OpenLoRa: Validating LoRa Implementations through an Extensible and Open-sourced Framework

Manan Mishra, Daniel Koch, Muhammad Osama Shahid, and Bhuvana Krishnaswamy, *University of Wisconsin-Madison;* Krishna Chintalapudi, *Microsoft Research;* Suman Banerjee, *University of Wisconsin-Madison*

https://www.usenix.org/conference/nsdi23/presentation/mishra

This paper is included in the Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation.

April 17–19, 2023 • Boston, MA, USA

978-1-939133-33-5

OpenLoRa: Validating LoRa Implementations through an Extensible and Open-sourced Framework

Manan Mishra
University of Wisconsin-Madison

Daniel Koch
University of Wisconsin-Madison

Muhammad Osama Shahid
University of Wisconsin-Madison

Bhuvana Krishnaswamy
University of Wisconsin-Madison

Krishna Chintalapudi
Microsoft Research

Suman Banerjee
University of Wisconsin-Madison

Abstract

LoRa is one of the most widely used LPWAN communication techniques operating in the unlicensed sub-GHz ISM bands. Its long range however also results in increased interference from other LoRa and non-LoRa networks, undermining network throughput due to packet collisions. This has motivated extensive research in the area of collision resolution techniques for concurrent LoRa transmissions and continues to be a topic of interest. In this paper, we verify the implementation and efficacy of four of the most recent works on LoRa packet collisions, in addition to standard LoRa. We implement *OpenLoRa*, an open-source, unified platform to evaluate these works and extensible for future researchers to compare against existing works. We implement each of the four techniques in Python as well as separate the demodulator and decoder to provide benchmarks for future demodulators that can be plugged into the framework for fair and easy comparison against existing works. Our evaluation indicates that existing contention resolution techniques fall short in their throughput performance in practical deployments, especially due to poor packet detection in low and ultra-low SNR regimes.

1 Introduction

LoRa is one of the most widely used Low Power Wide Area Network (LPWAN) technologies for IoT applications such as smart cities [1, 2], smart agriculture [3, 4], and industrial IoT [5, 6]. LoRa's popularity stems from its long operating range, low power consumption, low-cost, and ease of deployment [7–10]. Its long range however, is a double-edged sword as it also results in increased interference from other independently deployed LoRa networks, leading to poor network throughput due to packet collisions [11, 12]. Ghena et.al [13] show that LoRa falls short of meeting the requirements for a large variety of IoT applications due to two key reasons : (a) under-utilization of the network capacity and (b) lack of co-existence between networks.

Recently, a large number of *LoRa collision resolution* techniques have been proposed to address the above challenges: Choir [14], FTrack [15], NScale [16], CoLoRa [17], mLoRa [18], CIC [19], Pyramid [20], and AlignTrack [21]. These techniques develop novel LoRa de-modulation algorithms that can simultaneously decode multiple colliding LoRa packets to improve network throughput and address scalability challenge faced by LoRa networks.

In this paper, we seek to compare and verify the efficacy of state-of-the-art in LoRa collision resolution algorithms and ask the question, *"How effective are state-of-the-art collision resolution techniques in improving network throughput?"* . Our goal is to evaluate and analyze various existing techniques in a variety of important scenarios such as indoor/outdoor and low/high-SNR networks. Towards this evaluation, we have developed *OpenLoRa*, an extensible, open-source framework using Python to implement each of the demodulators and design an evaluation pipeline, along with extensive datasets that can be used for benchmarking future works in LoRa receiver designs. To this end, we pick four recent techniques, (i) FTrack [15], (ii) NScale [16], (iii) CoLoRa [17], and (iv) CIC [19] that provide public implementations.

The motivation for our paper stems from several key gaps in the available implementations and evaluations.

Lack of throughput evaluations. While increasing network throughput (in kilo bits per second) is their key goal, most LoRa packet collision resolution literature evaluates the performance of the demodulator only, which outputs data symbols rather than bits; this can perhaps be attributed to the difficult task of recreating LoRa's encoder and decoder. In the absence of the decoder, one can only evaluate the average symbol error rate, but not bit or packet error rates and hence network throughput in kbps. As we demonstrate in Section §5.1, lower symbol error rates do not necessarily translate to lower packet error rates and corresponding higher network throughput. In fact, seemingly lower symbol error rates compared to standard LoRa might still result in lower throughput!

Lack of co-existence evaluation. Although co-existence of LoRa networks has been identified as a key challenge [11–13], to the best of our knowledge existing literature has not studied the impact of interference due to other LoRa and non-LoRa

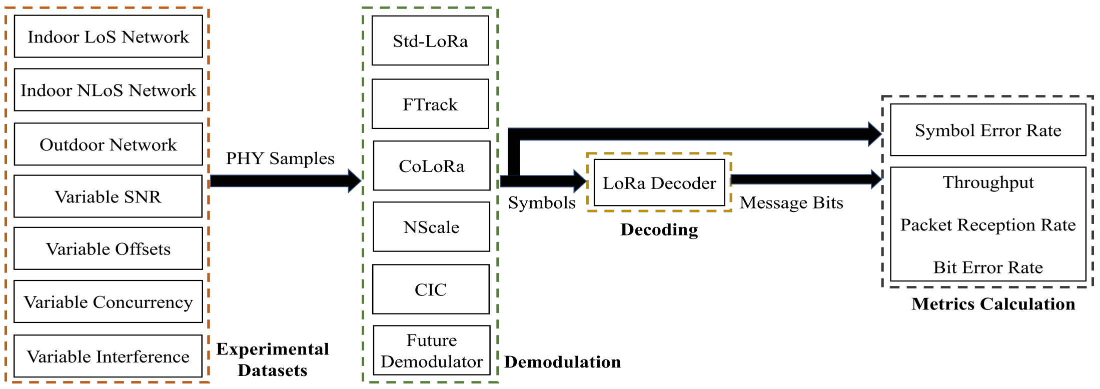


Figure 1: Overview of *OpenLoRa*, the proposed open-source framework

networks on the efficiency of collision resolution. In this paper, we design and perform a new set of experiments to measure the impact of LoRa interference from networks with different SF and non-LoRa interference such as FSK on the network throughput performance.

Lack of a uniform evaluation and a benchmark datasets. Understanding and analyzing the performance of various techniques in different settings is crucial to future research. Existing evaluations differ in terms of evaluation metrics, methodology, scenarios that impact performance such as signal-to-noise-ratio (SNR), indoor/outdoor settings, nature of traffic, effect of bursts etc. In some cases, these have only been studied using simulations. In this paper, we create a set of benchmark datasets spanning various important LoRa scenarios that can be used to evaluate uniformly. In our experiments we find that existing techniques under-perform compared to standard LoRa in very low SNR scenarios.

Implementation variability. Current implementations do not use a common tool: Python, GNURadio, and MATLAB are some of the tools used. Additionally, each of the implementations have a different data preprocessing methodology, making it a challenging task to input a sample file and determine the metrics of interest. Therefore, despite the availability of public code repository, it is a challenging task to input a new file and obtain the performance of the demodulator.

In order to address the above gaps, we have implemented an evaluation framework (depicted in Figure 1) with the goal of providing a common framework to benchmark existing methods. We believe that our extensible framework will help future researchers evaluate LoRa collision resolution techniques uniformly against prior works with common datasets, and analyze them. *OpenLoRa* includes a pipeline of four key stages. First, a suite of experimental datasets comprising received raw samples of LoRa transmissions obtained from various experimental deployments, specifically designed to evaluate various important aspects of collision resolution algorithms. Second, a uniform Python based interface to interact with each demodulator. A future LoRa demodulator algorithm can be simply plugged into this framework and compared against other implementations on the metrics of interest for real-world deployments. Third, a standard LoRa decoder that can convert symbols generated by any demodulator into packets, so that we can measure bit error rates, packet error rates, and throughput. Last but not the least we provide a suite of important metrics such as bit error rate, packet reception rate, and network throughput.

In summary, we make the following contributions towards our verification of state-of-the-art LoRa demodulators:

- We present *OpenLoRa*, an extensible, open-source framework to evaluate and compare different techniques that we hope future researchers will be able to use (provided in GitHub repository[1] as well as in a Docker container[2] enabling environment-independence to run the demodulators locally). Our framework comprises benchmark datasets, a standard interface to plug in various demodulators, a LoRa decoder that outputs bits and a suite of relevant metrics.

- We implement and verify the performance of four state-of-the-art LoRa collision resolution demodulators and standard LoRa, comparing their throughput (in kbps) improvements. We find a surprising fact that even though many techniques decode more symbols on average than standard LoRa, this does not necessarily translate to throughput improvements. As the network traffic increases in long-range outdoor scenarios, standard LoRa outperforms all existing techniques.

- In order to cross-validate the results reported in original works, we recreate the key experimental scenarios presented by each and compare the results. Our results are in line with the results in the respective papers evaluated. This extra effort validates the fidelity of our framework and implementation of various demodulators.

- We develop a web interface[3] for users to easily add new custom demodulation techniques for benchmarking and analyze the performance of implemented techniques.

[1]https://github.com/UW-CONNECT/OpenLora
[2]Linked in OpenLoRa Github page.
[3]`https://openlora.wisc.edu`

- We implement and validate the standard LoRa decoder in Python, allowing the comparison of different demodulation techniques based on end-to-end metrics such as throughput (in kbps) and the number of successfully received packets after error correction. We hope this openly accessible implementation enables future researchers to evaluate their works based on similar end-user metrics.
- We design new experiments to study the effects of collisions under extremely-low SNR, interference from LoRa networks with different spreading factor and non-LoRa interference such as Frequency-Shift-Keying from other networks on the throughput performance.

2 LoRa Demodulators Validated

In LoRa, data is modulated using a Chirp Spread Spectrum (CSS) scheme which confers it long range and sub-noise decoding ability. We present details on LoRa's modulation/demodulation and effects of collision on network throughput in Appendix A. Choir [14] is a pioneering work in LoRa collision resolution with the goal of improving network throughput. It leverages the inherent hardware imperfections in the radio of LoRa transmitters and distinguishes colliding packets by uniquely mapping their imperfections to the transmitters. mLoRa [18] leverages Successive Interference Cancellation to iteratively decode the symbols with the highest power and remove them from consideration. In this paper, we implement four demodulator algorithms that improve upon Choir and mLoRa to decode multi-packet collisions. We discuss these demodulators in the rest of the section.

2.1 FTrack [15]

FTrack is one of the first approaches to use time and frequency domain features to resolve LoRa collisions. FTrack relies on Short Time Fourier Transform (STFT) to obtain time and frequency features. FTrack proposes to apply STFT on the dechirped LoRa symbol to leverage the spread spectrum gain as well as to remove the linear change in frequency with time. Appendix B.1 explains how FTrack chooses an appropriately sized window and leverages time-frequency resolution to resolve collisions.

