From Demodulation to Decoding: Toward Complete LoRa PHY Understanding and Implementation

ZHENQIANG XU, SHUAI TONG, PENGJIN XIE, and JILIANG WANG, School of Software, Tsinghua University, China

LoRa, as a representative of Low Power Wide Area Network technology, has attracted significant attention from both academia and industry. However, the current understanding of LoRa is far from complete, and implementations have a large performance gap in SNR and packet reception rate. This article presents a comprehensive understanding of LoRa physical layer protocol (PHY) and reveals the fundamental reasons for the performance gap. We present the first full-stack LoRa PHY implementation with a provable performance guarantee. We enhance the demodulation to work under extremely low SNR (−20 dB) and analytically validate the performance, where many existing works require SNR > 0. We derive the order and parameters of decoding operations, including dewhitening, error correction, deinterleaving, and so on, by leveraging LoRa features and packet manipulation. We implement a complete real-time LoRa on the GNU Radio platform and conduct extensive experiments. Our method can achieve (1) a 100% decoding success rate while existing methods can support at most 66.7%, (2) −142 dBm sensitivity, which is the limiting sensitivity of the commodity LoRa, and (3) a 3,600-m communication range in the urban area, even better than commodity LoRa under the same setting.

CCS Concepts: • **Networks** → **Network protocols**; **Network performance analysis**; **Wide area networks;**

Additional Key Words and Phrases: Low-power wide-area network, LoRa, physical layer

ACM Reference format:
Zhenqiang Xu, Shuai Tong, Pengjin Xie, and Jiliang Wang. 2022. From Demodulation to Decoding: Toward Complete LoRa PHY Understanding and Implementation. *ACM Trans. Sensor Netw.* 18, 4, Article 64 (December 2022), 27 pages.
https://doi.org/10.1145/3546869

1 INTRODUCTION

Low Power Wide Area Network (LPWAN) technology has been shown to be very promising for connecting millions of devices in the **Internet of Things (IoT)** by providing long-distance and low-power communication under a very low SNR. LoRa, as a representative LPWAN technology, has recently been widely adopted in both academia and industry [1–4]. It has been used in various IoT applications, e.g., smart city, smart agriculture, and smart logistics [5–8]. It also attracts a large

This work is in part supported by NSFC No. 62172250, No. 61932013, Tsinghua University Initiative Scientific Research Program.
Authors' address: Z. Xu, S. Tong, P. Xie, and J. Wang (corresponding author), School of Software, Tsinghua University, Beijing, China; emails: xu-zq17@mails.tsinghua.edu.cn, tl19@mails.tsinghua.edu.cn, xiepengjin@tsinghua.edu.cn, jiliangwang@tsinghua.edu.cn.

1550-4859/2022/12-ART64
https://doi.org/10.1145/3546869

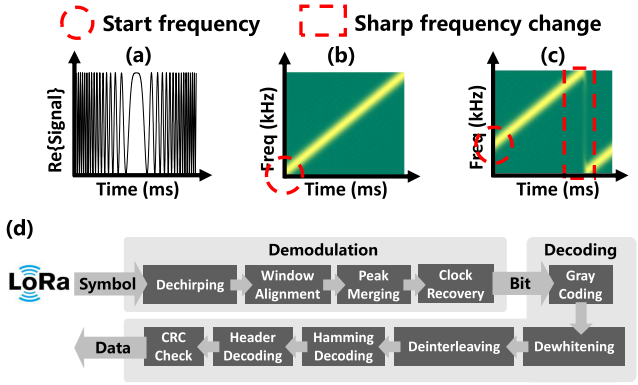

Fig. 1. (a) Real part of a base up-chirp symbol. (b) Base up-chirp symbol. (c) Shifted symbol. (d) Complete procedure of LoRa PHY.

number of research works including resolving collisions [9, 10], LoRa backscatter [11–13], weak signal decoding [14, 15], and so on.

LoRa adopts the **Chirp Spread Spectrum (CSS)** modulation mechanism, which provides anti-interference and long-range communication capability. As shown in Figure 1(a) and (b), a base symbol in LoRa **physical layer protocol (PHY)** is a chirp with frequency linearly increasing with time. The start frequency (f_{start}) of a symbol represents the encoded information. A LoRa symbol has two segments with a sharp frequency drop, as shown in Figure 1(c). Though LoRa has attracted much attention in academia and industry, the details of LoRa PHY, i.e., how LoRa demodulates and decodes the received signal, is still unknown to us yet, because LoRa PHY is a closed protocol owned by Semtech Corporation. In this work, we aim to present a complete understanding of the LoRa physical layer and provide a better tool for analyzing the LoRa network. We reveal that most existing understandings are incomplete and even incorrect. As a result, even the most widely used LoRa implementation ***rpp0/gr-lora*** **(RPP0)** [16] has a large performance gap to the real hardware, e.g., **Packet Reception Rate (PRR)** of RPP0 is only 20.0%, and RPP0 requires a much higher SNR than LoRa.

There are two main steps in understanding LoRa PHY: demodulation and decoding.[1] Demodulation translates symbols to raw bits, while decoding translates these raw bits to meaningful data bytes. The goal of LoRa demodulation is deriving f_{start} of each symbol. Demodulation directly determines the performance of the receiver. The current mainly adopted LoRa demodulation first translates the signal to a single tone by multiplying each symbol with a linearly decreasing chirp and extracts the frequency of the single tone as the start frequency. This operation can lead to a high SNR loss due to the signal loss in the translation and failure to leveraging LoRa features, which prohibits LoRa from long-range low-SNR communication. We explain the exact reasons for signal loss in Section 3.1.

For decoding, existing works [16–21] cannot derive the order and parameters for decoding operations. Thus, they have a very low PRR (e.g., under 66.7%) even for high SNR due to the incorrect understanding of the LoRa decoding process. Meanwhile, they are often incomplete and over-claimed, e.g., methods in References [16–18] claim they can support all **spreading factors (SF)**, but our experiments show that they fail to do so.

[1]For the transmitter, modulation, and encoding. Here we only discuss the behavior of the receiver for simplicity.

Table 1. Comparison of BastilleResearch/gr-lora (BR), RPP0 and tapparelj/gr-lora_sdr (TAPP) and our implementation

	BR [17]	RPP0 [16]	TAPP [18]	Ours
All SFs	✗	✗	✗	✓
All CRs	✓	✓	✓	✓
CRC	✗	✗	✓	✓
Explicit Mode	✗	✓	✓	✓
Implicit Mode	✓	✗	✓	✓
Clock Recovery	✗	✗	✗	✓
Low SNR Support	✗	✗	✗	✓
Real Time	✓	✓	✗	✓
Packet Coverage	**4.2%**	**20.0%**	**66.7%**	**100%**

This article provides a full understanding and a complete full-stack real-time LoRa PHY with a provable performance guarantee. In the demodulation part, we reveal the fundamental reason for SNR loss is phase misalignment due to window misalignment and internal symbol phase offset. We first compensate the window misalignment by accurate window offset measurement. To address internal symbol phase offset, unlike existing works with sampling frequency B (Bandwidth), we oversample the signal with $2B$ to calculate peaks for each segment separately. Then we propose a method to merge those two peaks with minimal sensitivity loss. We theoretically prove that the sensitivity of our method is very close to the "perfect" demodulation. Moreover, we propose a dynamic clock drift compensation algorithm for low-cost LoRa devices and relatively long LoRa packets.

In the decoding part, we show how LoRa PHY translates the bits from the demodulation part to meaningful packets. According to the observations from the official formula [22] for calculating the number of symbols in a LoRa packet, we infer and verify the packet structure from the black-box LoRa by manipulating the packet contents. Based on the packet structure, we analyze the requirements of each decoding process and derive the order of decoding processes for Gray coding, deinterleaving, Hamming decoding, and dewhitening. Then, leveraging the inferred decoding order and packet structure, we manipulate the transmitting packets to derive the configuration parameters in the decoding process, the header structure, and the CRC polynomial.

We implement a real-time **Software Defined Radio (SDR)** LoRa PHY, as shown in Figure 1(d). Table 1 shows the comparison with states of the art. Our implementation outperforms those approaches by (1) providing real-time decoding with very low SNR requirement, (2) supporting all modes and parameters of LoRa, and (3) working robustly under low-cost devices with clock drift and long packets. We theoretically show that our implementation could work under SNR = -20 dB. The order of decoding operations is also provable, which can be verified by the generated random sequence (Section 4.3). In our experiments, 100% of the packets could be correctly decoded. Therefore we consider the configuration as reliable.

Contributions.

- We show the performance gap between existing LoRa implementations and the commodity LoRa and reveal the fundamental reasons. We present a comprehensive understanding of the demodulation and decoding of LoRa.
- We implement a complete full-stack real-time LoRa PHY on the GNU Radio SDR platform.[2] Based on our analysis, we can even analyze and improve the performance of commodity

[2] https://github.com/jkadbear/LoRaPHY.

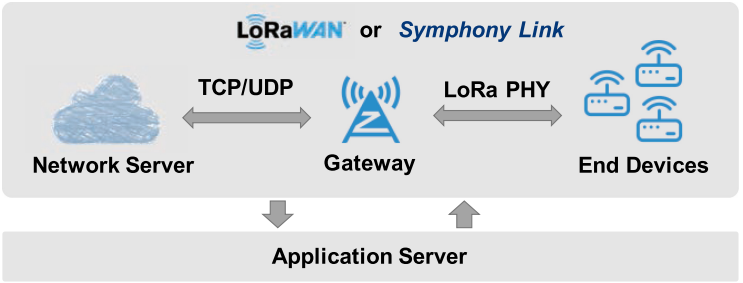

Fig. 2. LoRa network architecture. LoRaWAN or Symphony Link protocol controls how a network server interacts with end devices.

LoRa. For example, we show that the commodity LoRa chip is vulnerable to "*phase misalignment attack*" (Section 6.5), and we show how to address this vulnerability.

- We conduct extensive experiments to verify the performance. The results show that the sensitivity of our implementation reaches −142 dBm, which is the limiting sensitivity of commodity LoRa. Our implementation has an effective communication range of 3,600 m in the urban environment, while the commodity RPI gateways (Section 5) only reach the maximum of 2,800 m. Our LoRa implementation has a much higher decoding success rate (100%) than existing works (≤66.7%).

2 BACKGROUND

2.1 Known and Unknown Facts of LoRa

Figure 2 shows the mainstream LoRa network architecture. LoRaWAN [23] or Symphony Link [24] protocol controls the whole LoRa network and provides a data interface to the application server. There are three main communication entities in the LoRa network, i.e., network server, gateway, and end device. The network server communicates with the gateway using a traditional internet connection while the end devices exchange data with the gateway by using LoRa PHY wireless protocol. Nearly all components and protocols used in Figure 2 have complete open source implementations except LoRa PHY. Though there exists some reverse-engineering works, the understanding of LoRa PHY is not yet complete. In the following, we demonstrate some basic concepts of LoRa PHY and discuss the details about demodulation and decoding in Sections 3 and 4.

LoRa PHY employs CSS to reach long-range transmission. The basic communication unit in LoRa PHY is a chirp symbol. As shown in Figure 1(b), we denote this unshifted chirp symbol as *base chirp*. When the frequency increases with time, we name it up-chirp and otherwise down-chirp. The novelty of LoRa modulation is using the start frequency of a cyclically shifted symbol to encode data. Figure 1(c) shows a LoRa symbol with two chirp segments. In LoRa specifications [22, 25, 26], the number of bits encoded by a symbol is SF. The SF satisfies $2^{SF} = B \cdot T$, where B is bandwidth and T is chirp period. There are at most 2^{SF} cyclically shifted up-chirp symbols given a specific SF. An up-chirp can be represented as

$$chirp(t; f_0) = Ae^{j2\pi(f_0 + \frac{B}{2T}t)t}, \tag{1}$$

where A is amplitude and f_0 is start frequency ($t = 0$).

Figure 3 shows the basic understanding of a standard LoRa packet. Overall, it has three parts: preamble, **start frame delimiter (SFD)**, and data symbols. The preamble is a series of base

Fig. 3. A LoRa packet in the time-frequency plane.

up-chirps followed by two up-chirp symbols indicating network ID.[3] The SFD is 2.25 down-chirps indicating the start of data symbols. Data symbols include PHY header, payload, and payload CRC (header and CRC are optional).

