An Open-Source GNU Radio Framework for LoRa Physical Layer and Collision Resolution

Weixuan Xiao, Gil de Sousa, Nancy El Rachkidy, Alexandre Guitton

An Open-Source GNU Radio Framework for LoRa Physical Layer and Collision Resolution

Weixuan Xiao*, Gil De Sousa†, Nancy El Rachkidy*, Alexandre Guitton*‡

*Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne, Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France
†Université Clermont-Auvergne, INRAE, UR TSCF, 63178, Aubière, France
‡Univ Lyon, INSA Lyon, Inria, CITI, F-69621 Villeurbanne, France
Emails: {weixuan.xiao, nancy.el_rachkidy, alexandre.guitton}@uca.fr, gil.de-sousa@inrae.fr

***Abstract*—LoRa (Long Range) is a physical layer designed for low-power wide area networks. It is widely used to provide long range connectivity to Internet of Things devices. In order to improve the limited throughput of LoRa, researchers have proposed several collision resolution algorithms. However, a common software framework to compare these algorithms is lacking. In this paper, we propose an open-source framework using GNU Radio, mainly designed to test and compare collision resolution algorithms, as well as physical layer algorithms. Our framework can help optimizing the parameters of algorithms according to channel conditions such as very low signal to noise ratio for instance. We also discuss technical implementation issues of existing collision resolution algorithms. Finally, we show how our framework can be used for either real experiments on USRPs, or for simulations with a large number of nodes.**

I. Introduction

A Low-Power Wide Area Network (LPWAN) is a network where battery-powered devices are distributed in a large area, typically of tens or hundreds of square kilometers. LPWANs have recently obtained a lot of attention from industrial and academic communities thanks to recent wireless technologies such as LoRa (Long Range) [1], enabling one-hop communications over several kilometers. LoRa is widely used for environmental monitoring of forests [2] or volcanoes [3], smart buildings, smart cities [4], the Internet of Things [5], etc.

The main drawback of LoRa is its very small throughput, which is typically a few hundred bits per second. This throughput is further reduced by the overhead of the LoRaWAN protocol, the most common MAC protocol for LoRa, and by country-dependent regulations on the ISM sub-GHz band used by LoRa. For instance, in Europe, LoRaWAN devices cannot transmit for more than 1% of the time, which reduces the actual throughput to a few bits per second.

As the network density increases, more transmissions occur, and these transmissions might collide. When a collision occurs, the devices typically retransmit their frames, which increases the energy consumption and the end-to-end delay, while further reducing the throughput. Thus, several researchers have proposed algorithms to resolve LoRa collisions.

The performance of LoRa collision resolution algorithms greatly depends on several physical parameters, including the signal-to-noise ratio (SNR) of the LoRa signals (which can even be negative), the presence of a carrier frequency offset (CFO), the presence of a sample time offset (STO) in transmissions, etc. These parameters are not studied consistently in the literature, which makes the existing algorithms hard to compare. Moreover, existing algorithms generally use sophisticated signal processing techniques, which are often programmed in different environments (ex: Matlab, Python, etc.) or not described in the publications. They are difficult to re-implement, and their details might be missing in the papers.

Setting up a real test bed is also difficult, because it is often hindered by the duty-cycle on the ISM band. In order to study collisions, researchers either need a large number of devices (e.g., more than 20) or configure the devices to exceed the maximum duty-cycle of 1% in order to produce enough collisions. Additional collisions might also occur due to pre-deployed nodes from other networks or from concurrent technologies on the ISM band. In addition, the CFO, the STO and the noise are often hard to predict or to control, and their randomness make the experiments difficult to realize.

To the best of our knowledge, there is no framework that compares existing collision resolution protocols with all the parameters of a physical LoRa signal such as the noise strength, the CFO and the STO. Thus, in this paper, we present our framework on GNU Radio in which we implemented several existing collision resolution protocols. This framework can be used to optimize the parameters of an individual collision resolution protocol, or to compare all the protocols in similar conditions. To do so, our contributions can be seen as follows.

- We propose an open-source framework using GNU Radio, aimed at comparing LoRa collision resolution algorithms, or more generally, LoRa algorithms. Our framework is extensible to new algorithms and features, and it enables homogeneous comparisons between algorithms, based either on real USRP hardware or on simulations.
- We carry on with a technical discussion on some implementation issues of existing algorithms, when these issues are missing from the original description. These issues include CFO correction, STO correction, and computational complexity.
- We use our framework to compare the existing algorithms

in a common scenario. We also include an analysis on the memory requirement and on the time complexity of our implementation of these algorithms.