2.2 CoLoRa [17]

CoLoRa, similar to FTrack, leverages time offsets and frequency features to resolve collisions. CoLoRa observes that collided packets are misaligned in time and therefore have different lengths of symbol segments appearing in the demodulation window. This results in FFT peaks with heights proportional to the length of the symbol in the current demodulation window. CoLoRa also observes that the ratio of FFT peak between two consecutive windows remains the same throughout a packet. It uses these key insights to translate time offsets to frequency features and differentiate colliding packets. Details on CoLoRa's demodulation window choice and the use of peak ratios can be found in Appendix B.2.

2.3 NScale [16]

As the range increases, the SNR of LoRa packets decreases and the relative performance improvement of FTrack and CoLoRa degrades. NScale [16] focuses on decoding packet collisions at SNRs as low as -10dB. Similar to CoLoRa, it translates the timing offsets to frequency features and further amplifies the time offsets by non-stationary signal scaling. NScale's strength lies in its ability to decode and resolve LoRa packet collisions at SNRs below -10 dB. In Appendix B.3, we explain how NScale achieves sub-noise decodability.

2.4 Concurrent Interference Cancellation [19]

Concurrent Interference Cancellation (CIC) also leverages time and frequency domain analysis to decode multi-packet collisions. CIC identifies that due to Heisenberg's Time Frequency Uncertainty Principle, one can achieve either the best frequency resolution or best time resolution, not both. CIC attempts at getting the best of both resolutions by accumulating multiple windows of varying lengths, resulting in varying time and frequency resolutions. Appendix B.4 explains CIC's technique to resolve packet collisions.

3 Framework Implementation

We implement *OpenLoRa* and evaluate standard LoRa, FTrack, CoLoRa, NScale, and CIC in a uniform framework using Python as illustrated in Fig. 1. The extensible framework also provides the ability to add a new demodulator and evaluate its performance against the implemented techniques over the datasets collected over a range of scenarios.

We have built an easy-to-use web interface to help users analyze the implemented demodulators' performance in more detail. We also provide the option to plug-in one's own new demodulator for benchmarking, with a few simple steps. Some screenshots to walk through the web page are provided in Fig. 2. The homepage asks for user selection to either add a new demodulator or run existing techniques as shown in Fig. 2(a). Fig 2(b-c) show the flow to add a new custom demodulator with the user downloading the existing framework, adding their files, testing on a sample dataset and uploading to run and compare against already implemented techniques. Similarly, Fig 2(d-f) show the flow to analyze the existing demodulators in detail by choosing a scenario and configuration among those presented in Section §5 and presenting detailed data points as well as plots.

OpenLoRa has a pipeline of four major blocks: datasets, interface to the demodulators, decoder, and metrics. We describe each one of these blocks in detail below.

3.1 Datasets : Experimental setup for data collection

For a thorough and fair evaluation of the demodulation algorithms, a uniform set of data sample files is necessary. We deployed practical networks of LoRa nodes in varying configurations (SF and BW) and scenarios (Line-of-Sight, SNR) as shown in Fig 3. We believe the presented evaluation accounts for a comprehensive (but not exhaustive) set of conditions to assess the feasibility of real-world usage, and serve as a benchmark to gauge the performance of future work in this domain. We will make the datasets, along with the ground truth, publicly available for reproducibility. We believe this will help other researchers to evaluate their work on a variety of scenarios as well.

We used 20 battery-powered Adafruit Feather M0 with RFM95 [22] as LoRa transmitters deployed in the following settings, with a USRP B200 as the receiver. Unless otherwise mentioned, by default, each transmitter sends a known message, while the duty cycle and the load follows a Poisson distribution. The arduino code flashed onto Adafruit Feather M0 boards and the circuit diagram to setup the boards can be found in the github repository [4] under the folder Experiment_ Setup. At each of the locations (red dots in Fig. 3), the individual transmitters were verified to be reachable from the receiver i.e.,in the absence of collisions, LoRa packets from each of the transmitters were successfully received using the standard LoRa demodulator. Each data point in the evaluation results was averaged over multiple iterations of data transmissions (ranging from a minimum of 200 packets to over 6000 packets depending on the scenario). The details on each experimental setup and the methodology specific to each experiment can be found in the Appendix E. The three settings used in our experiments are:

1. **Indoor Line-of-Sight (LoS) :** This setting serves as a high-SNR scenario in our experiments. Twenty LoRa nodes were deployed in a 15m x 10m room, distributed uniformly in LoS with the receiver as shown in Fig. 3 (a). This setting emulates a deployment similar to a smart home, with IoT nodes distributed in a small space, many of which have LoS to the receiver. In this setup, we performed experiments which required precise control over collision parameters, parametric analysis with controlled time offset between colliding packets, concurrent collisions, and high SNR collisions.

2. **Indoor Non-Line-of-Sight (NLoS) :** This setting serves as a low-SNR setting inside a building spanning an area of 150 m x 75 m (per floor) over two floors. The transmitters were deployed as per Fig. 3 (c) and (d), showing the distribution of nodes on first and second floor respectively. The nodes were distributed in NLoS setting, separated from the receiver by multiple concrete walls, elevator shafts, and metal obstructions. This setting emulates a typical deployment of IoT nodes in an indoor office or factory building, with human movement as well as wireless traffic interfering with active transmissions. This setup was used to collect datasets for collisions with increasing aggregate transmission rate. We also performed controlled interference experiments here.

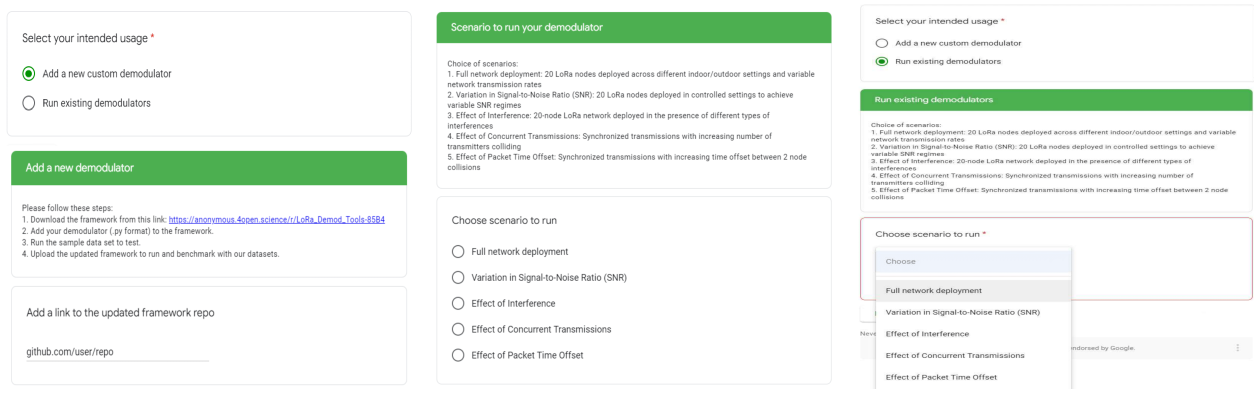
3. **Long-range outdoor :** This setting serves as an extremely low-SNR setting with nodes at distances of 1 to 8.25 km from the receiver. The nodes were distributed across urban areas and along a lake shore as seen in Fig. 3 (b). This setting emulates applications which particularly befit the use of LoRa modulation where communication over long distances is necessary, such as city monitoring applications or sensor deployment across huge agricultural fields. We collected datasets for various transmission rates in extremely low-SNR conditions.

[4]https://github.com/UW-CONNECT/OpenLora/tree/main/Experiment_ Setup

3.2 Demodulators Block

Fig. 13 in Appendix C.1 shows an overview of the Python implementation and how the demodulators were integrated into the overall flow of the framework. Appendix C.2 also describes the organization of the implementation code into Python modules. We thank the authors of each of the works presented here for sharing their implementations with us. In addition to re-implementing the code in Python, the following refinements were made to accept and pre-process a variety of datasets, as well as to reduce the computation time of demodulation using parallel processes:

FTrack : FTrack processes the entire input data file at once. While it is feasible for slices with a few packets, it is computationally intensive for real-life data capture with tens of million of samples. This also posed a challenge in the memory constraints for practical datasets such as those from our experiments. To mitigate this, we built a pre-processing block that separates out the active data transmission from silence periods based on energy thresholds of the signal and passes only the valid transmission data to the demodulator. FTrack's demodulator algorithm uses a number of threshold parameters for separating collisions such as peak power ratio, noise floor power, and ratio of stronger to weaker peaks, among others. These parameters need to be modified based on the input dataset for accurate functioning of the algorithm. We set these parameters based on our datasets empirically, however a formal procedure for the derivation of these thresholds would be highly beneficial for any user.



Select your intended usage *

[✓ Add a new custom demodulator]

Run existing demodulators

Add a new demodulator

Please follow these steps:
1. Download the framework from this link: https://anonymous.4open.science/r/LoRa_Demod_Tools-85B4
2. Add your demodulator (.py format) to the framework.
3. Run the sample data set to test.
4. Upload the updated framework to run and benchmark with our datasets.

Add a link to the updated framework repo

github.com/user/repo

(a)

Scenario to run your demodulator

Choice of scenarios:
1. Full network deployment: 20 LoRa nodes deployed across different indoor/outdoor settings and variable network transmission rates
2. Variation in Signal-to-Noise Ratio (SNR): 20 LoRa nodes deployed in controlled settings to achieve variable SNR regimes
3. Effect of Interference: 20-node LoRa network deployed in the presence of different types of interferences
4. Effect of Concurrent Transmissions: Synchronized transmissions with increasing number of transmitters colliding
5. Effect of Packet Time Offset: Synchronized transmissions with increasing time offset between 2 node collisions

Choose scenario to run

- Full network deployment
- Variation in Signal-to-Noise Ratio (SNR)
- Effect of Interference
- Effect of Concurrent Transmissions
- Effect of Packet Time Offset

(b)

Select your intended usage *

Add a new custom demodulator

[✓ Run existing demodulators]

Run existing demodulators

Choice of scenarios:
1. Full network deployment: 20 LoRa nodes deployed across different indoor/outdoor settings and variable network transmission rates
2. Variation in Signal-to-Noise Ratio (SNR): 20 LoRa nodes deployed in controlled settings to achieve variable SNR regimes
3. Effect of Interference: 20-node LoRa network deployed in the presence of different types of interferences
4. Effect of Concurrent Transmissions: Synchronized transmissions with increasing number of transmitters colliding
5. Effect of Packet Time Offset: Synchronized transmissions with increasing time offset between 2 node collisions

Choose scenario to run *

Choose
Full network deployment
Variation in Signal-to-Noise Ratio (SNR)
Effect of Interference
Effect of Concurrent Transmissions
Effect of Packet Time Offset
Neve
ndorsed by Google.

