se produire lorsque le ré-échantillonnage du côté récepteur n'est pas parfaitement synchronisé avec le signal d'origine.

Script pour lancer des simulations qui passent à l'échelle. Les grands réseaux sont généralement difficiles à simuler car ils nécessitent la création et l'interconnexion de plusieurs blocs, ce qui est fastidieux et sujet aux erreurs. Pour faciliter la configuration de simulations de grande envergure, nous avons développé un script avec un fichier de configuration optionnel permettant de générer automatiquement un réseau complexe. Le script accepte un ensemble de paramètres pour les dispositifs terminaux et les passerelles. Ces paramètres peuvent contrôler la durée totale de la simulation et le taux d'échantillonnage utilisé. Le bloc "LoRa Encoding" est contrôlé par des paramètres tels que le cycle d'activité, le taux d'échantillonnage, l'intervalle entre les trames et le SF pour générer des signaux LoRa à partir des données des trames. Les données des trames sont générées de manière aléatoire et leurs longueurs sont également fournies en tant que paramètre du script. Un fichier de configuration peut également être inclus pour stocker des informations de déploiement telles que l'emplacement des dispositifs terminaux ou des passerelles.

A.5.3 Simulateur de NS-3

Nous avons aussi modifié la réalisation initiale du module NS-3, décrite dans [9] et [28] se concentrait principalement sur la couche MAC LoRaWAN, qui comprend l'accès au canal et la contrainte du cycle d'activité, ainsi que les communications entre les dispositifs terminaux, les passerelles et le serveur réseau. La couche physique LoRa était modélisée de manière simpliste, avec une fonction `Send` acceptant les données de trame LoRaWAN et les paramètres de transmission. Les auteurs avaient décrit plusieurs configurations de la couche physique LoRa, notamment le SF, la fréquence du canal et la puissance de transmission. Une passerelle et un dispositif terminal disposaient de leurs propres fonctions `Send`, définies respectivement dans `simple-gateway-lora-phy.cc` et `simple-end-device-lora-phy.cc`. La partie commune des deux réalisations était le calcul de la durée de la trame. Cette durée était ensuite utilisée comme paramètre pour la transmission vers le canal, en combinaison avec les paramètres de la couche supérieure. Le canal devait trouver le récepteur, calculer le temps de réception et atténuer la puissance de transmission. La figure A.12 montre les informations utilisées entre les différentes couches. Du côté du récepteur, la couche physique utilisait le temps superposé pour détecter les collisions de trames. Elle calculait les niveaux de puissance accumulées en fonction des SF des trames en collision. Une trame avec un SF donné était considérée comme perdue si l'un des niveaux de puissance accumulée dépassait les seuils correspondants donnés dans [11]. Sinon, la trame était transmise à la couche supérieure et renvoyée au serveur réseau pour un traitement ultérieur. De plus, le module limitait également la capacité de décodage simultanée à huit, comme expliqué dans [47].

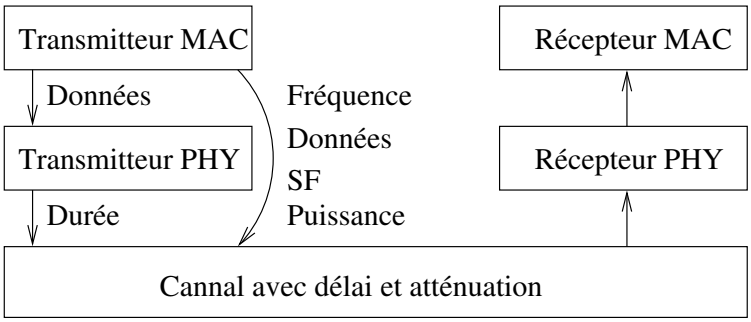


Figure A.12: Modélisation du canal, abstraction de la couche physique et abstraction de la couche MAC de LoRa et LoRaWAN dans le module NS-3 de [9] et [28]. Dans cette réalisation initiale, la couche physique ne faisait que calculer et transmettre la durée de la trame au modèle de canal, afin de déterminer si les trames entraient en collision ou non.

Notre principale contribution a été d'implémenter le codage LoRa, en nous basant sur les connaissances issues des travaux de rétro-ingénierie existants. Nous avons ajouté les fichiers "real-gateway-lora-phy.cc" et "real-end-device-lora-phy.cc" pour remplacer la couche physique LoRa initialement peu réaliste. Les trames

encodées contenant les symboles LoRa sont transmises au canal. Du côté du récepteur, la couche physique permet à différents algorithmes de résolution de collisions de décoder les données des trames à partir des symboles. Si une trame est décodée sans erreur, elle est transmise à la couche supérieure. Sinon, la trame est rejetée. En effet, les blocs "LoRa Encoding" et "LoRa Decoding" de GNU Radio partagent un code similaire avec ces réalisations physiques dans NS-3, ce qui montre que la réalisation dans NS-3 est réaliste.