LoRa supports the following configurations. **Code Rate (CR)**: LoRa applies Hamming code of coding rate $\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$, or $\frac{4}{8}$ [22]. **Low Data Rate Optimization (LDRO)**: When sending long packets, LoRa enables LDRO mode for better stability at the data rate cost. Implicit/explicit mode: In explicit mode, there is a PHY header in the packet, while in implicit mode, there is no PHY header.

Unknown:[4] (1) The ideal demodulation of LoRa, i.e., how a symbol is translated to bits, is still unknown, which is the key for LoRa to achieve low SNR and long-distance communication. (2) None of the existing works proposes a full understanding of the LoRa packet structure. (3) The details of LoRa specific modes are unknown, e.g., LDRO mode.

2.2 State-of-the-Art Open Source Implementations

The space of prior research in LoRa communication is quite rich, including LoRa collision decoding [1, 2, 4, 9, 10, 27], LoRa chirp based backscatter [11–13, 28], LoRa network optimization [15, 29], and so on. Unfortunately, most of them do not release public accessible implementations. Considering that the LoRa network architecture is brand new and not maturely understood by the community, any new open source efforts deserve appreciation. Therefore, we first summarize the public available LoRa related open source projects.

Upper Layer Implementations. The upper layer of the LoRa network has rich open source implementations. For LoRa gateway and end device, Semtech corporation provides official embedded implementations [30, 31]. For network servers, ChirpStack [32] is a widely used implementation compatible with LoRaWAN. LoRaWAN-Server [33] provides a compact combination of network server and application server. The lowest level code here is the driver for LoRa chips. Therefore, it does not include the physical layer mechanism of LoRa communication.

Physical Layer Implementations. Since LoRa PHY is a proprietary protocol belonging to Semtech corporation, there is no public document explaining all the details of LoRa PHY. Existing researches and systems mainly rely on three open source SDR implementations, i.e., BR [17], RPP0 [16] and TAPP [18]. BR focuses on the implicit header mode of the LoRa packet. BR applies dechirping based demodulation. It ignores some crucial details about dechirping and is not reliable in low SNR. We will discuss the details in Section 3. Besides, the implementation of BR only supports decoding LoRa packets with SF=8. Our tests show that BR can only decode the first four bytes of a packet and fails to decode any complete packet with more than four bytes. RPP0 targets explicit header mode LoRa packet decoding. As shown in Figure 1(c), RPP0 demodulates LoRa symbols by finding sharp frequency changes and then uses them to estimate the start frequencies of the

[3]The two symbols are not base up-chirps typically. They are used like network masks to separate different networks. For example, they are set to 0x0304 for public network in LoRaWAN.

[4]Though existing works have revealed many details about LoRa PHY, the non-ideal demodulation performance and non-100% decoding success rate mean their results are incomplete.

symbols. RPP0 relies on time-domain information, and it cannot decode LoRa packets for SNR<0. TAPP is a recent implementation of LoRa. TAPP is not optimized for LoRa demodulation, which results in high computation overhead. PRR of TAPP decreases when the decoder cannot consume the input data in time. According to our evaluation, its packet loss reaches 70% for a data rate of one packet per second. The average PRR of TAPP is only 66.7%. For decoding, none of those existing works can provide full decoding capability, and thus they can only decode a portion of LoRa packets. Besides, there are other LoRa PHY related works that provide open source implementations. Charm [14] focuses on weak signal decoding by leveraging the multi-gateway structure of the LoRa network. However, the Charm decoder [34] is only implemented in offline mode with MATLAB. Moreover, the author ignores the decoding procedure and the phase alignment problem (see Section 3). TinySDR [35] is an SDR platform tailored for internet-of-things networks. The author implements a LoRa modulator [36] in FPGA, which shows us a convenient hardware chirp generation method. But the demodulation process is commonly much more complicated than modulation, and there is no open source FPGA-based LoRa demodulator currently. In addition, the decoding process is also not considered in TinySDR. We see that existing works have not provided a fully usable implementation and complete understanding of LoRa PHY.

Fundamental limitations of existing works. (1) Existing works cannot work for low SNR, and thus they do not have the crucial advantage of LoRa. (2) Existing works cannot provide full decoding capability for LoRa, and their performance is significantly lower than the commodity LoRa hardware. They cannot provide full support of LoRa, i.e., LDRO mode, all SFs, and so on.

Our goal. We aim to provide a complete LoRa PHY implementation, including demodulation and decoding, to enable extremely low SNR signals receiving and all parameters/modes of LoRa.

3 DEMODULATION

In this section, we reveal the demodulation process of LoRa and present how to break the fundamental limitations of existing LoRa implementations.

3.1 Phase-aligned Dechirping

The LoRa demodulator converts chirp symbols to bitstreams by first dechirping the received signal. The dechirping process mainly consists of two steps: First, it multiplies each received chirp with a base down-chirp, and then it performs the Fourier transformation on the multiplication results, translating chirps to energy peaks in the frequency domain. Ideally, the multiplication will result in single tone signals with continuous phase, whose energy can be accumulated to a single peak, as shown in Figure 4(a3) (denoted as IDEAL). However, in practice, there is an inevitable phase misalignment when the chirp frequency drops from its maximum to minimum. This phase misalignment is mainly induced by the hardware instabilities of inexpensive LoRa devices and is randomly distributed between 0 and 2π. With the phase misalignment, energy peaks from the Fourier transformation are severely distorted, as shown in Figure 4(a4), which limits the LoRa demodulation performance, especially under low SNR.

Existing works fail to resolve the phase misalignment problem in dechirping. Thus, their performance deteriorates dramatically in low SNR scenarios, far from the ideal sensitivity of LoRa demodulation. Our goal is to approach the ideal sensitivity of LoRa demodulation under phase misalignment. The design of our LoRa demodulator mainly consists of the following components.

Window Alignment. We need to accurately align the demodulation window (sample points level) with each symbol to retain the energy of the entire symbol for the demodulation process. Traditional works use preamble correlation for window alignment. However, merely using preamble correlation cannot achieve accurate alignment due to **Carrier Frequency Offset (CFO)**. We leverage the SFD part in the LoRa packet for window alignment while eliminating the impact of

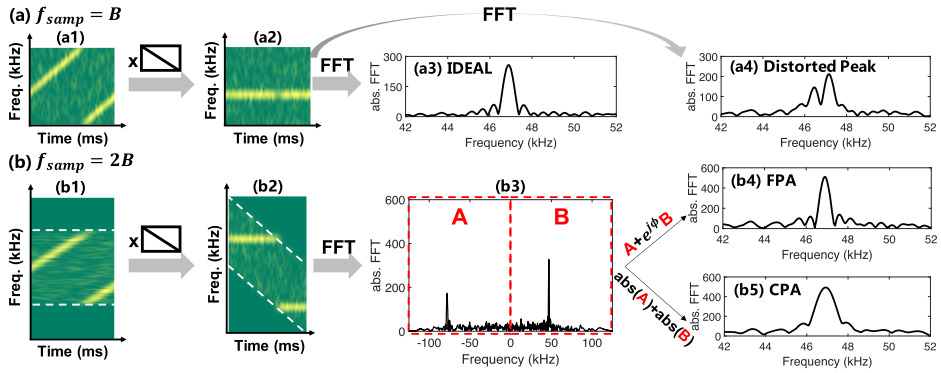

Fig. 4. The main step of dechirping is multiplying a down-chirp and then doing FFT. (a) Normal dechirping with SNR loss. For an ideal situation, we derive a peak like (a3). But for a non-ideal situation, the frequency peak would be distorted like (a4), which incurs wrong demodulation results. (b) Our proposed phase-aligned dechirping. We use the low-pass filter and oversampling to separate the two chirp segments in a LoRa symbol and then combine them. (b4) and (b5) give the fine and coarse phase-aligned dechirping results, respectively, which are stable under any situation.

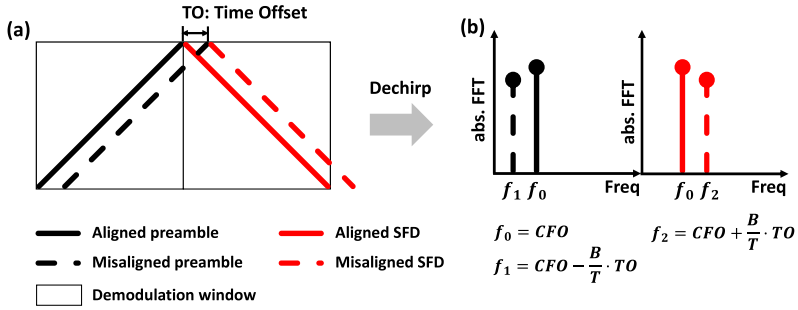

Fig. 5. Window alignment using base up-chirps and down-chirps.

CFO. The key operation is to combine the down-chirps in SFD together with up-chirps in the preamble. We apply dechirping to both of them (the down-chirp is multiplied with a base up-chirp). If the demodulation window is perfectly aligned with the signal, then the peak of the up-chirp and peak of the down-chirp appear at the same frequency, regardless of the CFO, as shown in Figure 5. Otherwise, there is a significant difference in the frequency of the two peaks, depending on the time offset between the chirp and the demodulation window. Therefore, we can achieve accurate alignment based on the frequency of peaks corresponding to the preamble and SFDs.

Oversampling and Peak Merging. We have previously revealed that the fundamental reason for SNR loss in demodulation is the peak distortion due to the phase misalignment when the chirp frequency drops from its maximum to minimum. To address this problem, we propose an oversampling-based approach that recovers the peak distortion in the existence of phase misalignment and approaches the ideal sensitivity of LoRa demodulation.

As illustrated in Section 2, LoRa modulates signals by cyclically shifting the frequency of a base up-chirp. Each modulated chirp consists of two chirp segments, one with the initial frequency of f_0 and the other with $f_0 - B$. Existing works demodulate the received chirp symbols using a sampling frequency equaling to the chirp bandwidth B. Thus, after multiplying with the base

down-chirp, both chirp segments are translated to single tone signals of the same frequency due to frequency aliasing. However, there exists an inevitable phase misalignment between these two chirp segments. When performing the Fourier transformation on the whole symbol, the energy of these two chirp segments adds up destructively, leading to peak distortion and SNR loss for LoRa demodulation. Existing works have no idea of the value of phase misalignment. Thus, they cannot distinguish the signal of these two chirp segments and fail to correct the peak distortion.

We propose an oversampling-based strategy for efficiently separating the two chirp segments in the frequency domain. Specifically, as the initial frequencies of the two chirp segments are f_0 and $f_0 - B$, when oversampling the signal with a frequency higher than $2B$ (i.e., $F_s \geq 2B$), the dechirping results in two distinct peaks separately locate at f_0 and $F_s - B + f_0$, as shown in Figure 4(b3). Since there is no phase misalignment within each chirp segment, both of these two peaks are distortion-free. This gives us a chance to perfectly concentrate the energy of the whole chirp by coherently adding up these two peaks of chirp segments and thus eliminating the influence of phase misalignment. For the rest of this section, we illustrate how to merge these two energy peaks in the frequency domain efficiently with small sensitivity loss for LoRa demodulation.

Due to the existence of phase misalignment, the phase of the two peaks in Figure 4(b3) are out of coherence. Thus, directly adding up those two complex peaks will not increase the peak height or even cancel each other due to the phase difference of these two peaks. We propose **Fine-grained Phase Alignment (FPA)** to search for all possible values of phase misalignment $\Delta\varphi = \frac{i \times 2\pi}{k} (i = 0, 1, \ldots, k-1)$ and compensate it before adding up two peaks. We iterate k possible phase offset and select the $\Delta\varphi$ that generates the highest peak. The result of FPA is shown in Figure 4(b4). Compared with Figure 4(a3), FPA can approach the performance of IDEAL as the phase difference is compensated.