(c)

Figure 2: Screenshots from the system interface web page : (a) Flow to add a new demodulator (b) choose a scenario to benchmark (c) Flow to run an existing demodulator for analysis with network setting and SF configuration

NScale : In addition to re-implementing in Python, we parameterized NScale implementation to accept datasets with different LoRa configurations. Various parameters and thresholds for recognizing sync words, clustering packets from the same transmitters together, and choosing the correct demodulation window were generalized to get the optimal demodulation results. The original implementation resulted in missed or repeated symbols when any of the collided packets was split approximately in half by the demodulation window due to ambiguity in length and peak ratios. We implemented an up-chirp correlation-based packet identification, as described in the paper, and aligned the demodulation window appropriately to avoid this corner case.
CoLoRa :We implemented an up-chirp correlation based strategy to process the input file parallelly using Python multiprocessing. This enabled us to identify the start of packets and choose a reception window, ensuring any symbol's split ratio to be between 1/3 and 3 as derived in the paper. We implemented the Akaike Information Criterion based algorithm to detect the onset of the received packet.
CIC: CIC iteratively decodes packets and hence stores the entire data transmission session. At higher data rates, frequent transmissions can lead to very long packet transmissions that can overflow the memory. We overcome this challenge by splitting active data transmissions longer than a threshold into multiple sessions and process them separately.
Std-LoRa Demodulator: For the implementation of the standard LoRa demodulator, we used *rpp0/gr-LoRa* [23], an open source GNURadio block for decoding LoRa packets. *gr-LoRa* has demodulator and decoder integrated as a single block. We split the two into separate blocks and used the demodulator as part of our std-LoRa demodulator implementation. It looks for the strongest peak in each demodulation window and tries to find consecutive occurrences of the same symbol to detect preambles. Once all the preambles are detected and the preamble indices saved, it continues finding the strongest peak in every demodulation window of the detected packet and outputs the corresponding symbols.

3.3 Standard LoRa Decoder Block

Majority of the existing works focus on demodulator performance as a function of the symbols received. A LoRa receiver consists of a demodulator followed by a decoder. The decoder maps received symbols to message. LoRa decoder (and encoder) is responsible for performing forward error correction using Hamming codes, interleaving, whitening, and gray coding to decode symbols to bits and then message. LoRa supports four coding rates ranging from 4/5 having the least redundancy to 4/8 having the highest. This redundancy allows the LoRa signal to endure interferences and correct small errors. While the demodulated symbols provide some understanding of the receiver, the output of the decoder is required to obtain metrics such as throughput, bit error rate, and number of packets successfully received. Additionally, the number of symbol errors that can be corrected by the receiver depends on the coding rate used by the LoRa transmitter.

In order to evaluate the throughput performances of the demodulators, we implemented LoRa decoder in Python. We used an open-source LoRa receiver framework [23] that jointly demodulates and decodes LoRa samples and separated the decoder and demodulator modules. We then implemented the decoder module separately in Python such that the output of any demodulator can be decoded. This allows for a modular implementation of a LoRa receiver pipeline. Our decoder implementation first extracts the symbols corresponding to LoRa header from the demodulator output. It infers the payload CRC and payload length information by reversing the process of Gray encoding, interleaving, shuffling, and Hamming encoding. Finally, these operations are performed on the symbols corresponding to the payload and the final message is displayed as the output. This process is repeated for

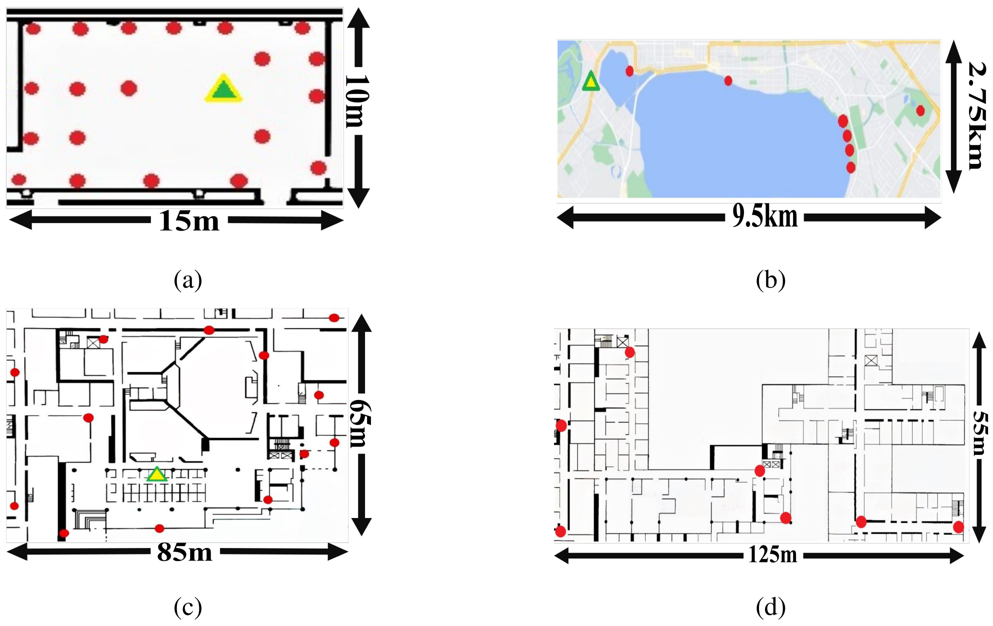


Figure 3: Experimental Setup for LoRa deployments. (a) Indoor LoS, (b) Outdoor, (c) 1st and (d) 2nd Floor Indoor NLoS. Triangle:Base Station, Circles:Transmitters

each demodulator's symbols and the final output is used to calculate the metrics. We validate this Python implementation of the standard LoRa decoder in Section §4. As shown in Fig. 13 in Appendix C.1, the implemented decoder is independent of the demodulators and thus can be integrated with any other demodulator in the pipeline. This openly accessible implementation we have made available, can be used by other researchers to evaluate their demodulator on end-user metrics.

3.4 Metrics

The final module of *OpenLoRa* is the set of metrics that evaluate the end-to-end-performance. This module takes in the demodulated symbols and decoded bits and outputs the calculated metrics. We use the following definitions for the metrics exposed by our framework:

1. Symbol Error Rate (SER): SER is calculated as the ratio of number of incorrect symbols to the total number of transmitted symbols. This metric evaluates the efficiency of demodulator algorithms. With prior knowledge of the transmitted symbols, a one-to-one comparison is used to determine the number of incorrect symbols per packet.

2. Packet Reception Rate (PRR): PRR is calculated as the ratio of number of correct packets received to the total number of transmitted packets. A packet is considered correct if and only if the received message (after error correction) is equal to the transmitted message. Thus, PRR is determined from the output of the decoder.

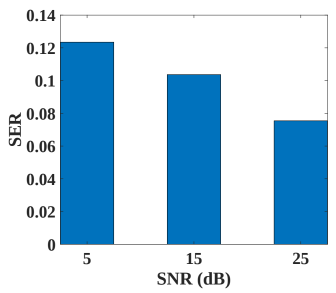
3. Throughput: This is the one of the most important metrics from an end-to-end workflow perspective. Throughput is defined as the number of correct bits received per second. Towards calculating throughput, we only consider correct packets i.e.,packets where all the received bits are correct.

Calculating metrics from the decoded bits provides a holistic evaluation of the receiver performance of the demodulator algorithms, which we believe is critical in practical LoRa deployments. We will use the metrics used in the papers being validated first, in order to cross-validate our implementation in Section §4. Then, we will present our evaluations using the end-to-end metrics of Packet Reception Rate and Throughput obtained from the output of decoder in Section §5 under varying settings, configurations, and scenarios.

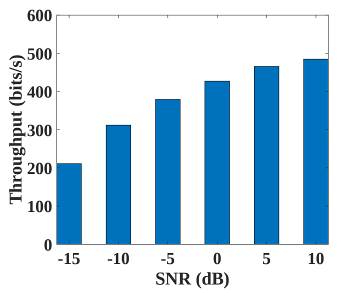
4 Cross Validation of the Demodulators

In order to validate the fidelity of our framework and implementation of the various demodulators, we recreate a representative result from each of the four papers considered. As mentioned in Section §2, each existing work proposes unique techniques to resolve multi-packet collisions. Existing works compare their performance against standard LoRa and a few other existing state-of-the-art. However, each one of them uses a dataset that has been captured in different experimental scenarios, using different configurations for Spreading Factor, Bandwidth, duty cycle and concurrency.

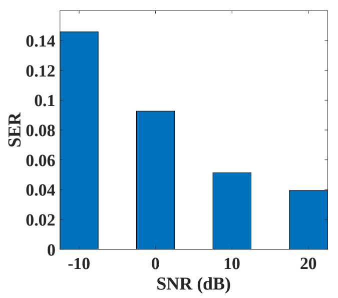
In this section, we recreate the experimental setup as discussed in the original papers to the best of our knowledge and



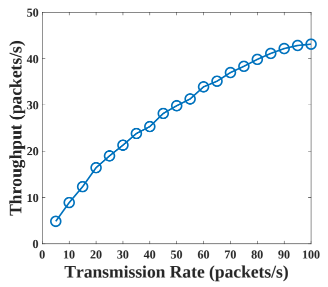
(a)



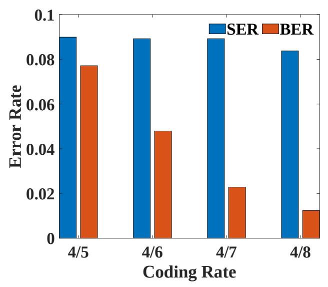
(b)



(c)



(d)



(e)

Figure 4: Cross-validation of results from original papers : (a) FTrack: SER vs SNR (b) CoLoRa: Network Throughput vs SNR (c) NScale: SER vs SNR (d) CIC: Throughput vs Aggregate rate
(e) Cross-validation of Std-LoRa Decoder : SER and BER for varying coding rates of a single transmitter

report the same result metrics. The goal of this exercise is two-fold : 1) To validate our implementation by recreating the original reported results in each of the papers considered 2) To provide a module in our framework for future works to recreate the existing works and their results.

We have selected the following results to recreate :

1. FTrack's Fig. 14 : SER vs SNR
2. CoLoRa's Fig. 12 : Throughput vs SNR
3. NScale's Fig. 11a : SER vs SER
4. CIC's Fig. 28 : Throughput vs transmission rate

FTrack: Following FTrack's setup, we created the two-node collisions initiated by beacon packets. Upon listening to the beacon packets, the two LoRa nodes send packets with a random delay within a packet duration ensuring collisions. Each node transmitted packets of fixed length with SF8 and BW 250kHz. To achieve the SNR ranges of low (<5dB), medium (5 – 20 dB) and high (>20dB) as mentioned in the paper, we installed nodes at appropriate distances to achieve SNRs of 5, 15 and 25dB respectively. Fig. 4 (a) shows that the SER decreases with increasing SNR, implying better collision resolution at high SNRs by FTrack. SER of ≈ 0.1 is in complete agreement with the results presented in the original work.

CoLoRa: We design the experiment with a 20-node architecture that closely follows the description in the original paper. Each node transmitted SF10, BW 250kHz packets at a fixed rate of 1 packet per second. As mentioned in the paper, we captured data for high SNR packets and then added Additive White Gaussian Noise (AWGN) on the captured data to vary its SNR in a controlled manner. Fig. 4 (b) shows the throughput that we obtain for varying levels of emulated SNR. As the SNR increases from -15dB to 10dB, the network throughput increases from 200 bits/s to over 400 bits/s. This result is in complete agreement with the original work.