Trois algorithmes de résolution de collisions sont mis en œuvre dans les simulations que nous avons décrites dans [61] pour évaluer les performances de nos algorithmes : CHOIR, SIC et GS-MAC. Par conséquent, nous avons dû modéliser les fonctionnalités requises par ces algorithmes. Pour CHOIR, les trames sont distinguées en utilisant de petits décalages de fréquence, le simulateur peut donc ajouter un petit décalage de fréquence aléatoire en tant que paramètre associé à un dispositif. Lors du décodage, de tels décalages de fréquence sont explorés pour déterminer si une trame peut être entièrement décodée. Pour SIC, après l'atténuation, la puissance de réception est exploitée pour chaque trame afin de confirmer si elle peut être récupérée à partir de la collision. Pour GS-MAC, la couche physique calcule les valeurs des symboles à chaque sous-slot en combinant les valeurs des symboles et les décalages temporels des différentes trames en collision. Si les conditions correspondantes ne sont pas remplies, la couche physique compose les symboles des différentes trames, ce qui entraîne des incertitudes potentielles.

A.6 Résultats

Résultats concernant l'algorithme de résolution des incertitudes basé sur le codage LoRa. Nous utilisons le simulateur NS-3 pour évaluer la performance de notre algorithme de récupération de trames en collision en utilisant le codage LoRa. La topologie du réseau comprend un NS, une seule passerelle et plusieurs dispositifs terminaux répartis uniformément dans un rayon d'un kilomètre autour de la passerelle. La passerelle est simulée comme un matériel SX1301 [47] avec huit voies de démodulation, ce qui permet de décoder jusqu'à huit trames simultanément. Toutes les voies de démodulation sont configurées pour décoder les trames avec le même SF. La simulation dure une heure. Nous avons fait varier le nombre de dispositifs terminaux de 400 à 2000 pour SF7 et de 200 à 1000 pour SF12. Le CR pour le codage des trames LoRa est fixé à 1, ce qui donne la plus petite longueur de trame mais la plus faible amélioration de performance pour notre algorithme. Chaque dispositif terminal envoie une seule trame avec une charge utile aléatoire de 10 octets, avec un délai aléatoire depuis le début de la simulation. Après l'application du codage LoRa, chaque trame contient 28 symboles pour SF7 et 38 symboles pour SF12. Les trames ne nécessitent pas d'accusé de réception.

La figure A.13 montre le taux moyen d'erreurs de trames, nommé FER, pour CHOIR, avec et sans notre algorithme proposé, et en fonction du nombre de dispositifs terminaux dans le réseau. SF7 est présenté à gauche, et SF12 est présenté à droite. Dans CHOIR, les trames indiscernables en collision sont celles qui ont

les mêmes petits décalages de fréquence, avec une granularité d'un dixième de symbole. Le gain de notre algorithme proposé provient de la récupération de ces trames. Les résultats montrent que notre algorithme proposé améliore les capacités de décodage pour le trafic non confirmé de 13% avec SF7 et de 11% avec SF12.