The searching process for determining the value of the phase misalignment has high computational complexity. To reduce this overhead, we further propose **Coarse-grained Phase Alignment (CPA)**, which requires much less computation overhead and thus can work on low-cost hardware with very limited computation resources. The key observation is that the peak amplitude is in proportion to the energy of the corresponding chirp segment. Given the energy of the whole symbol is exactly the sum of the two chirp segments, if we directly add up the absolute value (amplitude) of part A and part B, then the resulted peak will have the same height as that of the IDEAL peak. The result of CPA is shown in Figure 4(b5). We see that CPA significantly changes the shape of the peak, i.e., the width of the main lobe increases. Besides, CPA also lifts the noise level during the adding up process, thus slightly degrading the demodulation performance compared with FPA.

Depending on the computing power of the receiver, we can switch between FPA and CPA methods. In the following, we theoretically analyze the **symbol error rate (SER)** with respect to different SNRs for IDEAL, FPA, and CPA.

We assume that the LoRa signal is transmitted over an **additive white Gaussian noise (AWGN)** channel, where the noise is modeled as following the complex Gaussian distribution. During the dechirping process, both the chirp symbols and the noise are transformed into energy peaks in the frequency domain. A symbol error happens when any of the noise peaks exceeds the peak of the target symbol. Suppose the peak height of the target symbol is h_d, and the maximum peak corresponding to noise is h_n; we can derive the expected symbol error rate as

$$SER = P(h_d < h_n). \tag{2}$$

SER is a function of SNR theoretically. The calculation details of h_d and h_n for IDEAL, FPA, and CPA are listed in Section 8. We plot the theoretical performance of IDEAL, FPA, and CPA under SF8/12 in

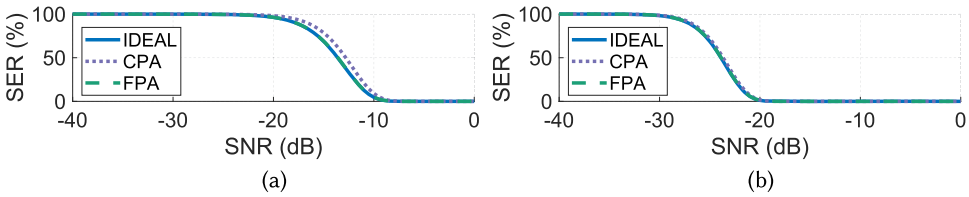

Fig. 6. Theoretical SER-SNR curve for (a) SF8 and (b) SF12.

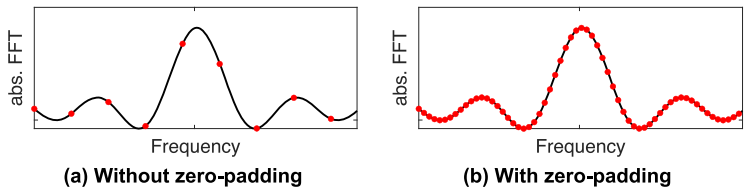

Fig. 7. Zero-padding effects. Red dots represent the DFT results after dechirping. Black lines represent the DTFT result after dechirping.

Figure 6.[5] The searching step length in FPA (i.e., $1/k$) is set to 1/16 empirically (a tradeoff between performance and computation overhead). We see that both FPA and CPA have performance very close to IDEAL for SF12, while the theoretical SER of CPA in SF8 is slightly larger than other two methods. In summary, the theoretical analysis tells us that FPA and CPA are two practical and effective methods in LoRa demodulation, which is further validated in experiment results in Section 6. Our theoretical analysis also shows that LoRa could work under extremely low SNR (i.e., SNR as low as −20 dB), as can be seen from Figure 6(b).

Peak Refinement. Previously, we have illustrated how to convert received chirps to frequency domain peaks. Next, we need to estimate the highest peak frequency to recover the modulated data bits. However, in practice, height estimation for peaks is closely related to the frequency resolution after the Fourier transformation. The black line in Figure 7(a) shows the result of ideal **Discrete-Time Fourier Transform (DTFT)** on the target signal, which is a continuous function of frequency. In practice, the receiver only performs **Discrete Fourier Transform (DFT)**, a sampled version of DTFT in the frequency domain, on the target signal. The output of DFT is marked as red dots in Figure 7. The naive DFT has a low-frequency resolution. Thus, its outputs are sparsely distributed over the ideal DTFT curve, decreasing the derived peak height and also impacting the peak estimation. We apply zero-padding to the signal in the time domain to obtain accurate peak estimation, equivalent to interpolation in the frequency domain. As shown in Figure 7(b), with zero-padding, the frequency resolution of DFT improves significantly, benefiting our peak estimation. We show that fourfold zero-padding achieves a balance between computation overhead and demodulation sensitivity in Section 6.3. The benefit of more zero-padding is marginal.

3.2 Clock Recovery

Since LoRa is commonly applied on low-cost devices, the equipped oscillators may not have high accuracy, which results in inaccurate bin values. Table 2 shows the first eight dechirped peak bins of two packets from a LoRa device having about 16-kHz CFO. We observe that the fractional part

[5]Bit-level parameters such as CR and CRC are irrelevant to SER in symbol level. It should be noted that bandwidth does not appear in the above SER formula, because SNR is a function of bandwidth (if we increase the bandwidth without raising the transmitting power, then the in-band SNR will drop).

Table 2. Peak Bin Drift under SF10 and SF12 (with Bandwidth 125 kHz)

		First 8 Peak Bins							
SF10 (LDRO off)	Expected	677.0	333.0	861.0	93.0	853.0	149.0	789.0	597.0
	Received	677.**2**	333.**2**	861.**3**	93.**3**	853.**4**	149.**4**	789.**4**	597.**5**
SF12 (LDRO on)	Expected	1757.0 = 011011 011101	3453.0 = 110101 111101	673.0 = 001010 100001	3757.0 = 111010 101101	2409.0 = 100101 101001	809.0 = 001100 101001	2981.0 = 101110 100101	2545.0 = 100111 110001
	Received	1757.8 = 011011 011101 +0.8	3453.9 = 110101 111101 +0.9	674.0 = 001010 100010 +0.0	3758.2 = 111010 101110 +0.2	2410.3 = 100101 101010 +0.3	810.4 = 001100 101010 +0.4	2982.5 = 101110 100110 +0.5	2546.6 = 100111 110010 +0.6

Expected: the ideal bins without clock error; received: the actual received bins.

of the peak bins keeps drifting with time. If we do not cancel that drift, then the fractional bins will accumulate and incur errors in the integer bin. Such bin drifts mainly come from **Sampling Frequency Offset (SFO)**. Due to the difference in sampling frequency, the actual LoRa symbol is τ shorter (or longer) than the standard symbol period T. Therefore, the start frequency of the next symbol has an additional drift as

$$\Delta f = \frac{B}{T} \cdot \tau. \tag{3}$$

Because the carrier frequency and sampling frequency are generated by the same oscillator, we have

$$\frac{\tau}{T} = \frac{SFO}{f_{samp}} = \frac{\Delta f_{osc}}{f_{osc}} = \frac{CFO}{f_{RF}}, \tag{4}$$

where f_{RF} is reference carrier frequency, f_{osc} is reference oscillator frequency, and Δf_{osc} is oscillator frequency bias. After translating to bin values, the estimated bin drift between two consecutive symbols is[6]

$$\Delta bin = \frac{\Delta f}{B/2^{SF}} = \frac{CFO}{f_{RF}} \cdot 2^{SF}. \tag{5}$$

For example, when CFO= 16 kHz, f_{RF} = 470 MHz, and SF = 10, the estimated bin drift $\Delta bin \approx 0.035$, which matches the tendency in Table 2. We subtract accumulated Δbin from the peak bins before decoding.

Another thing we need to consider here is the LDRO mode. Typically, when a chirp period is too long (i.e., $T > 16$ ms), LDRO is automatically enabled in LoRa chips [22]. In LDRO mode, as seen in the bottom of Table 2, the expected bin values always have the form of $4n+1$, which means the two **Least Significant Bits (LSBs)** do not encode data. Such a design aims to better protect the data, since lower bits are more vulnerable in long packet transmitting. For LoRa signals lasting a short time, the two LSBs are stable enough to encode data (e.g., SF10 and bandwidth 125 kHz). Nevertheless, in the experiment, we find that the first eight symbols are always in LDRO mode to protect the header information (Section 4).

[6]Without zero-padding, one integer bin in LoRa always represents a frequency of $\frac{B}{2^{SF}}$ regardless of the number of FFT points or sampling frequency.

4 DECODING

4.1 Overview

This section about decoding is *not a review* of previous works. We aim to provide a provable inference about the LoRa packet structure, decoding order, and configurations. From official LoRa datasheets [22, 25, 37] and patents [26], we know Equation (6) for calculating the number of symbols in a packet. In contrast, the packet structure is unknown. We also know that there are four main processes in decoding: Gray coding (G), deinterleaving (I), Hamming decoding (H), and dewhitening (W). In contrast, the order of the processes and the configuration parameters of each process are unknown. To show how LoRa PHY translates the bits from the demodulation module to meaningful packets, we analyze the LoRa decoding by three steps. First, we infer the packet structure from the black-box LoRa leveraging the number of symbols formula. Next, we reveal the order of the four main decoding processes. Finally, we infer the configuration parameters of each decoding process.

4.2 Packet Structure Inference

The official document [22] provides a formula to calculate the number of symbols in a packet:

$$n_{sym} = 8 + \max\left(\left\lceil \frac{2PL + 4CRC - 5IH - SF + 7}{SF - 2DE} \right\rceil \cdot \frac{4}{CR}, 0\right), \tag{6}$$

where PL is the number of payload bytes, SF is spreading factor, CRC is 1 if CRC check is enabled and otherwise 0, IH is 1 if implicit header mode (without header) is enabled and 0 if explicit mode (with header) is enabled, and DE is 1 if LDRO is enabled and otherwise 0.

We can derive the following *properties* from Equation (6):

- A LoRa packet has at least eight data symbols as $n_{sym} \geq 8$.
- $2PL + 4CRC$: The n_{sym} of a packet with PL payload bytes and CRC check enabled is equal to the n_{sym} of a packet with $PL + 2$ payload bytes and CRC check disabled. We can infer that the CRC check takes 2 bytes in the packet.
- $2PL + 4CRC - 5IH$: Similarly to CRC, we can infer the header takes 2.5 bytes.
- $SF - 2DE$: SF is the number of bits in a data symbol. Enabling LDRO causes the reduction of two bits per data symbol.
- $\frac{4}{CR}$: The number of data symbols increase with the unit of $\frac{4}{CR}$ (CR=$\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$, or $\frac{4}{8}$). For example, $CR = \frac{4}{7}$ means the packet applies (7,4) Hamming code, and n_{sym} has the form $\frac{4}{CR}n + 8 = 7n + 8(n = 0, 1, 2, \dots)$.

Based on the properties, we then manipulate the data packets to infer the packet structure. We first gradually increase packet size and observe a stairlike increase of the number of data symbols in real signals. If $CR = \frac{4}{7}$, then the number of symbols n_{sym} would increase following an arithmetic sequence (i.e., 8, 15, 22, ...). We also observe that when continually increasing the packet size, the newly added bytes are encoded in the last $\frac{4}{CR}$ symbols, and the first $\frac{4}{CR}(n-1) + 8$ symbols do not change. It reveals that data bytes are encoded by a $\frac{4}{CR}$ symbols block.