NScale: For NScale, we generated beacon-initiated collisions with the responding nodes transmitting packets with SF10 and BW 125kHz, following the experimental setup of NScale. We installed nodes at distances that ensured SNRs of -10, 0, 10, and 20 dB. As shown in Fig. 4 (c), lower SER of 0.04 at 20dB SNR indicates strong collision resolution capability of NScale. SER increases slightly for -10dB SNR to approximately 0.1, close to the results presented in NScale. This trend is similar to the original results presented and verifies that NScale is able to achieve low SER even at -10dB SNR.

CIC: In order to recreate the result in CIC, we used the open source data-set provided by CIC in the GitHub repo instead of designing a new experiment. The results, as shown in Fig. 4

(d) are in complete agreement with actual results presented in the paper. CIC's throughput improves from 5 packets per second to over 40 packets per second as the aggregate rate increases, indicating its ability to resolve collisions.

Std-LoRa Decoder : We also validate the correctness of our decoder implementation, since it forms the basis of throughput in Section §5. To validate the decoder, we designed an experiment to study the impact of Forward Error Correction (FEC) over raw symbols. LoRa offers 4 coding rates : 4/5, 4/6, 4/7, 4/8 where 4/8 implies twice as many redundant bits as data bits. Therefore, higher coding rates lead to better reliability and resilience to bit errors, but lowers effective data-rate. We placed a LoRa transmitter in a NLoS setting from the receiver to ensure low SNR of approximately -15dB. We transmit SF8 packets with 250 kHz BW and vary the coding rate of the transmitter for the same message. Fig 4 (e) shows SER and BER for varying coding rates. We can observe that the SER remains almost constant, since SER is unaffected by the coding rate and only depends on the SNR. However as the coding rate varies from 4/5 to 4/8, redundancy increases and as expected, the decoder is able to correct more errors such that the Bit Error Rate decreases from 7.7% to 1.2%. This result establishes the expected decoder operation and validates our implementation of the standard LoRa decoder. This reliability comes at the cost of longer packet time with the coding rate of 4/8 needing 64 ms to transmit the same data as compared to 49 ms for 4/5 coding rate.

To summarize, we recreated one representative result from each of the papers being validated to verify the correctness of our implementation as well as validate the results with the exact same experimental setup described in the respective papers. We show that the results recreated are in complete agreement with that in the original papers. We also validated the decoder block implemented by comparing the SER and BER performance for varying coding rates. In the next section, we design new experiments to further test the throughput performance of each of these demodulators integrated with our LoRa decoder block.

5 Experimental Evaluation

We evaluate the performance of the five demodulators on several metrics and configurations. We aim to answer the following questions in this section through our experiments:

- Do collision resolution techniques improve the overall network throughput in the presence of collisions?
- What is the impact of variations in the SNR for different transmitters on decoding multi-packet collisions?
- What is the impact of LoRa and other non-LoRa narrowband interferers in decoding concurrent transmissions?
- How many concurrent transmissions can be successfully decoded from the cumulative signal?
- How does the time offset between two colliding packets affect the demodulation and throughput performance?

In the rest of this section, we describe the experiments performed and the results observed to answer these questions. Most of the experiments were repeated for two different configurations: SF8, BW 125kHz and SF10, BW 250kHz, to represent low and high air-times respectively. Unless otherwise mentioned, the default configuration is the larger packet airtime with SF10, BW 250kHz, and a coding rate of 4/5.

5.1 Impact of transmission rate on Network Throughput

In this section we evaluate the overall network throughput achieved by various techniques for *Indoor LoS*, *Indoor NLoS* and *Outdoor* data sets for a 20-node network. The transmission rate of each node is varied from 1 packet/s to 5 packets/s, resulting in an aggregate network rate of 20 packets/s to 100 packets/s as the x-axis. Packets were generated with inter-arrival times following an exponential distribution. We collected upto 6000 packets for each iteration of this experiment for different transmission rates, repeated for both SF, BW configurations. Each data point in the figures presented is averaged over all these packets.

Indoor LoS: In this scenario, packets from almost all the nodes are captured at high SNR therefore, all techniques perform at their best. We present the results and analysis of this scenario in Appendix D.1.

Indoor NLoS: At low-SNR indoor NLoS scenario, where 20 nodes are deployed across two floors in an office building, a trend similar to indoor-LoS can be observed with increasing Transmission Rate. As shown in Fig. 5 (a) and (b), at low SNR, the average network throughput is lower than that of higher SNR for most demodulators. All the demodulators achieve their peak throughput at 40 packets/s in case of SF8 transmissions, beyond which it decreases with an increase in the aggregate transmission rate. In case of SF10 transmissions, a much sharper descent in the throughput curve can be seen due to the compounding of the lower SNR with more severe collisions. As noted by the authors themselves in [15], FTrack fails to detect packets at SNRs lower than 10dB. Since majority of the nodes operated near the noise floor (0dB SNR) in this setting, FTrack failed to detect any packets and is not shown in the corresponding Fig. 5.

Long Range Outdoor: To test the long-range capability of LoRa and the demodulators, we deployed LoRa nodes in outdoor environments at distances as far as 8 km from the receiver. Transmissions from these devices were received at SNRs as low as -15dBm. At this low SNR, even schemes that are specifically designed for low-SNR such as NScale were unable to detect any packets. While only CIC was able to

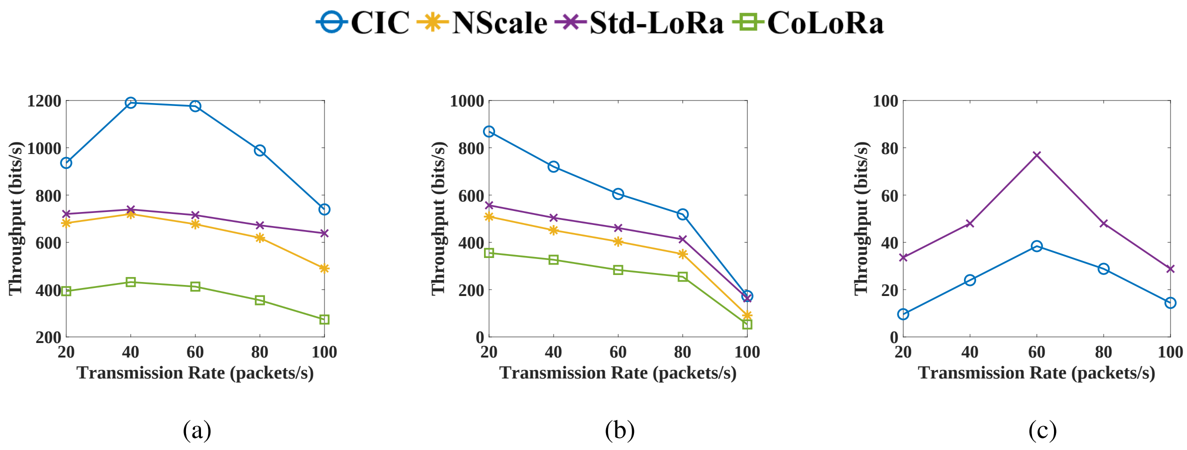


Figure 5: Throughput of a 20-node indoor NLoS network with increasing aggregate transmission rates
(a) SF8, 125kHz bandwidth (b) SF10, 250 kHz bandwidth
(c) Throughput of a 20-node outdoor network with SF10, 250 kHz bandwidth

receive packets, its network throughput is worse than that of standard LoRa as seen in Fig. 5 (c).

Summary: In summary we arrive at the following conclusions : on an average, the network throughput is higher for SF8/125kHz than that of SF10/250kHz for every demodulator. This is because, for the same message, SF10/250kHz packets have twice the duration of SF8/125kHz, which leads to a higher probability of collisions. As the aggregate rate increases, the network throughput peaks due to higher traffic. Further increments lead to an increased number of collisions, thus reducing network throughput. Extreme combination of configuration parameters: NLoS, SF10, 100 packets/s (Fig. 5 (b)), leads to comparable throughput for all demodulators, with excessive collisions nullifying any gains from the elaborate demodulation techniques as compared to the simplistic Std-LoRa. CIC demonstrates significant throughput gains over Std-LoRa in both high (>10dB) and low SNR (around 0dB) settings, followed by FTrack (in high SNR scenario) and NScale. CoLoRa despite performing better on the metrics of SER and BER, falls short of matching Std-LoRa's throughput. FTrack fails to demodulate any packets in the low SNR settings. Most techniques perform poorly at extremely low SNRs (around -15dB) as their preamble detection fails and is not as robust as Std-LoRa. Based on our experiments we believe that further study and *novel techniques for packet detection and collision resolution, especially in low-SNR regimes is needed.*

5.2 Impact of Signal to Noise Ratio (SNR) on Network Throughput

We note from the network throughput in the long-range outdoor settings above that none of the existing techniques perform better than Std-LoRa at extremely low SNR scenarios. To study the impact of SNR on the throughput performance of each demodulator, we design a new controlled experiment. In the Indoor setup, we deploy 20 LoRa nodes and accurately control their transmit power and physical placement such that all the nodes have comparable SNR that falls under one of the following categories. We repeat the experiment such that all the nodes are in High, Medium, Low, and Extremely-low SNR categories, as defined below :

- **High** SNR : >10 dB
- **Medium** SNR : 5 to 10 dB
- **Low** SNR : -5 to 5 dB
- **Extremely Low** SNR : <-5 dB

Each transmitter was configured at SF10, BW 250kHz; we collected upto 3000 colliding packets for each combination of SNR and transmission rate. Fig. 6 shows the throughput of the network as a function of decreasing SNR regime defined above, repeated for aggregate transmission rates of 20, 60 and 100 packets/s respectively. Due to the controlled setting needed for this experiment, it was not performed in the outdoor setting.