The remainder of this paper is organized as follows. Section II gives details on the LoRa physical layer and presents several representative LoRa collision resolution algorithms. Section III explains our proposed open-source framework. Section IV gives our simulation results in an ideal scenario first, and then in a more realistic scenario. Finally, Section V concludes our work.

II. State of the art

In this section, we first describe the LoRa physical layer, including LoRa modulation, demodulation and coding. Then, we present several representative collision resolution algorithms, which are implemented in our framework.

A. LoRa

LoRa modulation: LoRa is a physical layer for LPWANs based on a chirp spread spectrum modulation. A chirp is a linear frequency sweep over a fixed bandwidth (BW). The chirp is called upchirp when the frequency increases over time, and downchirp otherwise. The initial frequency of a chirp encodes its value. A LoRa uplink frame starts with a preamble composed of 8 upchirps encoding value 0, a network identifier of 2 upchirps, and a delimiter of 2.25 downchirps. The frame continues with the payload encoded with upchirps and including a cyclic redundancy check. A LoRa downlink frame follows the same structure except that upchirps are replaced with downchirps, and conversely.

The duration SD of a chirp depends on a parameter called spreading factor (SF), and is equal to $2^{SF}/BW$. SF also defines the number of bits encoding the value of the chirp. A large SF improves the sensitivity of the receiver, thereby increasing the communication range, but also increases the chirp duration, thereby reducing the throughput.

LoRa demodulation: To demodulate a frame, the receiver uses the preamble to synchronize itself with the transmitter. Then, the receiver captures the time-varying signal $f^m(t)$ for each chirp of value m. The signal is multiplied by an inverse chirp $f'(t)$ encoding value 0, which removes the time-variant. The multiplication on phase is translated into an addition in frequency, as follows (for the case of an upchirp):

$$f^m(t)+f'(t) = \begin{cases} \frac{BW}{SD}\tau_m, t \in [-\frac{SD}{2}, \frac{SD}{2} - \tau_m) \\ \frac{BW}{SD}\tau_m - BW, t \in [\frac{SD}{2} - \tau_m, \frac{SD}{2}) \end{cases} \quad (1)$$

where $\tau_m = (m \times SD)/2^{SF}$ encodes the m-th value. One of the two cases of Equation 1 is outside of the bandwidth. However, when BW is used as the sample rate, the aliasing makes the two cases identical in terms of frequency. Once the time-variant part of the signal is removed and the signal contains a single constant frequency, the receiver performs a Fast Fourier Transform (FFT) on the result, and finds the strongest FFT peak. This peak translates into the chirp value m, encoded with SF bits. After having computing the value of each chirp of the payload, the frame is completely demodulated and can be thus decoded. An open-source LoRa demodulation module[1] for GNU Radio is described in [6].

LoRa coding: LoRa introduces four channel coding steps. First, LoRa uses a Hamming code in order to add redundancy in the bits of the application payload. According to the coding rate (CR) parameter, from CR=1 to CR=4 bits of redundancy are added every 4 bits of payload. Second, LoRa uses a whitening sequence in order to reduce the probability of long sequences of identical bits. Third, LoRa uses a diagonal interleaver to distribute consecutive payload bits into different codewords, and consequently into different transmitted symbols. This allows to distribute the potential errors of a symbol on distant bits, which can be detected more easily, and even corrected if a large coding rate (CR=3 or CR=4) is used. Fourth, a Gray decoding is used for the transmission. This ensures that a decoding error of one in the symbol value translates into a single bit error.

B. Existing collision resolution algorithms

CHOIR [7] leverages tiny frequency offsets in symbols that occur due to hardware imperfections of real devices. These tiny offsets are shown to be stable over the entire duration of a frame. By having the receiver perform an FFT on a large window (typically, ten times the original window size), these tiny frequency offsets can be extracted and can help to identify the transmitter, even when frame collisions occur.

FTrack [8] uses the time offset among colliding frames to decode them. After the LoRa downchirp multiplication, each symbol becomes a series of constant frequencies over SD, called a track. FTrack uses these tracks to match the symbols to the transmitters. FTrack requires that the symbol frontiers of colliding frames are separated by at least $SD/10$.