Further, we find that the first eight symbols are specially treated. Whether in explicit or implicit header mode, the first eight symbols' values always appear in the form of $4k + 1(k = 0, 1, 2, \dots)$, which means the last two bits of data are discarded, and there are $SF - 2$ bits data in each symbol. We can infer that the first eight symbols are in LDRO mode, i.e., $DE = 1$. LDRO gives better protection, and the header may be in the first eight symbols. We know that LoRa uses coding rate CR to protect the payload, and the payload bytes are encoded by a $\frac{4}{CR}$ symbols block. Similarly, the first eight symbols form an eight symbols block, and we guess that it uses a coding rate $\frac{4}{8}$ to give the header the highest protection (4 parity bits for a nibble). We also observe that the first eight

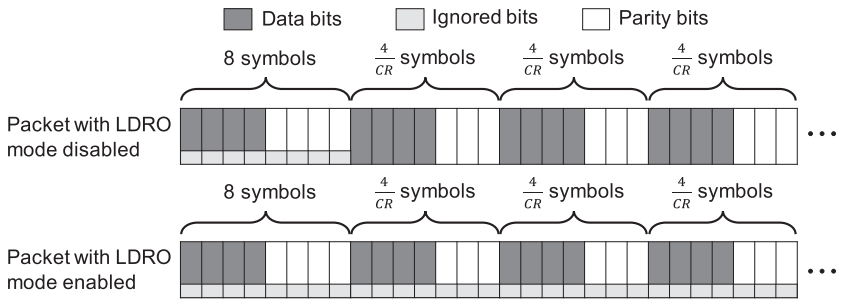

Fig. 8. The inferred LoRa packet structure for LDRO mode disabled/enabled. When LDRO mode is disabled, only the first eight symbols have ignored bits (protecting header information). When LDRO mode is enabled, the least significant two bits are ignored for all the symbols.

symbols encode payload bits in implicit header mode. Such a design can reduce the complexity of the decoding hardware complexity, because the explicit and implicit header mode can now be processed with the same procedure.

Figure 8 shows the inferred packet structure of a packet with LDRO mode disabled/enabled. In LDRO mode, the last two bits in each symbol are discarded. No matter whether the LDRO mode is enabled or disabled, the first eight symbols are in LDRO mode and use coding rate $\frac{4}{8}$. The following symbols are in blocks containing $\frac{4}{CR}$ symbols. The first four symbols encode data bits while the rest symbols encode parity bits.

Packet Structure Verification. We verify the above packet structure by comparing Equation (6) and the calculated symbol number using the inferred packet structure. For $SF \geq 7$, the first eight symbols contain $8 \cdot (SF-2) \cdot \frac{4}{8} = 4SF - 8 \geq 20$ data bits. Therefore, the first eight symbols could always include all header information. Except the header, the first eight symbols can contain $4(SF-2)-20(1-IH) = 4SF+20IH-28$ bits payload. Suppose there are PL bytes ($8PL$ bits) payload in total. If CRC is enabled, then additional 16 bits are needed. Except the first eight symbols, the total number of required data bits for the rest symbols is $8PL+16CRC-20IH-4SF+28$. Whitening and Gray decoding do not affect the total number of bits and the total number of symbols. The Interleaving operation only requires us to pad redundant bytes when the rest payload bytes cannot fill a block, thus not affecting the total number of symbol blocks. Each block has $\frac{4}{CR}$ symbols, and each symbol in the block could encode $SF-2DE$ bits. Due to the existence of parity bits, only $4(SF-2DE)$ bits in a block are actual data bits. Then the total number of symbols in the packet is

$$8 + \left\lceil \frac{8PL + 16CRC - 20IH - 4SF + 28}{4(SF - 2DE)} \right\rceil \cdot \frac{4}{CR}. \tag{7}$$

It is obvious that formula (7) is the same as Equation (6), which means the inferred packet structure is correct.

4.3 Order of Decoding Operations

In this section, we try to reveal the order of the four main processes in LoRa decoding: Gray coding (G), deinterleaving (I), Hamming decoding (H), and dewhitening (W). It is easy to know that Gray coding should be the first step, because it intends to solve the symbol adjacent drift problem (see Gray Coding in Section 4.4). Hamming decoding operates on bytes stream while a LoRa symbol contains $SF-2DE$ bits. So it relies on the transformed bytes stream after deinterleaving. Thus, the order of "G, I, H" should be "G→I→H." Hence, dewhitening has three possible positions, i.e., after

"G," "I," or "H." The implementations of BR, RPP0, and TAPP assume three different dewhitening positions, respectively.

We prove that the position of dewhitening does not affect the decoding results. Because the encoding process order is the reverse of decoding order, we use the operation in encoding to show the influence of the position of "W." Consider the data as a bit vector D, then interleaving is an operation that rearranges the position of bits. We can represent interleaving as a matrix I with each row or column containing only one "1." Hamming coding, as a linear coding method, can be represented as a matrix H. We have $H \cdot D = [P \cdot D\ D]$, where P is the parity matrix. Whitening is $W \oplus D$, where W is a random bit vector and $\oplus$ is exclusive-or (XOR). Following are the three possible orders of decoding and the corresponding encoding operations:

$$\text{"W}\rightarrow\text{I}\rightarrow\text{H"}: \quad W_1 \oplus (I \cdot [P \cdot D\ D]), \tag{8a}$$

$$\text{"I}\rightarrow\text{W}\rightarrow\text{H"}: \quad I \cdot (W_2 \oplus [P \cdot D\ D]), \tag{8b}$$

$$\text{"I}\rightarrow\text{H}\rightarrow\text{W"}: \quad I \cdot [P \cdot (W_3 \oplus D)\ (W_3 \oplus D)], \tag{8c}$$

where W_1, W_2, and W_3 are three different random bit vectors. Formula (8b) can be rewritten as

$$(I \cdot W_2) \oplus (I \cdot [P \cdot D\ D]). \tag{9}$$

Therefore, formula (8a) and formula (9) are equivalent (let $W_1 = I \cdot W_2$). Formula (8c) can be rewritten as

$$I \cdot ([P \cdot W_3\ W_3] \oplus [P \cdot D\ D]). \tag{10}$$

Formula (10) is a special case of formula (8b) (let $W_2 = [P \cdot W_3\ W_3]$). Therefore, both "I→W→H" and "I→H→W" can be represented by "W→I→H," which means the position of "W" does not affect the final decoding results. We will show the correct position for "W" in Section 4.4.

For the convenience of analysis, *we first assume the decoding order is* "G→W→I→H." Because the even parity of all-zeros is zero and interleaving in LoRa is just a diagonal realignment of all bits, the output codewords after "H" and "I" are zeros when the input bits are all zeros. Here we assume that LoRa adopts commonly used even-parity check, and the final results show that this assumption is correct. "W" is the XOR of data values and a pseudo-random sequence. When the decoding order is "G→W→I→H," as $x \oplus 0 = x$, if we set all transmission bits to zeros, then the output values after "G" are the dewhitening sequence. Then, we could use the derived dewhitening sequence to recover the temporal results before "I" and "H" for further analysis.

4.4 Configuration of Each Operation

This section reveals the key configurations of the four operations in LoRa decoding: Gray coding, deinterleaving, Hamming decoding, and dewhitening. We also show how to reveal the header structure and CRC polynomial.

Gray Coding. Gray coding is widely used in many wireless communication systems, which is a mapping from a bit vector to a binary representation. The adjacent representations of Gray coding only have a one-bit difference. In wireless communication systems, it is more likely to happen that we may misidentify a symbol to its adjacent symbol rather than another random symbol. For example, in LoRa modulation, the adjacent bin drift is common as described in Section 3.2. With Gray coding, the bit error caused by adjacent misidentification is reduced to one bit per symbol, which has a high possibility to be corrected by the error correction mechanism. The standard Gray coding can be expressed as

$$v = v_0 \oplus (v_0 >> 1), \tag{11}$$

where v_0 represents the raw demodulated bin value, $>>$ is the bitwise right shift operation, and v is the Gray coding output. However, when we continue our analysis based on natural symbol

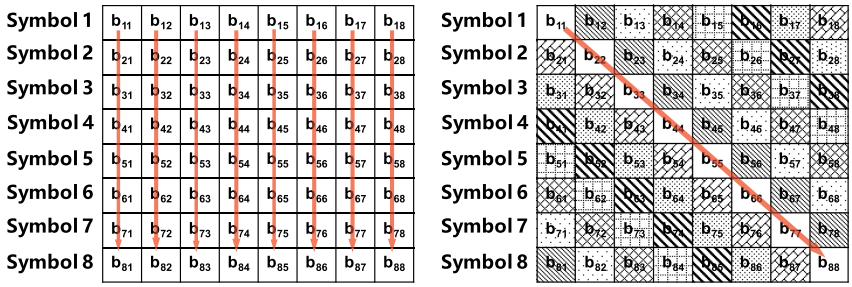

Fig. 9. An example deinterleaving block for 8 SF8 symbols ($CR = \frac{4}{8}$). Each row in the block is an 8-bit representation of a LoRa symbol after Gray coding. $b_{i,j}$ represents the jth bit of the ith symbol. For example, $b_{i,1}$ is the LSB of the symbol i. (a) Column-major order row-line interleaving, the nth codeword is composed of bits $b_{i,n}$ ($i = 1, \ldots, 8$). (b) Diagonal interleaving used in LoRa, the nth codeword is composed of bits $b_{i,(i+n-1)\%8+1}$ ($i = 1, \ldots, 8$).

mapping and standard Gray coding, we cannot decode packets with a 100% decoding success rate. Project RPP0 suffers from this problem and still contains an unfixed bug.[7] To understand the reason, let us recall the LDRO mode of LoRa. when LDRO is enabled, the encoder will put data bytes into high $SF-2$ bits of a symbol and then add "1" to reduce the influence of bin drift. From the hardware perspective, it is reasonable to add "1" under any condition, because we can save the circuit cost to judge whether LDRO is enabled. We guess that all symbols output from the encoder have one bin shift. If it is true, before we apply Gray coding, then we should subtract "1" from the demodulation results. Fortunately, our guessing is supported by the decoding results. The process of "G"in LoRa decoding could be modified as

$$v = (v_0 - 1) \oplus ((v_0 - 1) >> 1). \tag{12}$$

Note that TAPP also applies similar operations on Gray coding as we do, but they explain it as a different Gray coding mechanism and uses brute-force algorithm to derive the additional "one." However, it seems unnatural to use a non-standard Gray coding. By contrast, our explanation is more reasonable.

Deinterleaving. Interleaving is used to reduce the impacts of burst errors. With interleaving, the errors could be distributed to multiple bit groups and corrected by **forward error correction (FEC)**. It is mentioned [26] that LoRa applies diagonal interleaving instead of conventional row-line interleaving. Figure 9(a) shows the column-major order row-line interleaving of eight symbols. For row-line interleaving, the LSBs ($b_{i,1}$) of the eight symbols are assembled into a byte. From Section 3, we know that the LSBs of a symbol are more fragile than the **most significant bits (MSBs)** of the symbol. From the perspective of FEC, it is a bad design to group fragile bits together. Figure 9(b) shows diagonal interleaving used in LoRa. Diagonal interleaving distributes the fragile LSBs into different bytes and is more robust. We then manipulate the transmitted packets to derive the detailed diagonal mapping. As an example, we send packets with $SF = 8$ and $CR = \frac{4}{8}$ in implicit header mode. Thus, the interleaving block is an 8×8 block as shown in Figure 9(b). First, we assume the FEC used in $CR = \frac{4}{8}$ is the standard (7, 4) Hamming code with one bit extension. Therefore, after "G" and "W," the codeword for nibble "0000" is "00000000," and the codeword for "1111" is "11111111."[8] Suppose the sending bytes are all zeros except that the fourth byte is 0x0F, we observe

[7] https://github.com/rpp0/gr-lora/issues/99.

[8] Note that here we have removed the influence of whitening.

$b_{11} = b_{22} = \cdots = b_{88} = 1$ in Figure 9(b). Therefore the main diagonal represents the fourth byte 0x0F. The one bin shift problem mentioned in Gray coding reflects here that we cannot always get eight ones in a block if we directly apply the standard Gray coding. Shifting the mapping by one solves this problem and perfectly matches our following decoding process. By changing the all-1 data bits in the transmitted packet, we can derive the entire mapping for interleaving as shown in Figure 9(b). For other parameters, the deinterleaving process is similar. The only difference is that the block size becomes $\frac{4}{CR} \times (SF - 2DE)$. As a result, we summarize the deinterleaving process as

$$c_{i,j} = b_{j,(i+j-1)\%(SF-2DE)+1}, \tag{13}$$

where $c_{i,j}$ is the jth bit of the ith codeword after deinterleaving, $b_{j,i}$ is the ith bit of the jth symbol after Gray coding, and $i \in \{1, 2, \ldots, \frac{4}{CR}\}, j \in \{1, 2, \ldots, SF - 2DE\}$.