Summary: The results corroborate the earlier observations in indoor and outdoor experiments. CIC, NScale, and FTrack outperform Std-LoRa in high SNR, low traffic scenarios (Fig. 6 (a)) because of lesser collisions. However, as the SNR drops below noise floor, the throughput gains delivered by these demodulators decline sharply. FTrack fails to detect packets in lower SNR settings, whereas the throughput for other demodulators drop as we move towards lower SNR and higher transmission rates (Figs. 6 (b) and (c)). As stated in NScale, its performance is comparable to FTrack at high and medium SNR, and outperforms FTrack at lower SNRs. CIC improves the most over Std-LoRa in most of the scenarios, NScale also performs well in medium and low SNR regimes, not

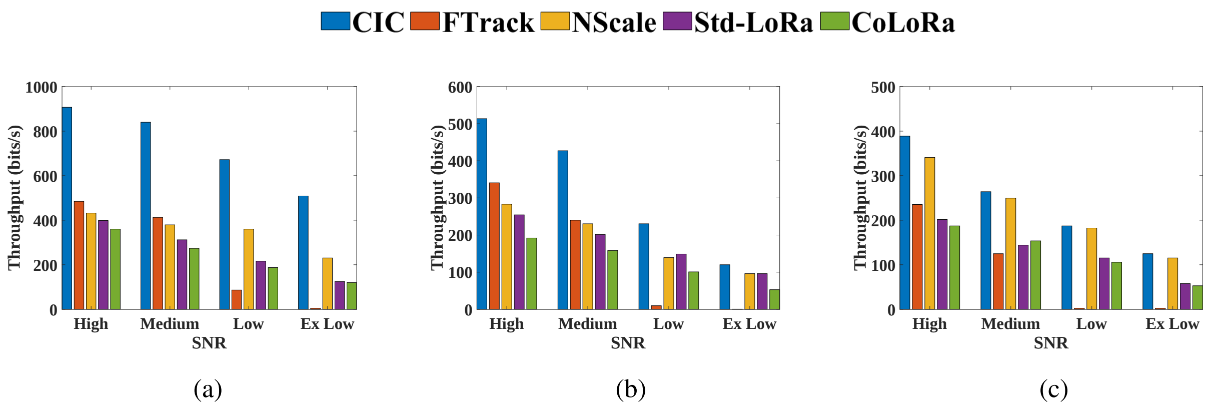


Figure 6: Throughput of a 20-node SF10 network with varying SNR and aggregate transmission rates
(a) 20 packets/s (b) 60 packets/s (c) 100 packets/s

suffering excessive drop in throughput with lower SNR. At extremely-low SNR settings, all these techniques fail to demodulate most of the packets, which is further worsened at higher traffic rate. Similar to our earlier observations, the SER analysis of CoLoRa indicates that even though it has low SER, at low and extremely low SNRs, the overall throughput is worse than that of Std-LoRa. From these experiments, *we conclude that further research is required to achieve significant throughput gains over standard LoRa in extremely-low SNR and/or dense deployments, that are typical scenarios for LoRa applications.*

5.3 Impact of Interference on Network Throughput

In addition to underutilized network capacity, co-existence of multiple LoRa networks as well as across technologies was identified as a bottleneck for scalability [11, 13]. In this experiment, we evaluate the throughput performance of the five demodulators in the presence of LoRa transmissions from neighboring LoRa networks as well as non-LoRa, Gaussian Frequency Shift Keying (GFSK) narrow-band transmissions, representing other LPWANs operating in the same band. To the best of our knowledge, this is the first work to design an experiment and evaluate the impact of interference on the demodulator performance.

The primary LoRa network in this setup consists of 20 nodes, configured at SF10, BW 250kHz setting. We study the impact of the following three interfering nodes, each placed within few meters from the receiver; the interfering signal transmits at a duty cycle of 50% with an SNR of approximately 7 dB at the receiver.

i SF8, BW 125kHz LoRa node sending 93 ms packets

ii SF12, BW 500kHz LoRa node sending 290 ms packets

iii GFSK node sending 50 ms packets

Summary: Fig 7 shows the network throughput under these three types of interference. *Ambient* shows the results with no added interference. Due to the orthogonality of CSS, we expected no impact from SF8, BW 125kHz node. However, it does have a very minor impact on the performance of SF10 nodes for most demodulators. Although LoRa signals with different SFs are typically assumed to be perfectly orthogonal, this result attests to the Quasi-Orthogonality of different SFs (discussed in [24]). This quasi-orthogonality leads to residual interference even after dechirping which raises the noise floor and consequently reduces the SINR (Signal to Interference + Noise Ratio). GFSK interference has a minimal impact on the throughput performance, with a minor drop attributed to SINR. This showcases the inherent robustness of CSS to other narrow-band interference. SF12, BW 500kHz interference however notably reduces the throughput of all demodulator algorithms by almost 50%. This is primarily due to the longer packet duration of SF12 which has higher probability of interfering with complete SF10 packets whereas shorter duration of SF8 packets are less likely to interfere complete SF10 packets. *Thus, higher SFs have more impact on lower SF transmissions due to the increased probability of collision.* This is critical to notice in practical deployments where multiple LoRa networks, each operating at a different SF would co-exist. Additionally, SF-based MAC protocols have been proposed as a way to improve LoRa's scalability. *Thus, we believe that collision resolution techniques that consider the impact of interference from other LoRa networks with multiple SFs remains to be developed.*

5.4 Impact of concurrent transmissions on Packet Reception Rate

In the experiments so far, the nodes transmitted packets randomly at a predetermined rate. As the rate increased, packet collisions increased, affecting network throughput. To understand the efficacy and limitations of each algorithm in decod-

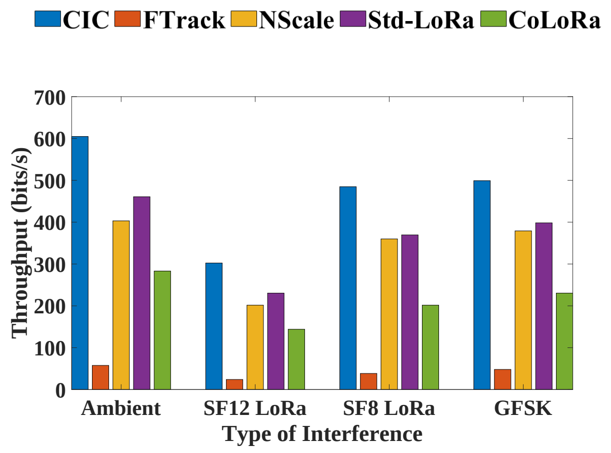


Figure 7: Throughput of a 20-node SF10 network in the presence of various interference signals

ing concurrent collisions, we performed controlled collision experiments. We synchronized all the nodes with a beacon packet: on receiving a beacon, each sender transmits a single packet with a random delay between 0 and packet duration. We define concurrency to be the number of colliding packets within one packet duration i.e.,each packet partially overlaps with every other packet.

We evaluate the number of colliding packets that can be resolved by each demodulator using the metric Packet Reception Rate (PRR). We measure PRR as we increase the number of concurrent packets from 2 to 12 in indoor LoS setup. We define PRR as the ratio of fully correct packets decoded at the receiver to the total number of packets transmitted from the transmitters. We collect upto 180 such colliding packets for each iteration of the experiment and present the metrics averaged over this dataset. Figs. 8 (a) and (b) show the PRR for SF8/125kHz and SF10/250kHz.

As the number of concurrent transmissions increases, PRR decreases for every demodulator. As expected, Std-LoRa has a sharp decrease in PRR since it demodulates at most one packet at a given time. We notice that irrespective of the number of nodes, Std-LoRa aligns with the strongest signal and successfully demodulates symbols corresponding to that packet. CoLoRa intentionally misaligns its demodulation window to detect more packets. Although CoLoRa detects most of the packets, it fails to demodulate when the majority of a packet overlaps with other packets. This is because of errors in finding peak ratios accurately. NScale builds on CoLoRa and improves demodulation; PRR using NScale is higher than that of CoLoRa. However, it suffers from not recognizing frequency bins of the detected peaks in presence of collisions, often leading to small offsets in the demodulated symbols. FTrack's use of time and frequency domain features to create frequency tracks and separate out collisions enables it to infer more number of colliding packets. FTrack is able to correctly output 3% to 5% more packets as compared to Std-LoRa. CIC, which improves on FTrack has the highest PRR. CIC's use of interference cancellation and spectral intersection features to demodulate enables it to demodulate most number of colliding packets. Beyond 8 concurrent collisions, PRR of all the approaches is below 0.2.

Summary: We observe a sharp decrease in PRR with increasing concurrency for all techniques because increasing concurrency reduces SINR. In this controlled collision setting, we see that SF10 transmissions result in a higher PRR, in contrast to the observations for scenarios with random collisions. This is because higher packet air-time with SF10 works in favor of demodulators here as there is more leeway in how closely in time the transmitters can collide. As the network scale of LoRa deployments increases, we expect concurrent collisions to occur with higher probability. We study the impact of the time between two colliding transmissions in more detail in the following section. *We believe that there is still room to improve in decoding collisions with more than two concurrent transmissions.*

5.5 Impact of Packet Time Offset (PTO) on Packet Reception Rate

It is evident from the concurrent transmission experiments that as the number of concurrent transmissions increases, the packet reception rate and hence network throughput decreases. Prior work [15] has also observed that the relative time (and frequency) offsets between two colliding packets plays a critical role in the throughput performance. Therefore, the impact of concurrent transmissions on throughput is a function of the time offset between the colliding packets. To understand the impact of time offsets, we design the following experiment: two LoRa nodes are connected to an Arduino microcontroller (MCU). The MCU, using a hardware interrupt, triggers the LoRa nodes such that the difference in the start of their packet transmission is configurable, thus controlling time offsets. Both the nodes transmit the same data. We repeat the experiment 20 times for each offset value and present the PRR metric averaged over the whole data.

Summary: Figs 9 (a) and (b) show the PRR as a function of increasing time offsets between two LoRa nodes. NScale and CoLoRa depend heavily on packet offsets and peak ratios to demodulate symbols and to group symbols from each transmitter together. As the packet time offset increases, their PRR performance improves as accuracy in peak estimation and peak ratio will increase. So, these show significantly better performance as the packet offset time increases, with NScale improving its PRR from 0 to 0.9 when changing collision time offset from 5% to 30% for SF8 packets. Similarly, CoLoRa shows a notable improvement in PRR over the same range, going from 0 to 0.4. Std-LoRa is unaffected by the time offsets since it always decodes the strongest signal. CIC and FTrack demodulate packets iteratively and use time and

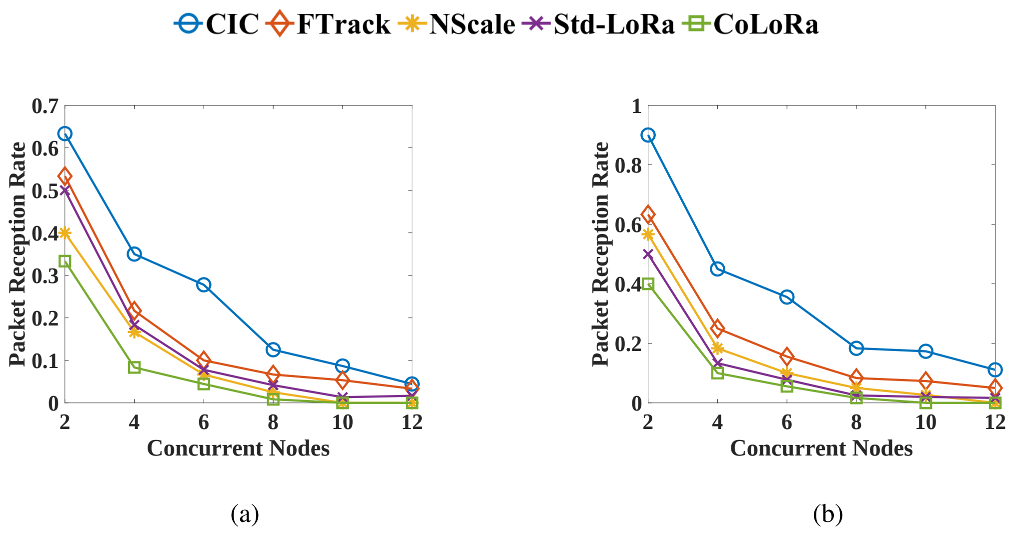


Figure 8: Reception rate of fully correct packets with increasing concurrent transmissions
(a) SF8, 125kHz bandwidth (b) SF10, 250 kHz bandwidth

frequency information to separate collisions. The impact of time offsets on FTrack and CIC is also therefore negligible. We conclude that although most demodulators focus on improving network throughput by resolving collisions, some are better suited for collisions with a higher overlap in time than others. We observe that demodulation techniques that utilize both time and frequency resolutions are more resilient to PTO. *Therefore, more study is needed on improving time-frequency resolution when multiple concurrent nodes collide within short time offsets.*

6 Discussions and Limitations

Our results shows that packet decobability and hence network throughput differs significantly under varying network conditions. Therefore, rigorous evaluation of the existing and future LoRa demodulators over a wide variety of network conditions, as described in Section 5. We have made the datasets of the various network conditions and scenarios evaluated publicly available. Although this is an extensive set of scenarios, it certainly is not an exhaustive list. We strongly encourage the community to share new scenarios and datasets.