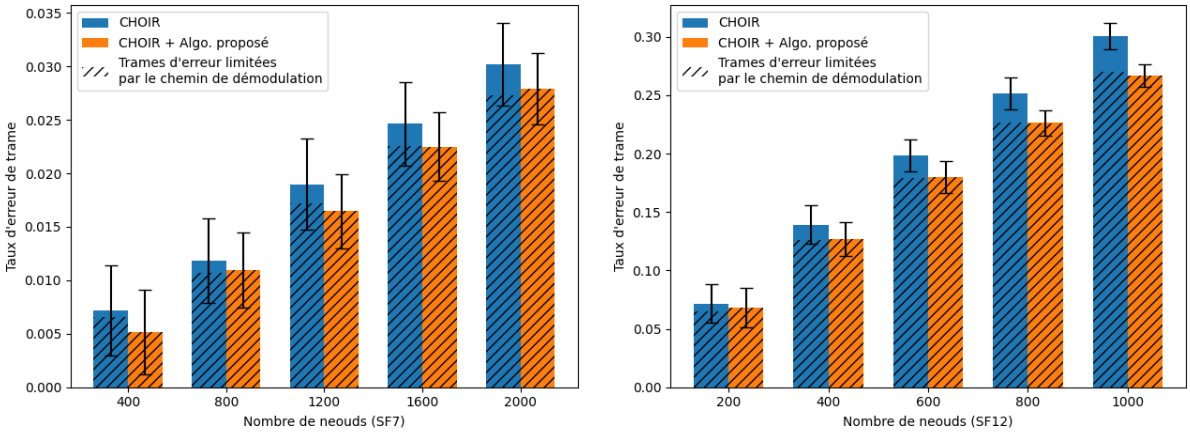


Figure A.13: Résultats de simulation de CHOIR avant et après l'application de notre algorithme de récupération des trames. SF7 est affiché à gauche et SF12 est affiché à droite.

La figure A.14 montre le FER pour SIC, avec et sans notre algorithme proposé, en fonction du nombre de dispositifs terminaux dans le réseau. SF7 est présenté à gauche, et SF12 est présenté à droite. Selon l'hypothèse de [5] et du modèle de propagation, les incertitudes proviennent des trames envoyées par les dispositifs terminaux à des distances similaires de la passerelle. Le gain de notre algorithme proposé provient de la récupération des trames avec les niveaux de puissance les plus forts et les deuxièmes plus forts. Les résultats montrent que notre algorithme proposé améliore les capacités de décodage pour le trafic non confirmé de 35% avec SF7 et de 47% avec SF12.

La figure A.15 montre le FER pour GS-MAC, avec et sans notre algorithme proposé, en fonction du nombre de dispositifs terminaux dans le réseau. SF7 est présenté à gauche, et SF12 est présenté à droite. La récupération des trames envoyées dans le même sous-slot et les incertitudes causées par les symboles répétés donnent un gain significatif à GS-MAC lorsque notre algorithme proposé est utilisé. Les résultats montrent que notre algorithme proposé améliore les capacités de décodage pour le trafic non confirmé de 28% avec SF7 et de 19% avec SF12.

Résultats concernant SF-DS Nous avons réalisé une comparaison de SF-DS avec un autre ensemble d'algorithmes, basée sur notre simulateur. Nous y avons aussi intégré les résultats de CIC [48]. Pour ce nouvel ensemble de simulations, nous générons et stockons des signaux superposés afin d'appliquer chaque algorithme aux mêmes signaux, pour des raisons de cohérence et de reproductibilité. Chaque algorithme de

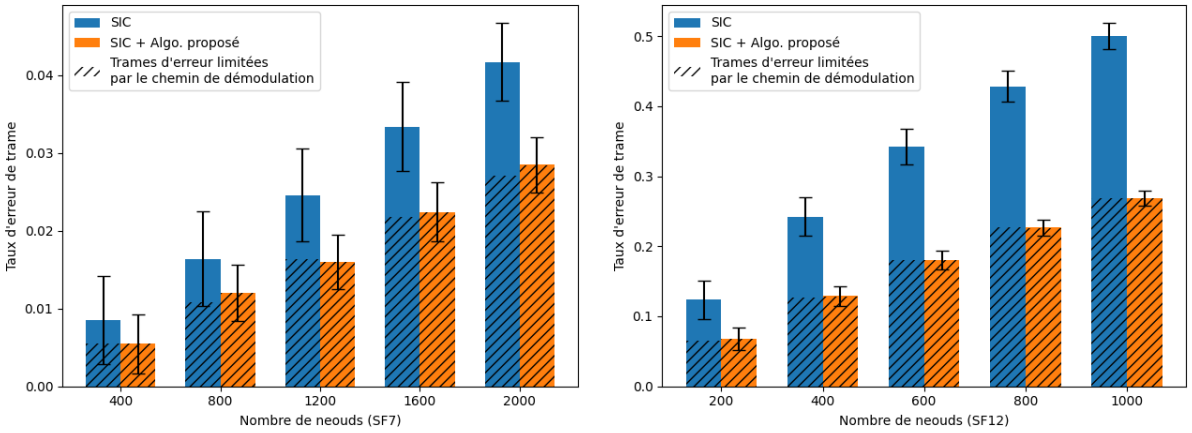


Figure A.14: Résultats de simulation de SIC avant et après l'application de notre algorithme de récupération des trames. SF7 est affiché à gauche et SF12 est affiché à droite.