Hamming Decoding. After deinterleaving, we get the encoded data in the form of codewords, but the position of data bits and parity bits in a codeword are still unknown. LoRa provides four valid CR values ($\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$, and $\frac{4}{8}$), which determine the codeword length and thus FEC strength. When CR is set to $\frac{4}{7}$ or $\frac{4}{8}$, LoRa applies Hamming(7, 4) code or extended Hamming code with one additional parity bit. When CR is set to $\frac{4}{5}$ or $\frac{4}{6}$, LoRa can detect bit errors but cannot correct them. First, we assume that a nibble with code rate $\frac{4}{8}$ is protected by the standard Hamming(8, 4) code. But our final analysis results show that this assumption requires some modifications. To determine the position of each data/parity bit in the codeword, we could vary the sending bytes but keep one specific bit fixed. For example, to test the position of the LSB of the fourth byte, we set the transmitting bytes as 0x01, 0x03, 0x05, $\cdots$, 0x0F.[9] Then the bit that is always 1 in the interleaving block is our target, i.e., LSB in this case. We find that the four LSBs in the codeword are data bits while the four MSBs are parity bits, which differs from the standard Hamming(8, 4) code $p_1p_2d_1p_3d_2d_3d_4p_4$, where d_i is the ith data bit and p_i is the ith parity bit. Equation set (14) shows the relation between data bits and parity bits in standard Hamming(8, 4) code,

$$\begin{aligned} p_1 &= d_1 \oplus d_2 \oplus d_4 \\ p_2 &= d_1 \oplus d_3 \oplus d_4 \\ p_3 &= d_2 \oplus d_3 \oplus d_4 \\ p_4 &= d_1 \oplus d_2 \oplus d_3 \oplus d_4. \end{aligned} \tag{14}$$

We denote the four LSBs as d_1, d_2, d_3, d_4 sequentially. To find which bit in the MSBs is p_i, we vary the data bits to keep $p_i = 1$. For example, selecting the codewords with $d_1 \oplus d_2 \oplus d_4 = 1$, the parity bit that always equals "1" is the parity bit p_1. Similarly, the positions of p_2 and p_3 are derived. However, the remaining parity bit does not fit for the definition of p_4. After careful observation, we find that it is a parity covering d_1, d_2 and d_3, i.e.,

$$p_5 = d_1 \oplus d_2 \oplus d_3. \tag{15}$$

For other code rates, we can derive the bit position similarly. When $CR = \frac{4}{7}$ is used, parity bit p_1 is abandoned. When $CR = \frac{4}{6}$ is used, parity bits p_1 and p_2 are abandoned. When $CR = \frac{4}{5}$ is used, there is only one parity bit and it is natural to use p_4 to cover all bits. The conclusion does not change when SF varies. Figure 10 shows the bit positions for different code rates. Note that LoRa applies non-standard Hamming code, the naming number of parity is arbitrary and our naming is just one kind of them.

Dewhitening. In Section 4.3, we assume the dewhitening operation happens after "G" and we prove that the dewhitening position does not affect the final results. We here discuss the "correct" position for dewhitening. The process of deinterleaving and Hamming decoding has been

[9]Note that we cannot directly control parity bits.

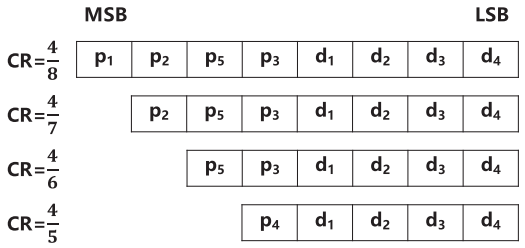

Fig. 10. Parity bits and data bits position in a codeword with CR = $\frac{4}{8}, \frac{4}{7}, \frac{4}{6}, \frac{4}{5}$.

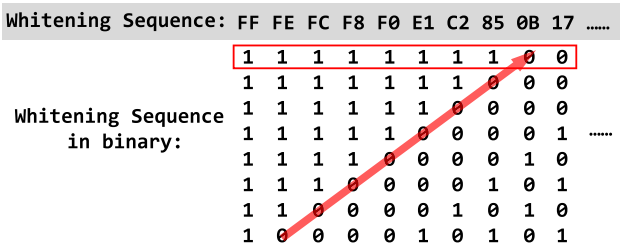

Fig. 11. The whitening sequence of "G→I→H→W."

interpreted clearly above. We move the position of "W" to the other two positions and send all-zero bytes to derive the whitening sequence under different order selections. We find that "G→W→I→H" gives different whitening sequences for various combinations of SF and CR while the whitening sequence of "G→I→W→H" and "G→I→H→W" keep the same. Using the same sequence for all packets is more reasonable. Thus we first exclude "G→W→I→H." Then we need to choose the correct order from "G→I →W→H" and "G→I →H→W." LoRa chip datasheet [22] mentions that the whitening sequence in FSK mode is generated by a **Linear Feedback Shift Register (LFSR)**. We guess that there also exists a LFSR generating the whitening sequence for LoRa mode. We apply the Berlekamp–Massey algorithm [38] on the whitening sequences of the two decoding orders to obtain the corresponding LFSR. For "G→I→W→H," no matter how we change the input order of the bits (e.g., LSB first or MSB first), the minimal LFSR size we get from the Berlekamp–Massey algorithm is at least 64. But we know $2n$ bits sequence could always be constructed from an n-bits LFSR. Any sequence containing 128 bits could be represented as the output of a 64-bits LFSR. The LFSR of the "G→I→W→H" sequence is too long. Carefully checking the whitening sequence of "G→I→H→W" as shown in Figure 11, we find that it is not like typical random bits, and each byte seems to be the state of an LFSR. We collect the MSBs of the sequence bytes, as shown in the red box of Figure 11, to run the Berlekamp–Massey algorithm. The LFSR polynomial for deriving such sequence is $x^8 + x^6 + x^5 + x^4 + 1$. It can be seen from the literature [39] that it is the maximal-length polynomial for 8-bits shift-register. Figure 12 shows the structure of the LFSR, and we can see that all the eight bits of the register compose a byte of the whitening sequence, and each bit of the register is used to whiten one-bit data. Since the whitening sequence of "G→I→H→W" can be generated by a LFSR with a much smaller length, we consider it the correct order.

4.5 CRC

After four steps of Gray coding, deinterleaving, Hamming coding, and dewhitening, the raw LoRa PHY packets are shown. In previous sections, we send packets in implicit header mode and disable CRC check. We next try to analyze the CRC algorithm used in the LoRa payload. From our observation in Section 4.2, CRC checksum occupies 16 bits at the tail of a packet. Therefore, we first

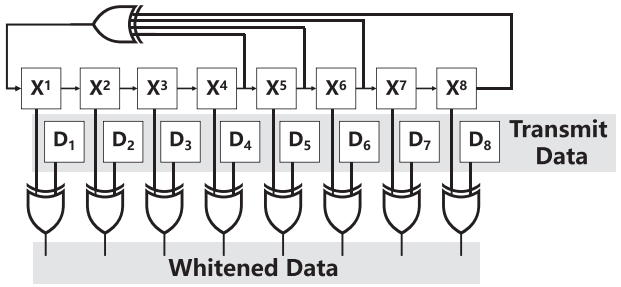

Fig. 12. LFSR $x^8 + x^6 + x^5 + x^4 + 1$ used in LoRa.

send explicit header and implicit header packets with the same data content to check the coverage of CRC. The results show that the CRC is only performed on the payload. Calculating the CRC checksum of a series of bits, in theory, is considering it as a large binary number and calculating the remainder with respect to a "divisor." The divisor is usually represented as a polynomial based on $GF(2)$ (Galois field of two elements), e.g., $x^3 + x + 1$. Analyzing the CRC part of LoRa PHY is indeed finding the polynomial used. A feature of CRC is that for a polynomial of degree n, if the original dividend is 1, then the remainder is the polynomial itself. If we construct a packet only with last bit equaling 1, then the CRC checksum in the last two bytes tells us exactly what polynomial is used. Meanwhile, the CRC polynomial cannot be randomly selected; it must follow some principles to ensure some specific requirements. Therefore, it is natural to choose the polynomial from standard CRC polynomials. Comparing the polynomial we derive and the standard polynomial sets, it takes little effort to find the actual polynomial used. Based on the above analysis, we enable the payload CRC in implicit header mode and set the last two bytes of payload as 0x0001, 0x0080, 0x0100, and 0x8000, respectively (other bytes are set to zero). Since we are not sure the endian and LSB-MSB order in CRC hardware implementations, we test the four possible combinations. The result, however, surprisingly, is beyond our expectation. Whatever we set in the last two payload bytes, CRC checksum are exactly the same with the last two payload bytes. In common CRC implementation, before doing the remainder calculation, it will pad n zero bits after data to ensure that the last n bits are also under full CRC protection.[10] The phenomenon we observed means that the zero-padding step is not implemented in LoRa PHY. Therefore, we send 0x0001, 0x0080, 0x0100, 0x8000 at the third and fourth bytes from last to derive the polynomial. The received CRC bytes are 0x1021, 0x9188, 0x3331, and 0x1B98, respectively. 0x1021 refers to the polynomial named CCITT-16, being $x^{16} + x^{12} + x^5 + 1$. We apply CRC check with this polynomial and test packets with random bytes. The CRC checksums calculated by our implementation are consistent with the CRC bytes in all samples. Due to the characteristic of CRC, it is hard for a wrong CRC implementation to have the same checksum with the correct one even in a small group of samples, which means our result is correct.

4.6 Header

The left unknown part about LoRa PHY is the header. In Section 4.2, we have already known that the header in explicit mode has 20 bits. Our goal is to find the organization and meaning of the 20 bits. **Payload Length (PL)** is said to be encoded in the header and occupies at least 8 bits, because the maximal payload length is 255. We vary the payload length and try to decode the packet using the same decoding process in implicit header mode assuming the header is also whitened. Unfortunately, we are not able to recover the data bytes in implicit header mode. The

[10]Though the code level implementation may not explicitly contain the zero-padding step [40].

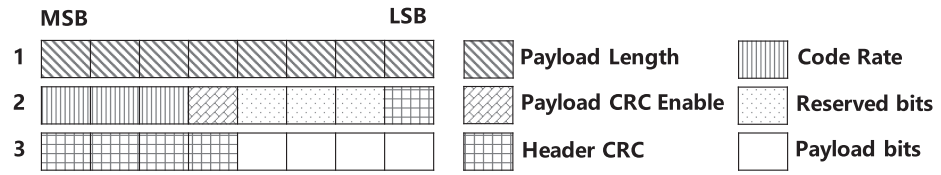

Fig. 13. Header structure of LoRa in the first three bytes.

problem comes from the whitening sequence used for header. Nonetheless, by observing the bits that change with the payload length, we can identify the position of PL. Note that in the first 2.5 bytes, only the bits of PL and header CRC will change when varying packet payload length. We observe that the bits in positions 1–8 and 16–20 are possible PL bits. Additionally, the intermediate results show us that the header is not whitened and the first byte of a header before dewhitening is exactly PL. Our derived LFSR can only generate 255 possible whitening bytes. There are no more additional whitening bytes for header dewhitening. Hence it is reasonable not to whiten the packet header. Our following tests on finding other bits strengthen this assumption. Therefore, we do not apply dewhitening operation on header in the following. Changing one parameter and fixing other parameters, the position of code rate and payload-CRC-enable bit are determined similarly. The code rate bits are 001, 010, 011, 100 for $CR = \frac{4}{5}, \frac{4}{6}, \frac{4}{7}, \frac{4}{8}$, respectively. The payload-CRC-enable bit is set to 1 if setting payload CRC on. In our experiments, there are five bits changing rapidly when setting different parameters. We assume these five bits as header CRC. There are three bits keeping zero whatever the parameter setting is, and we consider them as reserved bits. Figure 13 illustrates the header structure in the first three bytes.

We failed to find a standard CRC5 polynomial satisfying the header CRC results using the method in Section 4.5. But thanks to the linearity of CRC algorithm, we could use a CRC matrix to equivalently represent the CRC calculation.[11] We denote 12 parameter bits (PL, CR, and payload-CRC-enable bit) as a bit vector v_1. We denote the five header CRC bits as a bit vector v_2. Our target is to find a matrix M satisfying $v2 = M \cdot v_1$. Since the value of v_1 is under our control, we can design a series of v_1, say, $v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(12)}$. They form a matrix V_1. The relative v_2 series is $v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(12)}$. They form a matrix V_2. Therefore ,$V_2 = M \cdot V_1$. If V_1 is an identity matrix, then $M = V_2$.