In *OpenLoRa*, we focused on network throughput comparisons across multiple demodulators. However, we did not compare the computational complexity of existing techniques. Since we relied on accurate recreation of existing works, optimizing each technique's implementation was not the focus. In order to evaluate the practical usability of a demodulator, computational complexity is critical. For example, Std-LoRa and CoLoRa rely on dechirping followed by FFT for packet detection and demodulation and cost the least in terms of number of computations, but also have the lowest throughput improvements, i.e., the computational complexity for both is $Nlog(N)$ where N is the number of samples per symbol and is typically the size of the FFT window as well. On the other hand, FTrack, NScale, and CIC use auto-correlation to detect the start of packets and therefore their packet detection cost these schemes N^2 computations. FTrack computes the spectrogram for the whole received buffer using a sliding window and tracks the frequencies in each window of the spectrogram; therefore, its demodulation block has $N^2log(N)$ complexity per window. CIC computes spectrogram for a fixed number of sliding windows regardless of SF as opposed to sliding over whole demodulation window and therefore has computation cost of $cNlog(N)$ where c is a constant that depends on the number of fixed sliding windows. NScale performs multiple dechirpings followed by FFT to translate timing offsets to frequency features and have a computation cost of $kNlog(N)$ where k is the number of multiple dechirpings. The computation cost for CIC and NScale is therefore lower than that of FTrack and more than standard LoRa's and CoLoRa's computation cost. Thus, further evaluations on the network throughput improvements along with their computational complexity is needed to idenifty the practical challenges in deploying the demodulators.

In this work, we focused on novel demodulators that receive from commercially available LoRa transmitters. More recent works such as CurvingLoRa [25] and NetScatter [26] have proposed changes to the transmitter to improve resilience to packet collisions, communication range, and network throughput. Although our framework cannot be used for non-standard LoRa transmitters, the new techniques can still use our experimental scenarios to test their performance in different network conditions and compare against standard LoRa.

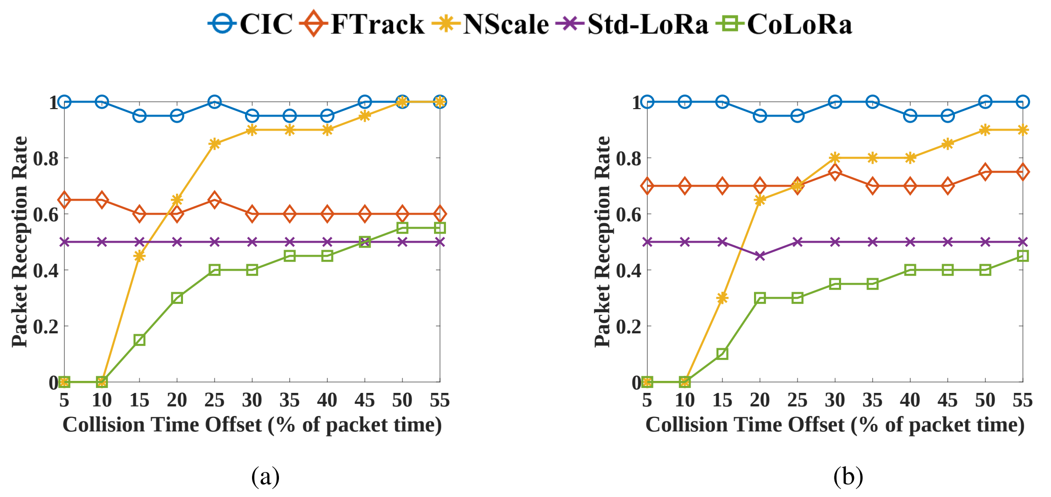


Figure 9: Reception rate of fully correct packets with increasing collision time offset (a) SF8, 125kHz bandwidth (b) SF10, 250 kHz bandwidth

7 Conclusions

In this work, we implement and validate four state-of-the-art collision resolution techniques for LoRa on a variety of system configurations and scenarios. We developed *OpenLoRa*, a Python framework that provides a uniform platform to evaluate existing works over the same datasets and metrics. We also design a standard LoRa decoder in order to study the end-to-end performance. This platform will allow future researchers to plug-in their demodulator and benchmark against existing works. We observe that metrics such as network throughput are more meaningful for practical deployments. We study the impact of interference and variations in SNR on the network throughput performance. We perform a wide range of experiments to emulate practical deployment settings, showcasing the strengths and challenges of existing LoRa demodulators. Our evaluations show that there are open challenges in the low-SNR, long-range regime, with more room for innovations in LoRa packet collision resolution.

8 Acknowledgements

We would like to thank our shepherd Aaron Schulman and the anonymous reviews for the valuable comments and helping us improve the paper. The authors are partially supported through the following NSF grants : CCSS-2034415, CNS-2142978, CNS-2213688, CNS-1838733, CNS-2112562, CNS-1719336, CNS-1647152, CNS-1629833, CNS-2107060, and CNS-2003129 and an award from the US Department of Commerce with award number 70NANB21H043.

References

[1] E. Asimakopoulou and N. Bessis. Buildings and crowds: Forming smart cities for more effective disaster management. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 229–234, 2011.

[2] María V Moreno, Miguel A Zamora, and Antonio F Skarmeta. User-centric smart buildings for energy sustainable smart cities. *Transactions on emerging telecommunications technologies*, 25(1):41–55, 2014.

[3] Achim Walter, Robert Finger, Robert Huber, and Nina Buchmann. Opinion: Smart farming is key to developing sustainable agriculture. *Proceedings of the National Academy of Sciences*, 114(24):6148–6150, 2017.

[4] Climate Smart Agriculture. https://www.worldbank.org/en/topic/climate-smart-agriculture.

[5] Fei Tao, Qinglin Qi, Ang Liu, and Andrew Kusiak. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48:157–169, 2018.

[6] Harsha V Madhyastha and Chinedum Okwudire. Remotely controlled manufacturing: A new frontier for systems research. In *Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications*, pages 62–67, 2020.

[7] Smart Animal Production. https://lora-alliance.org/wp-content/uploads/2020/12/THE-FARMING-OF-TOMORROW-IS-ALREADY-HERE-HOW-LoRaWAN%C2%AE-TECHNOLOGY-SUPPORTS-SMART-AGRICULTURE-PRECISE-ANIMAL-pdf.

[8] Precision Agriculture. https://cdn2.hubspot.net/hubfs/2507363/Semtech_Smart_Agriculture_White_Paper.pdf.

[9] Smart Building. https://lora-alliance.org/wp-content/uploads/2020/11/LA_WhitePaper_SmartBuildings_0520_v1.1_1.pdf.

[10] Smart Planet. https://info.semtech.com/hubfs/Semtech-UseCase-EBook-SmartPlanet_locked.pdf?hsCtaTracking=a1c4b6e2-c4a4-4fa6-942f-639ad15a6518%7C769d05d8-4178-4854-8ee6-f34d756fc141.

[11] LoRaWAN capacity trial. https://info.semtech.com/hubfs/machineQ_LoRaWAN_Capacity_Trial-2.pdf?hsLang=en-us.

[12] LoRaWAN capacity trial. https://cdn2.hubspot.net/hubfs/2507363/Semtech_Network_Capacity_White_Paper.pdf.

[13] Branden Ghena, Joshua Adkins, Longfei Shangguan, Kyle Jamieson, Philip Levis, and Prabal Dutta. Challenge: Unlicensed lpwans are not yet the path to ubiquitous connectivity. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2019.

[14] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yağan. Empowering low-power wide area networks in urban settings. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 309–321. Association for Computing Machinery, 2017.

[15] Xianjin Xia, Yuanqing Zheng, and Tao Gu. Ftrack: Parallel decoding for lora transmissions. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 192–204, 2019.

[16] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in lpwans. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 234–246, 2020.

[17] Shuai Tong, Zhenqiang Xu, and Jiliang Wang. Colora: Enabling multi-packet reception in lora. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2303–2311. IEEE, 2020.

[18] Xiong Wang, Linghe Kong, Liang He, and Guihai Chen. mlora: A multi-packet reception protocol in lora networks. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[19] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. Concurrent interference cancellation : Decoding multi-packet collisions in lora. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '21. Association for Computing Machinery, 2021.

[20] Zhenqiang Xu, Pengjin Xie, and Jiliang Wang. Pyramid: Real-time lora collision decoding with peak tracking. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021.

[21] Qian Chen and Jiliang Wang. Aligntrack: Push the limit of lora collision decoding. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2021.

[22] Adafruit Feather M0 with RFM95 LoRa Radio. https://www.adafruit.com/product/3178.

[23] RPPO. https://github.com/rpp0/gr-lora.

[24] Yujun Hou, Zujun Liu, and Dechun Sun. A novel mac protocol exploiting concurrent transmissions for massive lora connectivity. *Journal of Communications and Networks*, 22(2):108–117, 2020.

[25] Chenning Li, Xiuzhen Guo, Longfei Shangguan, Zhichao Cao, and Kyle Jamieson. CurvingLoRa to boost LoRa network throughput via concurrent transmission. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 879–895, Renton, WA, April 2022. USENIX Association.

[26] Mehrdad Hessar, Ali Najafi, and Shyamnath Gollakota. NetScatter: Enabling Large-Scale backscatter networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 271–284, Boston, MA, February 2019. USENIX Association.

[27] Chenning Li, Hanqing Guo, Shuai Tong, Xiao Zeng, Zhichao Cao, Mi Zhang, Qiben Yan, Li Xiao, Jiliang Wang, and Yunhao Liu. Nelora: Towards ultra-low snr lora communication with neural-enhanced demodulation. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, SenSys '21, page 56–68, New York, NY, USA, 2021. Association for Computing Machinery.

A Appendix: LoRa Primer

LoRa Modulation. In LoRa, data is modulated using a Chirp Spread Spectrum (CSS) scheme. In CSS, symbols are chirp signals whose instantaneous frequency increases linearly with time as shown in Figure 10 (a). A base chirp starts from a frequency of $\frac{-BW}{2}$ and increases linearly to $\frac{BW}{2}$ over a symbol duration of T_s where BW is the bandwidth of transmission and T_s can be defined as $T_s = \frac{2^{SF}}{BW}$. $SF \in \{7,8,9,10,11,12\}$ defines a packet's Spreading Factor, a value that dictates the data-rate, resistance to interference and range of transmission.