OCT [9] also uses the time offset to match the symbols to the transmitters. The receiver computes segments according to the symbol edges of the colliding LoRa frames, and performs an FFT on each segment. It stores all the peaks of each FFT, and determines the sent symbols based on the symbols that are repeated on all the corresponding segments. When the colliding frames are quasi-synchronized, the previous approach cannot be used and OCT instead uses the power level of symbols to match them to the correct frame.

CIC [10] is similar to OCT, and also leverages the time offset to divide each SD into segments, with an FFT being performed on each segment. CIC computes the intersection of the set of symbols of contiguous segments to extract the correct symbol. It is important to notice that CIC is one of the few collision resolution algorithms that has been tested under low SNR [10].

In Successive Interference Cancellation (SIC) approaches such as in [11], a strong LoRa signal can be demodulated even if it collides with weaker signals, thanks to the capture effect. By reconstructing the stronger signal and removing it from the superposed signals, it is possible to attempt to

[1]The module from [6] is available at https://github.com/rpp0/gr-lora.

iteratively demodulate the next stronger signal. This requires the signal-interference ratio to be above 6 dB, where the signal interference ratio is formally defined as:

$$SIR = 10\log_{10}\frac{S_{ref}}{S_{int}} = 20\log_{10}\frac{A_{ref}}{A_{int}}, \quad (2)$$

where S_{ref} and A_{ref} are the energy and amplitude of the reference signal, and S_{int} and A_{int} are the energy and amplitude of the interfered signal.

In [12], we showed that frame collisions might cause symbol ambiguities in most collision resolution algorithms. These ambiguities come from noise, close symbol edges in time-based approaches, or similar receive power in power-based approaches. We proposed to benefit from the LoRa coding, and especially from the diagonal interleaver and the Hamming code, in order to remove most symbol ambiguities.

III. Framework architecture in GNU Radio

In this section, we first describe the GNU Radio programming environment. Second, we present the architecture and the general modules of our framework. Third, we present our implementation of a few existing collision resolution algorithms, as well as how to integrate a new algorithm into our framework. Finally, we explain how to add extra features.

A. GNU Radio primer

GNU Radio [13] is a free and open-source software providing signal processing blocks for software-defined radios.

The basic element in GNU Radio is the block. A block usually contains input and output ports, from which it can respectively receive and send data to other blocks. The data is either a continuous stream (*e.g.*, a signal is usually a stream of complex float numbers representing the in-phase (I) and quadrature (Q) components of the signal) or a discrete message containing data with polymorphic type (*i.e.*, a generic structure holding a variety of data such as a number, a string, a tuple, a list or a dictionary). Streamed data is also able to carry discrete messages, named tags in GNU Radio. Each block can be considered as a state machine with some initial parameters. A block consumes the data from its input ports, processes it, and produces data for the output ports. The blocks do not require to load all the data at once. GNU Radio has an intuitive graphical interface where blocks are interconnected through links between output and input ports, as shown on Figure 1. In the example of Figure 1, the output of the "Throttle" block is connected to the input of the "Resampler" block. Blue ports use a stream of complex float numbers, while white ports represent messages.

B. Architecture of the proposed framework

Our proposed open-source framework is available at Zenodo[2]. The core of our framework is composed of two parts: a decoder and a node abstraction. The decoder contains an optional LoRa preamble detector, a demodulator (which can implement a collision resolution algorithm) that retrieves demodulated symbols from a signal, and a LoRa decoder that transforms symbols into application bits. The node abstraction is capable of generating superposed LoRa signals with different parameters from a simulated network.

1) Decoder: The decoder part implements a single LoRa demodulator with a known SF. Figure 1 shows the structure of the entire receiver. The receiver takes the samples from a file (block "File Source") or from a connected SDR hardware (block "UHD: USRP Source"). The "LoRa Preamble Detect" block processes the samples to find a preamble. The samples, combined with preamble information of the detected frame, are then processed by the "Payload Decoder" block in order to synchronize the receiver with the transmitted symbols. In the example of Figure 1, the standard LoRa decoding algorithm is used. Then, the demodulated symbols of the entire frame are passed as a message to our "LoRa Decoding" block. This block decodes and outputs the original data from the symbols.

In a simulated scenario, the "LoRa Preamble Detect" block can be removed. In this case, the frame information is directly dispatched to the decoders in a tag, along with the samples of the base band signal.

Any GNU Radio block can be replaced by another block with the same input and output, as long as they implement similar features. Such decoupling can help to develop, evaluate and improve the preamble detection algorithm, the payload demodulating algorithm (which is often the core of the collision resolution algorithms) or the decoding algorithm.