How to make an identity matrix V_1? Despite the fact that we can control the header content, we cannot finely control the state of each bit. In our case, it seems impossible to send a packet with v_1 containing only a single "1." However, to our surprise, the LoRa chip unexpectedly supports *sending a packet with zero payload length.* The eight PL bits then become all zeros. If we disable payload CRC and set $CR = \frac{4}{5}$ or $CR = \frac{4}{6}$ or $CR = \frac{4}{8}$, then only one code rate bit in v_1 is 1. Is it possible that only the payload-CRC-enable bit is "1"? The answer is again using the linearity of CRC. Suppose we send two packets with zero payload and $CR = \frac{4}{5}$. Packet A has payload CRC but packet B does not. Then the corresponding bit vectors are $v_1^A = 000000000011, v_2^A = 01100, v_1^B = 000000000010, v_2^B = 00111$. Therefore,

$$\begin{aligned} M \cdot (000000000001) &= M \cdot \left(v_1^A \oplus v_1^B\right) \\ &= \left(M \cdot v_1^A\right) \oplus \left(M \cdot v_1^B\right) \\ &= v_2^A \oplus v_2^B \\ &= 01011. \end{aligned} \tag{16}$$

[11]Here we ignore the reserved bits.

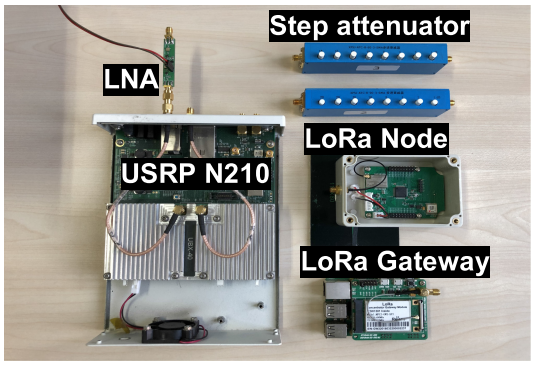

Fig. 14. Devices in our experiments: a USRP N210 SDR device with additional LNA, step attenuators for measuring sensitivity in wired environments, commodity LoRa nodes, and a commodity LoRa gateway.

A similar process can be applied for PL bits. Finally, we summarize the header CRC calculation as $v_2 = M \cdot v_1$, where

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{17}$$

5 IMPLEMENTATION

We implement the entire LoRa PHY in C++ on Software-Defined Radio platform GNU Radio. We also provide a MATLAB version code for simulation purposes. We design the receiver as two separated blocks: demodulator and decoder (for the transmitter: modulator and encoder). Thus, other new demodulation mechanisms can be integrated into our implementation in the future without changing the decoder block. For example, if we replace the LoRa demodulator with a collision demodulator, then we can decode collision packets directly without changing the decoding block logic. All the experiments in Section 6 are conducted on USRP N210 in real time (we do not save the raw baseband signals for offline processing). The USRP is connected to a laptop with Ubuntu 19.10, i5-7200U CPU, and 8 G RAM. We use commodity LoRa devices to evaluate the performance of our implemented LoRa PHY, including LoRa end nodes with SX1268 [25]/SX1278 [22] and Raspberry-Pi 3B+ LoRa gateways (RPI) with SX1301 [37]. Figure 14 shows the devices used in our experiments. The step attenuator is used in wired experiments for sensitivity measurements.

6 EVALUATION

6.1 Decoding Success Rate

We verify the effectiveness of our implemented LoRa PHY by testing if it can decode the LoRa packets transmitted from the commodity LoRa devices. We configure a commodity LoRa device to repeatedly transmit random packets to our implemented LoRa receiver. The ground truth of the transmitted bytes are recorded through serial port. For a comprehensive evaluation, we tune the LoRa transmitter with different parameters, including six SFs ($7 \sim 12$), four CRs ($\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$, $\frac{4}{8}$), and two header modes (explicit/implicit), and with/without CRC, i.e., $6 \times 4 \times 2 \times 2 = 96$ combinations. We use a carrier frequency of 475 MHz and a bandwidth of 250 kHz.[12] The payload length is set to

[12]Carrier frequency and bandwidth do not affect the decoding logic.

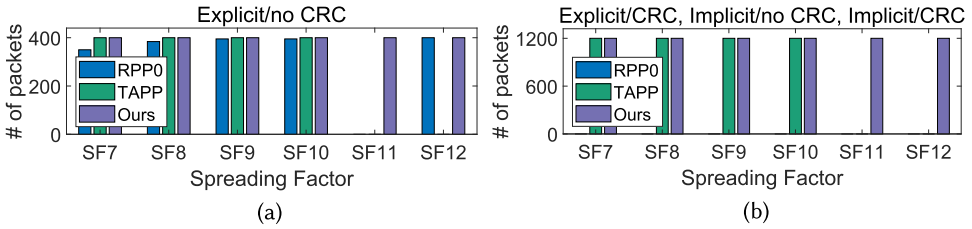

Fig. 15. Decoding success rate test. A total of 9,600 packets are sent, and all code rates ($\frac{4}{5}$, $\frac{4}{6}$, $\frac{4}{7}$, $\frac{4}{8}$) are tested. RPP0 cannot decode any SF11 packets. TAPP cannot decode any SF11/12 packets. Our decoder could decode all packets. (a) Number of correctly received packets of three implementations in explicit header mode with CRC off. (b) RPP0 does not support the configuration of {explicit header mode with CRC on, implicit header mode with CRC off, implicit header mode with CRC on} and therefore receives none.

16 bytes. For each configuration, the transmission repeats 100 times. We place the LoRa transmitter close to the USRP receiver, and thus all signals are received with high SNRs. Since the communication environment is ideal, any packet missing reflects the incompleteness of implementation.

The result of the experiment is shown in Figure 15. Among all the configurations, BR only works at SF8 with implicit header, while RPP0 only supports the explicit header. Neither of these two methods supports CRC. As the decoding processes of BR and TAPP are both computationally expensive, they failed to work with large packet length under high transmission rate. Specifically, BR cannot decode any packets longer than 5 bytes, and TAPP starts losing packets when the duty cycle of the transmitter is higher than 0.5. Due to all these limitations, the overall PRR of BR only reaches 4.2%. Besides, we can see that TAPP fails to decode any SF11/SF12 packets, and RPP0 cannot resolve SF11 packets. In summary, RPP0 covers 1,924/9,600 $\approx$ 20.0% packets and TAPP covers 6,400/9,600 $\approx$ 66.7% packets. On the contrary, our decoder can decode all packets under any LoRa configurations.

6.2 Sensitivity

In this section, we verify the sensitivity of our LoRa decoder. We connect a commodity LoRa transmitter to our decoder through a 20-m RF cable with two *step attenuators*. Thus, we can finely control the signal attenuation by adjusting the value of these two *step attenuators*.

Using the value of the *step attenuators* as the channel attenuation is not accurate, because the wired links, including RF cables and connectors, also introduce attenuation leading to sensitivity estimation errors. Therefore, before the experiment, we use a spectrum analyzer, Keysight N9322C, to precisely estimate and calibrate the link attenuation. Then, we set the transmitting power to its lowest, i.e., −7.9 dBm, to avoid wireless channel leakage. We experiment with the configuration of SF8 and bandwidth of 250 kHz. Since BR fails to decode any packets longer than four bytes, we do not compare its performance for the rest of this section. Figure 16(a) shows the sensitivity results of four SDR decoders. The criteria of sensitivity in LoRa is defined as the minimal RSSI achieving PRR > 90% when the payload is 32 bytes. As RPP0 and TAPP do not leverage the LoRa features for demodulation, their PRR drops quickly as the RSSI of the received signal goes down. The sensitivities of RPP0 and TAPP are −106 and −108 dBm, respectively. In contrast, the performance of our implemented decoder remains stable and achieves a sensitivity as low as −126 dBm.

We further conduct an experiment under a larger SF, i.e., SF12. The result is shown in Figure 16(b). Our measured sensitivity is −142 dBm, which approaches the optimal sensitivity of LoRa. Interestingly, we find that the sensitivity of the RPI gateway is only −139.5 dBm. That is because the RPI gateway does not adopt the optimal design of the LoRa gateway. Therefore, our decoder has better performance than it.

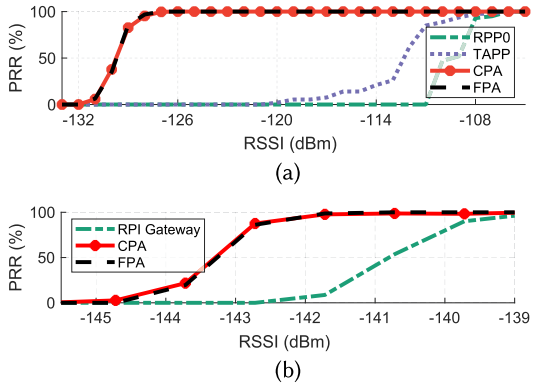

Fig. 16. Sensitivity tests of different decoders with (a) 10-byte payload, SF8, BW = 250 kHz and (b) 32-byte payload, SF12, BW = 125 kHz.

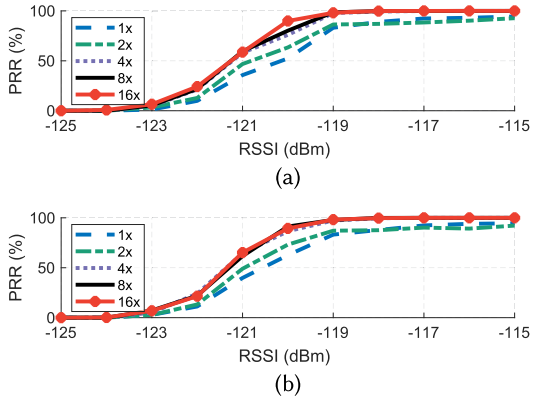

Fig. 17. Zero-padding ratio influence on (a) CPA and (b) FPA with $k = 8$.

6.3 Influence of Zero-Padding

Then we verify the validity of our proposed peak refinement strategy, which improves the frequency resolution by zero-padding. We use the same experimental setup as Section 6.2, evaluating both the FPA and CPA methods. Figure 17 shows the receiver performance under different levels of RSSI and zero-padding ratios. For both FPA and CPA, we see that PRR increases with higher zero-padding ratio r. The performance improvement for r from 1 to 4 is observable, while for $r \geq 4$ the performance improvement becomes negligible.

6.4 Encoder Test

In this section, we present the evaluation of our implemented LoRa encoder, which is a reverse version of the LoRa decoder. The only difference in encoder implementation comparing to the decoder is the redundant bytes for filling the last $\frac{4}{CR}$ symbols. Though these bytes seem fixed in LoRa chip implementation, they are meaningless on the receiver side. In our implementation, we pad zeros to fill the left bytes. When the encoder is up, it opens a UDP port and waits for data. We send 16 bytes to the encoder process through a python script. After that, the generated LoRa signal is sent from USRP to a LoRa end device. We check the serial port outputs from the device to see if the data are correctly transferred. We use different parameters (i.e., different SFs, CRs, and

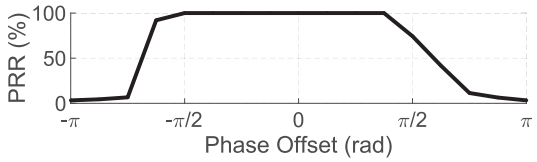

Fig. 18. LoRa chip's PRR when receiving specially constructed LoRa signals with different values of phase misalignment.

bandwidth) for sending data. Besides, for each parameter setting, the transmission is repeated 100 times. The final results show that 9,600 packets sent from our encoder could all be successfully decoded, which from another direction reveals that our analysis and implementation are reliable.