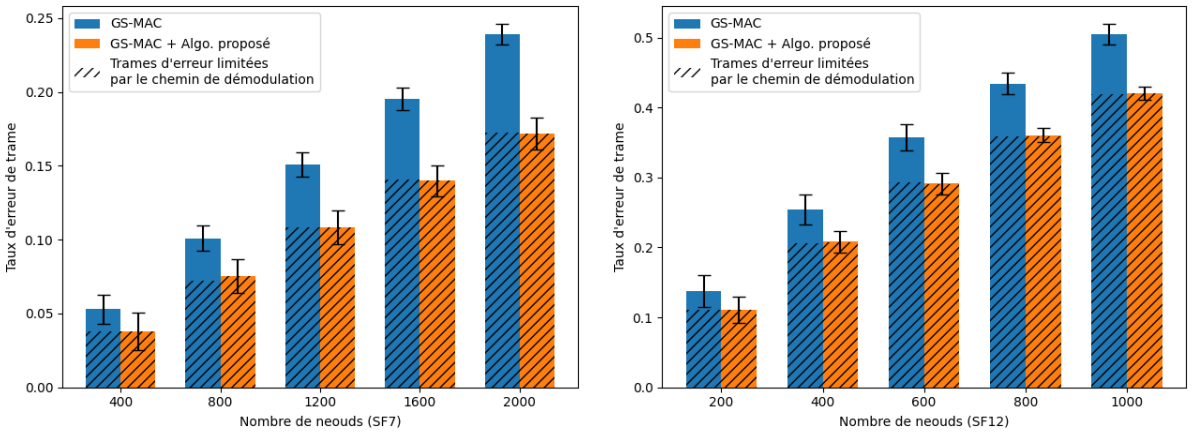


Figure A.15: Résultats de simulation de GS-MAC avant et après l'application de notre algorithme de récupération des trames. SF7 est affiché à gauche et SF12 est affiché à droite.

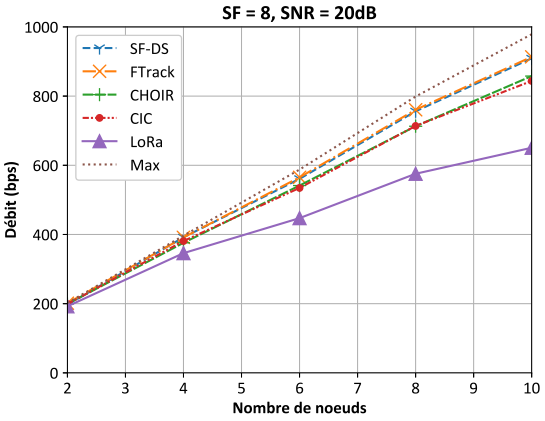
décodage opère sur les mêmes fichiers d'échantillons, sans détection de préambule. Nous avons modifié la réalisation du CIC, disponible en ligne, afin de supprimer sa détection de préambule et d'éviter l'impact des trames non détectées. Nous avons utilisé SF8 et SF12, et nous avons fait varier le SNR entre une valeur élevée (20 dB) et une valeur faible (-5 dB). Pendant chaque simulation, nous déployons de deux à dix dispositifs terminaux uniformément autour d'une passerelle, et les dispositifs terminaux transmettent périodiquement des trames avec des intervalles aléatoires entre 1000 et 2200 ms. Ces paquets sont constitués d'une charge utile aléatoire de 22 octets et, après encodage avec $CR = 1$, chaque trame contient 38 symboles de charge utile pour SF8 et 28 symboles de charge utile pour SF12.

La figure A.16 montre le débit réseau des différents algorithmes. Il est important de noter que le graphe étiqueté "Max" représente le débit idéal si toutes les trames transmises étaient décodées sans erreur. Lorsque SF8 est utilisé, le réseau n'a pas encore atteint sa capacité maximale avec dix dispositifs terminaux. Bien que certaines trames soient perdues en raison de collisions, l'ajout de plus de dispositifs terminaux augmente le débit réseau. Dans le scénario de fort SNR, tous les algorithmes de décodage de collision sont plus performants que LoRa. Cependant, dans le scénario de faible SNR, les performances de CHOIR sont moins bonnes que celles de LoRa car les petites déviations de fréquence sont masquées par le bruit et ne peuvent pas être détectées par la passerelle. Quand le SNR est faible, SF-DS connaît une légère diminution de ses performances car le bruit peut entraîner des échantillons incomplets, décalant ainsi les bandes de FFT et réduisant la précision de la détection des pics de FFT. Il convient de noter que tous les algorithmes de résolution de collision utilisant des FFT plus courts que la longueur du symbole souffrent de fuite spectrale, ce qui affecte la précision de la détection des pics. Lors de l'utilisation de SF12, LoRa maintient un débit relativement constant à mesure que le nombre de dispositifs terminaux augmente en raison de sa capacité à capturer une seule trame à la fois. Cependant, les autres algorithmes de résolution de collision se comportent de manière similaire à SF8 et n'affichent pas de diminution significative du débit. La longueur de symbole plus longue de SF12 (plus précisément, 16 fois plus longue qu'avec SF8) augmente les niveaux de puissance sur les valeurs obtenues par la FFT, ce qui aide à distinguer les pics de LoRa parmi le bruit. Il est intéressant de noter que SF-DS atteint les meilleures performances pour SF12 dans les scénarios de fort et de faible SNR.