Every symbol $S(t, f_{sym})$ is derived by cyclically shifting a base chirp $C(t)$, as shown in Equation 1. For example in Figure 10 (b) and (c), $S(t, f_1)$ and $S(t, f_2)$ are obtained by introducing a frequency offset of f_1 and f_2 to the base chirp. The data to be transmitted modulated these starting frequencies viz., f_1 and f_2.

$$C(t) = e^{j2\pi(0.5\frac{BW^2}{2^{SF}}t - \frac{BW}{2})t}, 0 \le t \le T_s \tag{1}$$

$$S(t, f_{sym}) = C(t) \cdot e^{j2\pi f_{sym}t} \tag{2}$$

LoRa Demodulation. To demodulate a symbol, a LoRa receiver aligns and multiplies a down-chirp $C^{-1}(t)$ (the complex conjugate of $C(t)$) with the received symbol $S(t, f_{sym})$ (Equation 3). Dechirping transforms the chirp signal to a sinusoid with a constant frequency equal to the start frequency f_{sym}. This frequency is located by finding the peak in the FFT of the dechirped signal. The operation of dechirping followed by FFT concentrates the symbol's energy into a single frequency, thus providing the spread spectrum gain necessary to decode symbols in sub-noise conditions.

$$C^{-1}(t) \cdot S(t, f_{sym}) = e^{j2\pi f_{sym}t} \tag{3}$$

Since dechirping requires the downchirp to align with the received symbol, a LoRa receiver determines the onset of a new packet by searching for its preamble and uses it to identify the symbol boundaries of symbols. LoRa preamble comprises of a sequence of $N = 6$ to 65535 consecutive $C(t)$ symbols, followed by two SYNC symbols $S(t, s_1), S(t, s_2)$ ($s_1 \neq 0$, $s_2 = s_1 + 8$) and 2.25 down-chirps $C^{-1}(t)$. To detect a new packet, the receiver continuously de-chirps and performs an FFT until it finds N consecutive peaks with the same frequency. The SYNC words and down-chirps then help locate the symbol boundaries. In CSS, time offsets are equivalent to frequency offsets. For example, as shown in Figure 11 time shifting a chirp symbol by τ will introduce an equivalent frequency offset in the starting frequency. Therefore, frequency offsets in LoRa can easily be compensated by identifying the equivalent time shifts during preamble detection.

Effect of collisions on demodulation. Standard LoRa is incapable of demodulating data symbols in case if multiple packets collide. Should multiple LoRa packets arrive simultaneously at the receiver, there will be multiple f_{sym} values to detect for any given symbol window, making it difficult for standard LoRa to choose one. Standard LoRa assumes that the maximum peak in the FFT always corresponds to the data value of the packet of interest. Whereas, in case of packet collision, multiple peaks from interfering packets show up in the FFT as shown in Figure 12 and the assumption of maximum peak's link to the packet of interest is not guaranteed anymore since height of interfering peaks may surpass the height of true peak.

B Appendix: LoRa Demodulators Validated

B.1 FTrack [15]

FTrack [15] relies on Short Time Fourier Transform (STFT) to obtain time and frequency features. Applying STFT to the LoRa symbols would result in frequency tracks that increases linearly with time. An appropriate STFT window size that offers good frequency resolution to identify the frequency and good time resolution to follow the progression of a chirp is challenging to determine. FTrack proposes to apply STFT on the dechirped LoRa symbol to leverage the spread spectrum gain as well as to remove the linear change in frequency with time. Dechirping the received buffer results in a sinusoid whose frequency remains constant throughout the symbol duration. This allows FTrack to choose a window size of a symbol length that yields a good frequency resolution (of upto 1 bin) if perfectly aligned with the symbol boundaries. Dechirping allows STFT to have the least possible frequency variation with time (single frequency over a symbol duration) and therefore yields the best possible frequency resolution. It yields a constant frequency track over a symbol duration, rendering it simpler to track the frequency of a packet of interest. FTrack, thus, employs dechirping followed by STFT to extract the longest frequency tracks to detect preamble as well as data symbols. Typical LoRa preambles consists of 8 base upchirps, that promise a constant frequency track for a duration of 8 symbols in the final spectrum. FTrack extracts symbol edges from preambles and uses this time information to detect the symbol boundaries of payload. FTrack builds on the observation that all the interfering packets are misaligned in time and hence their symbol boundaries are misaligned in time. FTrack detects the preambles of all colliding packets and leverages the time offset between colliding packets to differentiate transmitters. The receiver aligns itself with the boundaries of a packet of interest : once aligned, it observes the frequency tracks of current packet as well as that of the interfering tracks. The frequency track of the packet of interest will be continuous in the given window whereas all the interfering tracks will change abruptly. Therefore, after detecting all the LoRa packets in a received buffer, FTrack iteratively demodulates each packet. While demodulating a specific packet, FTrack cancels out interfering symbols by tracking the frequency continuity. At the end of this itera-

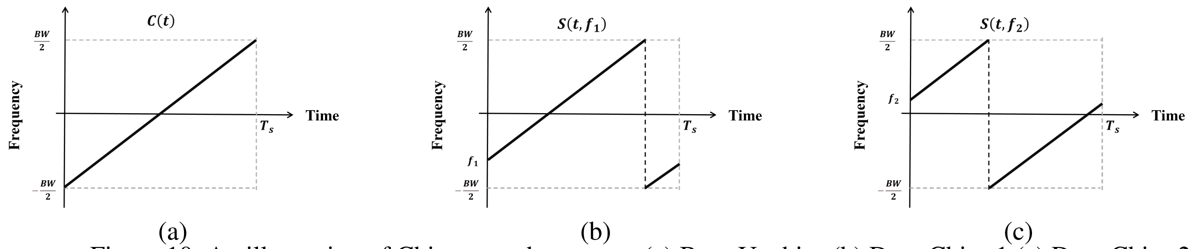


Figure 10: An illustration of Chirp spread spectrum (a) Base Upchirp (b) Data-Chirp 1 (c) Data-Chirp 2

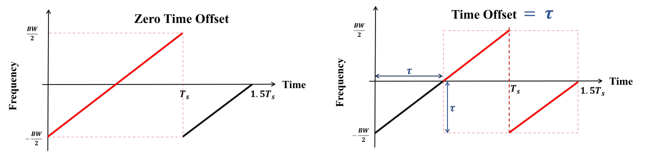


Figure 11: Time offsets in LoRa symbols translates to Frequency offsets

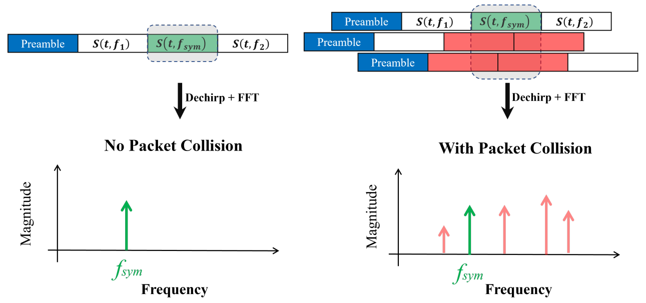


Figure 12: Symbol spectrum with and without collisions

tive process, FTrack receiver detects and demodulates data symbols from multiple transmitters that collided with each other. FTrack's performance suffers in low-SNR conditions. At SNRs below 5 dB, energy of the frequency tracks corresponding to preamble and data symbols are not high enough and hence are buried in the noise floor and is not decoded. Thus, it fails to detect and/or demodulate LoRa packets at SNRs below 5 dB.

B.2 CoLoRa [17]

CoLoRa [17] proposes a novel technique to translate time offsets to frequency features, in turn using that to resolve packet collisions at low-SNR regimes. CoLoRa starts with a misaligned window size of one symbol length. It determines the presence of interference based on the number of peaks appearing in the FFT obtained after dechirping; since multiple peaks imply packet collisions as discussed in Appendix A. Once collisions are confirmed, CoLoRa proposes an interleaved window selection strategy. It chooses a misaligned window such that no chirp is covered fully by the window i.e.,each chirp is segmented and thus falls into two consecutive windows such that the normalized FFT peak is bounded within [1/3,3] in each window. It then jumps the window over received buffer and performs dechirping followed by FFT at each point.The resulting spectrum contains peaks whose height is proportional to the segment of chirp appearing in the current window. CoLoRa observes that when a chirp is split into 2 windows, the frequency at which the peaks appear in the FFT remains the same across the 2 windows. However, the energy and hence the height of the FFT peak at the corresponding frequency in each window is proportional to the duration of the chirp segment within that window.

CoLoRa proposes *Peak Ratio*, which is defined as the ratio of peak heights of a chirp appearing in two consecutive windows; it captures time misalignment through frequency features. It proves that the peak ratio is identical for all chirps of the same packet since the in-window distribution of chirps is identical for all the symbols of the same packet. Additionally, peak ratios differ across packets since the in-window distribution of chirps is different due to misalignment of interfering packets. Since CoLoRa relies on accurate estimation of Peak ratio, it proposes an iterative peak recovery algorithm to estimate the heights of strong peaks first and cancel their contribution while estimating the low-SNR peaks. Since the chirps are segmented, wide side lobes appear around peaks that may bury low SNR peaks. CoLoRa uses k-means clustering (where k is the number of packets detected) to classify the packets of different clients. Peaks with identical peak ratios are clustered together, following the observation that the symbols of the same packet have the same peak ratio. Each cluster represents a unique packet, thus decoding multiple packets from the collided signal.

B.3 NScale [16]

While FTrack and CoLoRa focus on decoding multi-packet collisions, their performance improvements over standard LoRa demodulator is noticeable at high SNR. NScale [16] focuses on decoding packet collisions at SNRs as low as -10dB where the relative performance improvement of FTrack and CoLoRa degrades. Similar to CoLoRa, NScale translates the timing offsets to frequency features and further amplifies the time offsets by non-stationary signal scaling. NScale's strength lies in its ability to decode and resolve collisions of LoRa packets below -10 dB SNR. Instead of sliding the window as FTrack does, NScale jumps the window of size that promises maximum frequency resolution i.e.,duration of a symbol. To retain sub-noise decodability, NScale relies on dechirping to accumulate energy at the single frequency. While jumping the window across the received buffer, NScale observe that, for a specific LoRa packet, all the symbols of interest will have same in-window distribution in consecutive windows whereas in-window distribution for interfering symbols will be different. Simply put, the location of symbol edges where the symbols transition to next symbol is same across all the windows of a given packet but across different packets, these symbol edges are different. This essentially stems from the fact that collisions are misaligned in time. Similar to CoLoRa, NScale translates symbol edge offsets to the peak heights.

NScale introduces a novel non-stationary scaled window as opposed to conventional rectangular window of FTrack and CoLoRa. Non-stationary scaling across the windows amplifies the timing misalignment of symbols of interfering packets. The linear amplitude scaled window scales the amplitude of peaks with respect to their in-window distribution and therefore amplifies the misalignment of different packets. This amplification helps estimate the time offsets for very low SNR packets. Consequently, different packets get a unique fingerprint in terms of peak heights while symbols of a specific packet share the same fingerprint. NScale detects the number of packets and their corresponding start and end indices using correlation and then performs k-means clustering to classify symbols based on their fingerprint.