6.5 Influence of Phase Misalignment

In Section 3, we propose FPA and CPA for LoRa demodulation. We prove that the performance of the two methods is close to the IDEAL demodulation theoretically. That is because they are resistant to phase jitter caused by non-perfect hardware and multi-path. So what is the actual demodulation method implemented in commodity LoRa chip? Does it adopt FPA and CPA? Interestingly, we find that the demodulation algorithm used in the LoRa chip is not as good as FPA or CPA. We construct LoRa packets containing specially designed five bytes. As a result, the four consecutive data symbols are the same with $f_{start} = 0$, i.e., a symbol with a sharp frequency drop in the middle. Such a symbol is most vulnerable to phase misalignment between the two chirp segments. Before sending the packet via SDR, we manually add phase offset on the two segments. Our receiver using FPA and CPA are resistant to phase misalignment and their PPR is 100%. However, we find that the LoRa gateway faces frequent packet loss when receiving packets with significant phase misalignment. Figure 18 shows PRR with respect to phase misalignment. Its PRR is nearly 100% when the phase offset is between $(-\frac{\pi}{2}, \frac{\pi}{2})$. But it drops quickly when the phase offset becomes large. The result means the LoRa chip adopts a weaker demodulation method than our methods. In other words, the commodity LoRa chip is vulnerable to "*phase misalignment attack*." Despite the fact that this kind of "attack" is hard to accomplish, any small defects are worth noting in the security area. We conjecture that the LoRa chip may compensate a fixed phase offset $\Delta\varphi$ during the demodulation process, e.g., it estimates $\Delta\varphi$ from the preamble and applies $\Delta\varphi$ to the following data symbols. If $\Delta\varphi$ changes, then the LoRa chip fails to demodulate the packet correctly. Thus, the PRR drops when the phase offset changes. Of course, we do not know the internal implementation in LoRa chip unless Semtech publicly releases the documents. But, in summary, the LoRa chip demodulation implementation is weaker than our FPA and CPA method.

6.6 Outdoor Test

We finally compare our decoder with a commodity RPI gateway in a real-world environment. We place the receiver outside a window on the second floor. We carefully select the locations of the transmitters, assuring that there is no Line-of-Sight path between the transmitter and the receivers, which reflects the common situation of LoRa communication. The transmitter repeatedly sends a 32-byte packet in explicit header mode with setting SF12, BW = 125 kHz, CR = $\frac{4}{8}$, and CRC enabled. We lift the transmitter to 1.8 m in height with a tripod. Figure 19 shows the experiment setting mentioned above. Figure 20 shows PRR in different communication distances. Set PRR = 95% as a bar, our decoder supports a communication range as long as 3,600 m, while the commodity RPI gateway only reaches the maximum of 2,800 m. An interesting detail is the PRR fluctuation of RPI gateway in 2,400 m. We guess that the reason is the phase offset caused by the complex

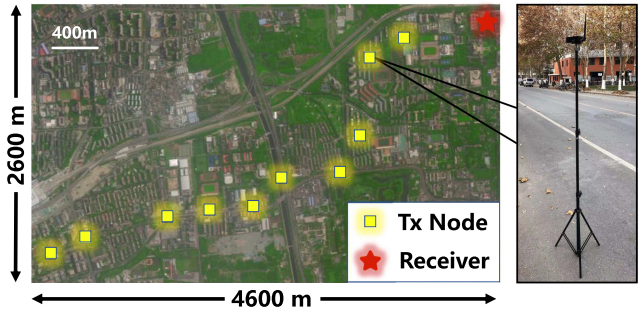

Fig. 19. Deployment of the transmitter and receiver. The transmitter is placed on a 1.8-m tripod.

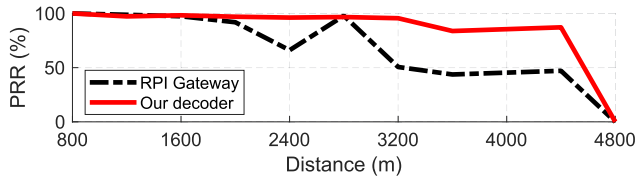

Fig. 20. Outdoor communication range experiments.

environment. As mentioned in Section 6.5, the commodity LoRa chip is vulnerable to phase misalignment problem. Therefore our decoder seems more reliable in outdoor communications.

7 RELATED WORK

Wireless protocol reverse engineering. The rapid development of Internet of Things brings a plurality of proprietary protocols. The endogenous driving force to develop closed protocols instead of using existed standard protocols comes from many reasons. For example, a proprietary protocol could save license fees for the Internet of Things manufacturers, or the new protocol provides specific functions targeting at specific application and embedded device. However, deviating from standard specifications sometimes paves the way for attackers, since the manufacturers might introduce some insecure designs. Meanwhile, these protocols are usually undocumented and closed to the public, which attracts network researchers to reverse engineer them, including Cryptographic RFID [41, 42, 53], In-Car wireless network [43], Apple Wireless Direct Link ad hoc protocol [44, 45], and so on. The reverse engineering on these protocols deepens our understanding of existing network and reveal flaws in protocol design. Our work also aims to give other researchers a more complete understanding of the LoRa physical layer and provide a better tool for analyzing the LoRa network.

Works related to LoRa PHY. LoRa provides a unique modulation offering long range, which is appealing in many applications. Traditional backscatter technology suffers from the communication range, which is limited to meter level. Recent works on backscatter combining LoRa [12, 47, 52] improve the range to the kilometer level. Meanwhile, adopting LoRa-like modulation, Netcatter [13] supports 256 current backscatter transmissions. Due to low data rate and large coverage characteristics of LPWAN, the network capacity is relatively small [46], which results in crucial collision problems. Many efforts have been made to improve the LoRa network throughput. Other kinds of representative work are collision decoding [1–4, 9, 10, 27, 49] and weak signal decoding [50, 51]. They mainly rely on the uniqueness of LoRa PHY demodulation to

separate chirps to difference packets. Our work gives a deeper understanding of LoRa PHY and will boost LoRa PHY-related research.

8 CONCLUSION

This article presents a comprehensive understanding of LoRa demodulation and decoding and reveals fundamental reasons for the performance gap between existing works and commodity LoRa. This work is the first complete LoRa PHY implementation with a provable performance guarantee to the black-box commodity LoRa chip. We enhance the demodulation to achieve extremely low SNR (−20 dB) decoding with a theoretical performance guarantee. Also, we derive the order and parameters of decoding operations, including dewhitening, error correction, deinterleaving, and so on, by leveraging officially known facts of LoRa and packet manipulation. Moreover, we implement the first complete real-time SDR LoRa PHY on the GNU Radio platform. The evaluation shows that our method can achieve (1) a 100% decoding success rate while existing methods can support at most 66.7%; (2) −142-dBm sensitivity, which is the limit of the commodity LoRa; and (3) a 3,600 m communication range in the urban area, similarly to commodity LoRa under the same setting.

APPENDIX

SER Calculation Model. Suppose the signal is transmitted through an AWGN channel, where the noise follows the complex Gaussian distribution, i.e., $\mathcal{CN}(0, \sigma^2)$. Then, after dechirping, the noise distribution turns to $\mathcal{CN}(0, M\sigma^2)$, where M is the number of samples within the demodulation window. The maximal height of noise has a small variance and can be approximated as $c\sigma$, where c is a constant. Suppose the data peak height h_d follows Gaussian distribution $\mathcal{N}(\mu_d, \sigma_d^2)$. We have

$$SER = P(h_d < c\sigma) = Q\left(\frac{c\sigma - \mu_d}{\sigma_d}\right), \tag{18}$$

where Q is the tail function of the standard normal distribution. Denote SNR as $\Gamma = \frac{A^2}{\sigma^2}$, where A is the signal amplitude.

IDEAL. Since multiplying and FFT are linear, the complex data peak in IDEAL follows complex Gaussian distribution $\mathcal{CN}(h, M\sigma^2)$, where $h = MA$. The peak height follows Rice distribution, which can be approximated as Gaussian distribution $\mathcal{N}(\mu_1, \sigma_1^2)$, where $\mu_1 = \sigma\sqrt{\frac{M\pi}{2}}L_{\frac{1}{2}}(-\frac{h^2}{2M\sigma^2})$ and $\sigma_1^2 = 2M\sigma^2 + h^2 - \mu_1^2$. $L_{\frac{1}{2}}(\cdot)$ is Laguerre polynomial. According to Reference [47], the maximal height of other $M-1$ noise height is approximately $\sqrt{2M\sigma^2 H_{M-1}} = c_1\sigma$, where $H_l = \sum_{i=1}^{l}\frac{1}{i}$ depicts the lth harmonic number. Taking μ_1, σ_1, and c_1 into Equation (18), we derive the SER of IDEAL as follows;

$$SER_1 = Q\left(\frac{\sqrt{2H_{M-1}} - \sqrt{\frac{\pi}{2}}L_{\frac{1}{2}}(-\frac{M\Gamma}{2})}{\sqrt{2 + M\Gamma - \frac{\pi}{2}L_{\frac{1}{2}}^2(-\frac{M\Gamma}{2})}}\right). \tag{19}$$

FPA. If the phase compensation $\theta = 0$, then the frequency domain after adding equals IDEAL. Shifting the noise by θ does not affect the zero-mean noise distribution. Therefore, the noise distribution after adding can still be considered as $\mathcal{CN}(0, M\sigma^2)$, which means the maximal noise height is $c_2\sigma$, where $c_2 = c_1$. The data peak in FPA follows $\mathcal{CN}(h_1 + e^{j\theta}h_2, M\sigma^2)$, where $h_1(h_2)$ is the first (second) chirp segment height and $h_1 + h_2 = h$. Let $h_3 = |h_1 + e^{j\theta}h_2|$. The peak height follows Rice distribution, which can be approximated as $\mathcal{N}(\mu_2, \sigma_2^2)$, where $\mu_2 = \sigma\sqrt{\frac{M\pi}{2}}L_{\frac{1}{2}}(-\frac{h_3^2}{2M\sigma^2})$, $\sigma_2^2 = 2M\sigma^2 + h_3^2 - \mu_2^2$. SER of FPA is related to the symbol transmitted, and here we simplify the sending symbol with $h_1 = h_2 = \frac{h}{2}$. The best case is that $\theta = 0$ and $h_3 = h$. The worst case is that $|\theta| = \frac{\pi}{k}$ and

$h_3 = h\cos(\frac{\pi}{2k})$. We consider the average of them $h \cdot \frac{1+\cos(\frac{\pi}{2k})}{2} = h\cos^2(\frac{\pi}{4k})$ as final h_3. Therefore, the SER of FPA can be approximated as

$$SER_2 = Q\left(\frac{\sqrt{2H_{M-1}} - \sqrt{\frac{\pi}{2}}L_{\frac{1}{2}}\left(-\frac{M\Gamma}{2}\cos^4(\frac{\pi}{4k})\right)}{\sqrt{2 + M\Gamma\cos^4\left(\frac{\pi}{4k}\right) - \frac{\pi}{2}L^2_{\frac{1}{2}}\left(-\frac{M\Gamma}{2}\cos^4\left(\frac{\pi}{4k}\right)\right)}}\right). \tag{20}$$

CPA. The two separated data peak heights in CPA follow Rice distribution, respectively. The sum of them have the approximated distribution $\mathcal{N}(\mu_3, \sigma_3^2)$, where $\mu_3 = \mu_3' + \mu_3'', \sigma_3^2 = 2M\sigma^2 + h_1^2 + h_2^2 - (\mu_3')^2 - (\mu_3'')^2, \mu_3' = \sigma\sqrt{\frac{M\pi h_1}{2h}}L_{\frac{1}{2}}(-\frac{h_1 h}{2M\sigma^2}), \mu_3'' = \sigma\sqrt{\frac{M\pi h_2}{2h}}L_{\frac{1}{2}}(-\frac{h_2 h}{2M\sigma^2})$. The noise peak height follows Rayleigh distribution. Denote the absolute value summation of two part of noise as $Y_i (i = 1, 2, \ldots, M-1)$. Y_i approximately follows $\mathcal{N}(\mu_Y, \sigma_Y^2)$, where

$$\mu_Y = \left(\sqrt{h_1} + \sqrt{h_2}\right)\sqrt{\frac{M\pi}{2h}}\sigma, \sigma_Y^2 = \frac{4-\pi}{2}M\sigma^2. \tag{21}$$