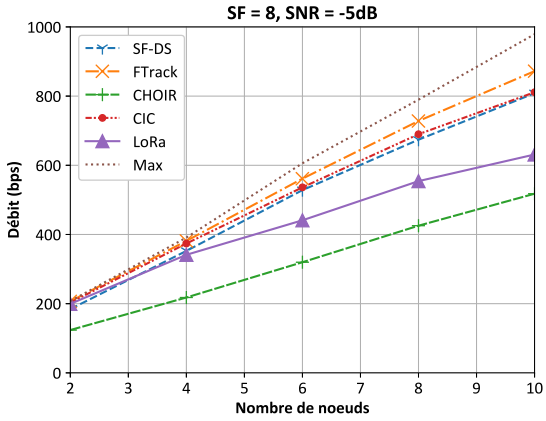
A.7 Conclusions et perspectives

Dans cette thèse, nous avons proposé un nouveau schéma de décodage pour les trames LoRa en collision, exploitant les fonctionnalités des trames LoRa, ainsi qu'un algorithme basé sur le codage LoRa et conçu pour améliorer les performances de tout algorithme de résolution de collision.

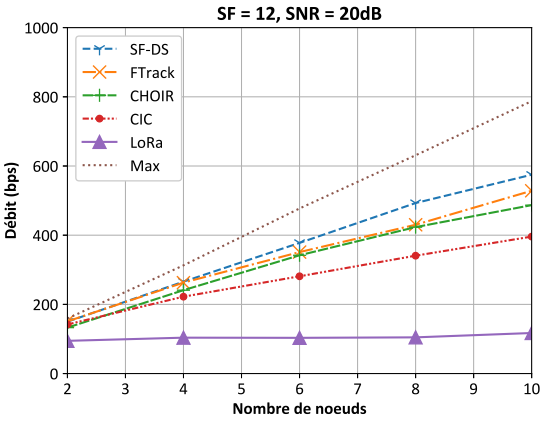
Pour aller plus loin, les paramètres du schéma de décodage proposé peut être configuré dynamiquement pour adapter les collisions différentes, ainsi que le choix d'algorithme de décodage. Par rapport à l'algorithme de décodage présenté dans la section A.4, la passerelle peut accélérer les réalisations par des matériaux. La



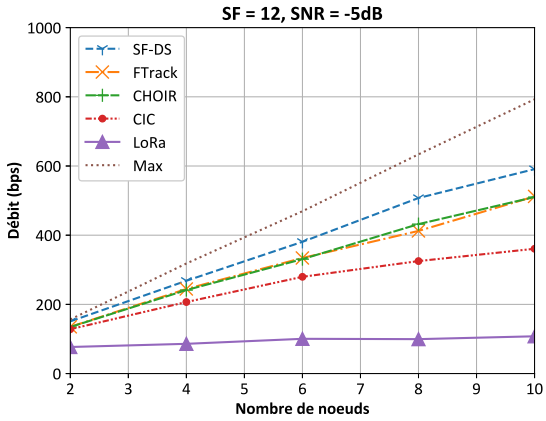
(a) SF8, $SNR = 20\ dB$



(b) SF8, $SNR = -5\ dB$



(c) SF12, $SNR = 20\ dB$



(d) SF12, $SNR = -5\ dB$

Figure A.16: Débit des algorithmes de décodage selon différents SNR. L'intervalle d'envoi sature le réseau avec un SF de 12. Le SNR influence également les performances relatives des différents algorithmes.

combinaison entre la couche physique dans le simulateur GNU Radio et le simulateur NS-3 est aussi une approche promise, qui peut également profiter toutes ces couches pour améliorer les performances d'ADR dans la couche MAC. L'exploration et l'évaluation de l'orthogonalité imparfaite dans LoRa deviennent aussi une approche à étudier, dans le cas de collision de trame LoRa.