B.4 Concurrent Interference Cancellation [19]

Concurrent Interference Cancellation (CIC) [19], similar to FTrack, leverages time and frequency domain analysis to decode multi-packet collisions. CIC introduces the concept of sub-windows, which are a portion of the demodulation window. It observes that, for a given packet of interest, symbols from the packet appear in all the sub-windows; symbols from interfering packets appear only in a subset of the sub-windows. Therefore, the intersection of FFT of all the sub-windows would result in the symbol of interest. CIC proposes a sub-window selection algorithm that maximizes interference cancellation.

CIC looks for the best sub-window which promises an acceptable time resolution while compromising the least on frequency resolution. It uses packet detection to determine the start of all the colliding packets inorder to select the best set of sub-windows. Unlike standard LoRa, CIC uses downchirp correlation to detect the start of a packet. With prior knowledge of symbol duration, CIC determines symbol boundaries within each of the colliding packet. The sub-windows are chosen such that it contains the most of each interfering symbol, using CIC's knowledge of the symbol boundaries of interfering packets. Spectral intersection of the FFT of the demodulating window and the optimum set of sub-windows selected results in a single FFT peak that corresponds to the symbol of interest. It iteratively demodulates the rest of the packets in the collided signal. CIC also proposes fractional frequency offset to filter out interfering peaks that were not cancelled in the spectral intersection. Additionally, it uses power-filtering to estimate received power of each packet from its preamble and discards symbols which do not qualify a certain power threshold. Finally Spectral Edge Difference, filters interfering peaks further and chooses one peak to be the final demodulated peak.

B.5 Other recent works on LoRa demodulation

In addition to the works presented above, other papers have focused on LoRa demodulator design. The modular design of our proposed framework renders it simple to integrate them. Pyramid [20] is one such work which tries to resolve LoRa collisions by tracking the change in FFT peak heights corresponding to different interfering symbols. AlignTrack [21] tracks and translates time offsets to frequency features, i.e. peak heights, similar to CoLoRa. AlignTrack chooses a window which completely overlaps the packet of interest instead of a misaligned window used by CoLoRa. AlignTrack's complete overlap gives highest peaks in FFT and therefore, has least SNR loss. NeLoRa [27] is another work which tries to push the limit of LoRa's range by using deep-neural-network. Their results show that its ability to decode packets with SNR as low as -30dB.

C Appendix: Implementation Overview

C.1 Python Implementation

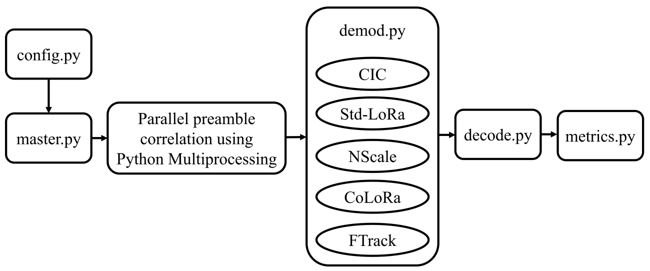


Figure 13: Flowchart of the Python implementation of the proposed framework

C.2 Code Organization

The implementation has been organized into the following major Python modules:

- *config.py*: Configures demodulation parameters such as SF, BW, sampling rate to pass to the demodulators as

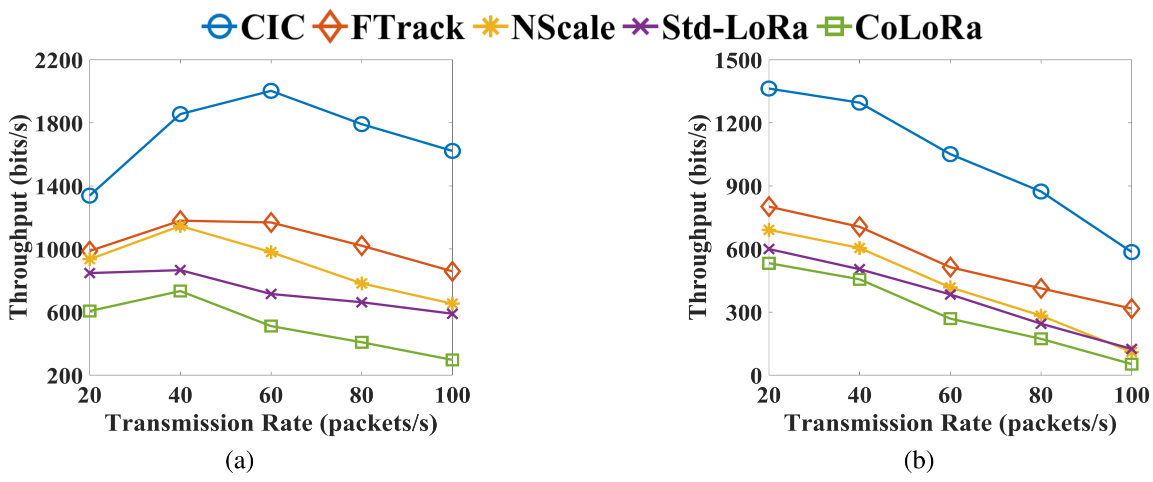


Figure 14: Throughput of a 20-node Indoor LoS network with increasing aggregate transmission rates
(a) SF8, 125kHz bandwidth (b) SF10, 250 kHz bandwidth

well as the transmitted symbols and bits to compare against and evaluate the desired metrics.

- *master.py*: Performs the highest level tasks and interfaces with other modules. Reads in the input file, calls each demodulator module to get the symbols and gets calculated metrics after decoding. Also implements parallel upchirp-based preamble detection using Python multiprocessing capabilities.

- *demod.py*: Imports each demodulator implementation and interfaces with each block to pass the data and parameters in appropriate format. Returns demodulated symbols to the *master* block.

- *decode.py*: Implements the standard LoRa decoder as described in the following subsection. Returns decoded bits to the *master* block.

- *metrics.py*: Takes in the demodulated symbols, decoded bits, and the configuration information to calculate all the relevant metrics and saves then in the specified output file.

D Appendix: Additional Results

D.1 Impact of transmission rate on Network Throughput

Indoor LoS: Figs. 14 (a) and (b) depict the average network throughput as a function of aggregate transmission rate in indoor LoS setting. Network throughput is calculated as the sum of the bits of successfully decoded packets per unit time. In case of SF8 transmissions, the network throughput increases with increasing aggregate rate upto the rate of 40 packets/s for most demodulators, due to increase in traffic. However, beyond a threshold, packet collisions are too high for a demodulator to resolve, leading to a drop in throughput. CIC achieves its peak throughput at 60 packets/s because of its power filtering and down-chirp based preamble detection features. FTrack and NScale perform considerably well in this high SNR scenario giving significant gains over Std-LoRa. Even though Std-LoRa is unable to demodulate concurrent transmissions, it latches on to the strongest signal due to capture effect and often correctly demodulates the packet from the strongest transmission. This results is throughput numbers of Std-LoRa being comparable to other demodulators at higher transmission rates when they suffer from abundance of collisions. Another interesting observation is CoLoRa's throughput being slightly lesser than that of Std-LoRa despite having a lower Symbol Error Rate (SER) and Bit Error Rate (BER). Our analysis indicates that the reason is its erroneous identification of peak frequencies and calculation of peak ratios in the presence of high amount of collisions. CoLoRa detects and correctly demodulates larger number of symbols on average but the symbol errors are distributed across all concurrent transmissions. It consistently makes errors in a few of the bits in packets, which result in those packets being discounted from throughput calculation.

This result is an important observation we make in noticing the significance of end-to-end metrics such as Throughput and Packet Reception Rate over Symbol Error Rate, which could be misleading for end users. For SF10 transmissions, due to its longer air-time, the impact of collisions lead to a decreasing network throughput for all demodulators even at transmission rates as low as 1 packet/second per node.

E Appendix: Experimental Setup & Methodology

E.1 Network Experiments

All the network experiments and SNR experiments were performed using 20 transmitting nodes T_1 through T_{20}, a roll-call node R, as well as a beacon node B. R helped in setting up the parameters for each experimental setup without manually changing the setting at each location serially. Each transmitter node (T_1 through T_{20}) would reply only to their specific roll-call message from R. The roll-call messages were sent at a predefined SF and BW. Therefore, all the nodes reset to this SF and BW to listen and respond to the roll-call message. These replies were also used for calculating node-specific SNR at the USRP B200 (serving as a base station for each experiment). Using the received SNR for each node T_i, we could ensure every T_i would hear broadcasts from B. Similarly, B transmits a broadcast to inform the nodes about the settings such as SF, BW, transmission rate for the upcoming experiment. After setup, B would broadcast control messages to all 20 nodes telling each to begin transmitting messages randomly. The beacon information about transmission rates that each node T_i had to follow with a random time offset. The time offsets were generated through Poisson distribution. Each node transmitted to the base station for approximately 30 seconds. All network and SNR experiments were repeated following the aforementioned roll-call and beacon process to ensure no nodes were lost.

E.2 Interference Experiments

Interference tests utilized a similar setup to the Network Experiments, however interfering transmitters were deployed near(<10m) the receiving USRP B200. Interfering nodes included a LoRa transmitter with varying SF's and BW's as well as a GFSK transmitter. Network and interference experiment Arduino code can be located within the Experiment_Setup/Random_Network directory on the OpenLoRa Github page[5].

E.3 Concurrent Transmissions Experiment

Concurrent transmission experiments were performed using a separate beacon node B to continually synchronize up to 12 transmitting nodes T_1 through T_{12}. B would broadcast a short message instructing all nodes T_i to respond within a pre-determined time. These time-limits were determined by recording transmissions from a USRP B200 and measuring total transmit time. Offsets followed a uniform distribution within these time limits thus ensuring random transmission overlaps. The concurrent transmission experiment's Arduino code can be located within the Experiment_Setup/Random_Offset directory on the OpenLoRa Github page[5].

E.4 Packet Time Offset Experiment

To achieve reliable, and precise collisions with microsecond accuracy, we relied on interrupt-driven transmissions. Two transmitting nodes T_1 and T_2 (Adafruit Feather M0 boards), were connected to a third driving node D periodically triggering interrupts via a pin tied on T_1 and T_2. Upon receiving the interrupt, T_1 immediately transmitted its LoRa packet. Node T_2 however utilized a pre-determined delay before transmitting. Delays on T_2 were experimentally gathered ahead of time by measuring packet transmission times on a USRP B200. These delays were then calculated as some fraction of the total transmit time for a single packet, and then flashed onto T_2. Both transmitting nodes were connected to the same breadboard with similarly oriented antennas thus ensuring similar SINR at the receiver. The packet time offset experiment's Arduino code can be located within the Experiment_Setup/Precise_Offset directory on the OpenLoRa Github page[5].

[5]https://github.com/UW-CONNECT/OpenLora