Let $S = \max_{i=1,2,\ldots,M-1} Y_i$. By Jensen's inequality [48],

$$\begin{aligned} e^{t\mathrm{E}(S)} &\le \mathrm{E}\left(e^{tS}\right) = \mathrm{E}\left(\max_i e^{tY_i}\right) \\ &\le \sum_{i=1}^{M-1}\mathrm{E}\left(e^{tY_i}\right) = (M-1)e^{\mu_Y t + \frac{1}{2}\sigma_Y^2 t^2}, \end{aligned} \tag{22}$$

where the last equality follows from the definition of the Gaussian moment generating function. Taking the logarithm on both sides of inequality (22), we have

$$\mathrm{E}(S) \le \frac{\ln(M-1)}{t} + \mu_Y + \frac{\sigma_Y^2 t}{2}. \tag{23}$$

For $t > 0$, according to the inequality of arithmetic and geometric means, we know

$$\mathrm{E}(S) \le \mu_Y + \sqrt{2\ln(M-1)}\sigma_Y = c_3\sigma, \tag{24}$$

where $c_3 = (\sqrt{h_1} + \sqrt{h_2})\sqrt{\frac{M\pi}{2h}} + \sqrt{(4-\pi)M\ln(M-1)}$. We use upper bound of $\mathrm{E}(S)$ to estimate the max noise peak height in CPA method. Consider the situation $h_1 = h_2 = \frac{h}{2}$; we have

$$SER_3 = Q\left(\frac{\sqrt{\pi} + \sqrt{(4-\pi)\ln(M-1)} - \sqrt{\pi}L_{\frac{1}{2}}(-\frac{M\Gamma}{4})}{\sqrt{2 + \frac{M\Gamma}{2} - \frac{\pi}{2}L^2_{\frac{1}{2}}(-\frac{M\Gamma}{4})}}\right). \tag{25}$$

REFERENCES

[1] Shuai Tong, Zhenqiang Xu, and Jiliang Wang. 2020. CoLoRa: Enabling multi-packet reception in LoRa. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'20)*. IEEE, 2303–2311.

[2] Xianjin Xia, Yuanqing Zheng, Tao Gu. 2020. FTrack: Parallel decoding for LoRa transmissions. *IEEE/ACM Trans. Netw.* 28, 6 (2020), 2573–2586. https://doi.org/10.1109/TNET.2020.3018020

[3] Zhenqiang Xu, Shuai Tong, Pengjin Xie, and Jiliang Wang. 2020. FlipLoRa: Resolving collisions with up-down quasi-orthogonality. In *Proceedings of the IEEE International Conference on Sensing, Communication and Networking (SECON'20)*.

[4] Zhe Wang, Linghe Kong, Kangjie Xu, Liang He, Kaishun Wu, and Guihai Chen. 2020. Online concurrent transmissions at LoRa gateway. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'20)*. IEEE, 2331–2340.

[5] Wenju Zhao, Shengwei Lin, Jiwen Han, Rongtao Xu, and Lu Hou. 2017. Design and implementation of smart irrigation system based on LoRa. In *Proceedings of the IEEE Globecom Workshops (GC Wkshps'17)*. IEEE, 1–6.

[6] Irfan Fachrudin Priyanta, Frank Golatowski, Thorsten Schulz, and Dirk Timmermann. 2019. Evaluation of LoRa technology for vehicle and asset tracking in smart harbors. In *Proceedings of the 45th Annual Conference of the IEEE Industrial Electronics Society (IECON'19)*, Vol. 1. IEEE, 4221–4228.
[7] Davide Magrin, Marco Centenaro, and Lorenzo Vangelista. 2017. Performance evaluation of LoRa networks in a smart city scenario. In *Proceedings of the IEEE International Conference on Communications (ICC'17)*. IEEE, 1–7.
[8] Umber Noreen, Ahcène Bounceur, and Laurent Clavier. 2017. A study of LoRa low power and wide area network technology. In *Proceedings of the International Conference on Advanced Technologies for Signal and Image Processing (ATSIP'17)*. IEEE, 1–6.
[9] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yağan. 2017. Empowering low-power wide area networks in urban settings. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'17)*.
[10] Shuai Tong, Jiliang Wang, and Yunhao Liu. 2020. Combating packet collisions using non-stationary signal scaling in LPWANs. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'20)*.
[11] Yao Peng, Longfei Shangguan, Yue Hu, Yujie Qian, Xianshang Lin, Xiaojiang Chen, Dingyi Fang, and Kyle Jamieson. 2018. PLoRa: A passive long-range data network from ambient LoRa transmissions. In *Proceedings of the ACM Conference of the Special Interest Group on Data Communication (SIGCOMM'18)*.
[12] Vamsi Talla, Mehrdad Hessar, Bryce Kellogg, Ali Najafi, Joshua R. Smith, and Shyamnath Gollakota. 2017. LoRa backscatter: Enabling the vision of ubiquitous connectivity. *Proc. ACM Interact. Mobile Wear. Ubiq. Technol.* 1, 3 (2017), 1–24.
[13] Mehrdad Hessar, Ali Najafi, and Shyamnath Gollakota. 2019. Netscatter: Enabling large-scale backscatter networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*. 271–284.
[14] Adwait Dongare, Revathy Narayanan, Akshay Gadre, Anh Luong, Artur Balanuta, Swarun Kumar, Bob Iannucci, and Anthony Rowe. 2018. Charm: Exploiting geographical diversity through coherent combining in low-power wide-area networks. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN'18)*. IEEE, 60–71.
[15] Akshay Gadre, Revathy Narayanan, Anh Luong, Anthony Rowe, Bob Iannucci, and Swarun Kumar. 2020. Frequency configuration for low-power wide-area networks in a heartbeat. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*.
[16] Pieter Robyns, Peter Quax, Wim Lamotte, and William Thenaers. 2018. A multi-channel software decoder for the LoRa modulation scheme. In *Proceedings of the International Conference on Internet of Things, Big Data and Security (IoTBDS'18)*.
[17] Matthew Knight and Balint Seeber. 2016. Decoding LoRa: Realizing a modern LPWAN with SDR. In *Proceedings of the GNU Radio Conference*.
[18] Joachim Tapparel, Orion Afisiadis, Paul Mayoraz, Alexios Balatsoukas-Stimming, and Andreas Burg. 2020. An open-source LoRa physical layer prototype on GNU radio. In *Proceedings of the IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC'20)*. 1–5. DOI: http://dx.doi.org/10.1109/SPAWC48557.2020.9154273
[19] Josh Blum. 2016. LoRa modem with LimeSDR. Retrieved from https://myriadrf.org/news/lora-modem-limesdr/.
[20] RevSpace. DecodingLoRa. Retrieved from https://revspace.nl/DecodingLora.
[21] Alexandre Marquet, Nicolas Montavont, and Georgios Z. Papadopoulos. 2019. Investigating theoretical performance and demodulation techniques for LoRa. In *Proceedings of the IEEE IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'19)*. IEEE, 1–6.
[22] Semtech. 2020. Data Sheet SX1276/77/78/79, Rev. 7.
[23] LoRaWAN Specification v1.1. Retrieved from https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/.
[24] Symphony Link. Retrieved from https://www.link-labs.com/symphony.
[25] Semtech. 2019. Data Sheet SX1268, Rev. 1.1.
[26] Olivier Bernard, André Seller, and Nicolas Sornin. 2014. Low power long range transmitter. Patent EP 2 763 321 A1.
[27] Xiong Wang, Linghe Kong, Liang He, and Guihai Chen. 2019. mLoRa: A multi-packet reception protocol in LoRa networks. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP'19)*. IEEE, 1–11.
[28] Rajalakshmi Nandakumar, Vikram Iyer, and Shyamnath Gollakota. 2018. 3D Localization for sub-centimeter sized devices. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys'18)*.
[29] Yinghui Li, Jing Yang, and Jiliang Wang. 2020. DyLoRa: Towards energy efficient dynamic LoRa transmission control. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM'20)*. IEEE, 2312–2320.
[30] LoRaMac-node. Retrieved from https://github.com/Lora-net/LoRaMac-node.
[31] SX1302 LoRa Gateway project. Retrieved from https://github.com/Lora-net/sx1302_hal.
[32] ChirpStack. Retrieved from https://www.chirpstack.io/.
[33] LoRaWAN Server. Retrieved from https://github.com/gotthardp/lorawan-server.

[34] Charm Decoder. Retrieved from https://github.com/WiseLabCMU/charm-decoder.
[35] Mehrdad Hessar, Ali Najafi, Vikram Iyer, and Shyamnath Gollakota. 2020. TinySDR: Low-power SDR platform for over-the-air programmable IoT testbeds. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*. 1031–1046.
[36] TinySDR. Retrieved from https://github.com/uw-x/tinysdr.
[37] Semtech. 2017. Data Sheet SX1301, V2.4.
[38] Nadia Ben Atti, Gema M. Diaz-Toca, and Henri Lombardi. 2006. The berlekamp-massey algorithm revisited. *Applic. Algebr. Eng. Commun. Comput.* 17, 1 (2006), 75–82.
[39] Wikipedia. LFSR. https://en.wikipedia.org/wiki/Linear-feedback_shift_register.
[40] Ross Williams et al. 1993. A painless guide to CRC error detection algorithms. https://zlib.net/crc_v3.txt.
[41] P. Fraga-Lamas and T. M. Fernández-Caramés. 2017. Reverse engineering the communications protocol of an RFID public transportation card. In *Proceedings of the IEEE International Conference on RFID (RFID'17)*. 30–35.
[42] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. 2008. Reverse-engineering a cryptographic RFID tag.. In *Proceedings of the USENIX Security Symposium*, Vol. 28.
[43] Ishtiaq Rouf, Robert D. Miller, Hossen A. Mustafa, Travis Taylor, Sangho Oh, Wenyuan Xu, Marco Gruteser, Wade Trappe, and Ivan Seskar. 2010. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *Proceedings of the USENIX Security Symposium*, Vol. 10.
[44] Milan Stute, David Kreitschmann, and Matthias Hollick. 2018. One billion apples' secret sauce: Recipe for the Apple wireless direct link ad hoc protocol. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 529–543.
[45] Milan Stute, Sashank Narain, Alex Mariotto, Alexander Heinrich, David Kreitschmann, Guevara Noubir, and Matthias Hollick. 2019. A billion open interfaces for Eve and Mallory: MitM, DoS, and tracking attacks on iOS and macOS through Apple wireless direct link. In *Proceedings of the USENIX Security Symposium*. 37–54.
[46] Branden Ghena, Joshua Adkins, Longfei Shangguan, Kyle Jamieson, Philip Levis, and Prabal Dutta. 2019. Challenge: Unlicensed LPWANs are not yet the path to ubiquitous connectivity. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MobiCom'19)*. 1–12.
[47] Tallal Elshabrawy and Joerg Robert. 2018. Closed-form approximation of LoRa modulation BER performance. *IEEE Commun. Lett.* (2018).
[48] Pascal Massart. 2007. *Concentration Inequalities and Model Selection*, Vol. 6. Springer.
[49] Qian Chen and Jiliang Wang. 2021. AlignTrack: Push the limit of LoRa collision decoding. *IEEE 29th International Conference on Network Protocols (ICNP'21)*, 1–11. DOI: 10.1109/ICNP52444.2021.9651985
[50] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. 2021. NELoRa: Towards ultra-low SNR LoRa communication with neural-enhanced demodulation. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 56–68.
[51] Shuai Tong, Zilin Shen, Yunhao Liu, and Jiliang Wang. 2021. Combating link dynamics for reliable lora connection in urban settings. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 642–655.
[52] Jinyan Jiang, Zhenqiang Xu, Fan Dang, and Jiliang Wang. 2021. Long-range ambient LoRa backscatter with parallel decoding. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 684–696.
[53] Qianwen Miao, Fu Xiao, Haiping Huang, Lijuan Sun, and Ruchuan Wang. 2019. Smart attendance system based on frequency distribution algorithm with passive RFID tags. *Tsinghua Science and Technology* 25, 2 (2019), 217–226.

Received 24 February 2022; revised 19 May 2022; accepted 5 June 2022