2) Node abstraction: Figure 2 presents two node abstractions in our framework, depicted here as Node 1 and Node 2. Node abstractions are used to generate application data. This data can be randomly generated in GNU Radio for test purpose (as in Node 1) or from an external application (as in Node 2 for an example with UDP) including a network simulator. Note that the communication between the external application and GNU Radio can be made through TCP, UDP or ZeroMQ connections [13]. The generated data is packed as a message and passed to our LoRa Encoding block, which performs encoding and modulation, and produces LoRa signals.

The signals generated by our LoRa Encoding block have a normalized amplitude and are perfectly synchronized at the sample level. It is possible to implement a propagation model by delaying the signals (see the "Delay" blocks on Figure 2, which can be configured with a given number of samples) and by reducing the strength of the received signal with a given factor (see the "Multiply Const" blocks on Figure 2). Then, the signals from the two nodes are added, and the output is a colliding signal. Note that a very similar setup can be configured to evaluate the impact of low SNR, by adding a strong random noise to a weak signal. Also notice that this delay is also used to implement desynchronized transmissions, required by several existing collision resolution algorithms.

For instance, let us consider that we deploy a LoRa gateway which can receive the signals of two nodes, with a relative power difference of 20 dB. From Equation 2, we have $A_{ref} = A_{int} \cdot 10^{\frac{SIR}{20}}$. If we set the amplitude of the weaker signal A_{int}

[2]Available at https://doi.org/10.5281/zenodo.6421804

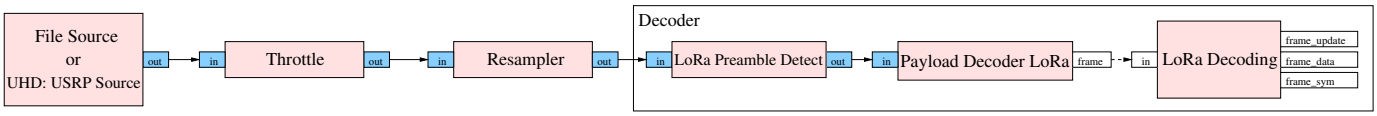

Fig. 1. Decoding LoRa frames from a signal sample, obtained from a file or from a USRP SDR hardware.

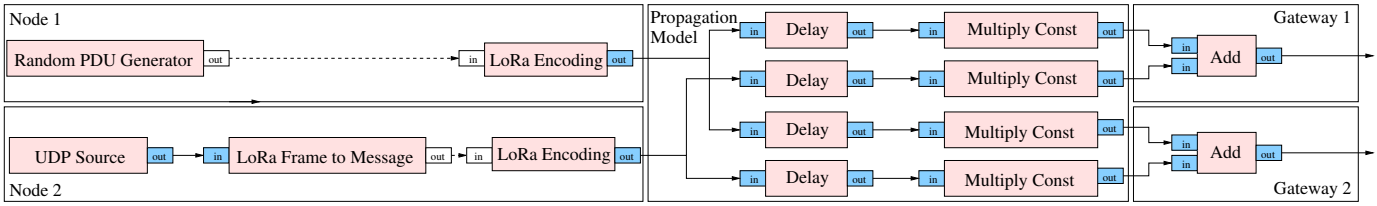

Fig. 2. Generating signals with the node abstraction and multiple gateways.

to 1, this means that the amplitude of the stronger signal A_{ref} has to be multiplied by $10^{20/20} = 10$.

Multiple gateways can be implemented by having as many preamble detector and payload decoder blocks as there are gateways (or more precisely, as there are demodulators). For each gateway, the parameters corresponding to the propagation model are likely to be different, as the signals received at each gateway will experience different channel losses. Therefore, we can set different parameters in the propagation models by calculating them in advance using the relative distances between the nodes and gateways.

C. Implementations of collision resolution algorithms

We hereby discuss the implementations of various algorithms in our framework.

Our implementation of the default LoRa demodulation consumes 2^{SF} samples during SD, multiplies it with a downchirp, performs an FFT on the samples, and finds the strongest FFT bin. The index of this bin is the obtained symbol value.

In CHOIR, the decoder takes 2^{SF} samples from the synchronized frame during SD. CHOIR adds a zero padding to expand the FFT size to ten times its default value. Then, it stores the strongest FFT bin peaks into ten lists. The symbols are taken from these lists when decoding.

FTrack requires parameters whose values are not given in the reference article [8]: the number n_{step} of steps between each FFT to extract a track, and the size N_{win} of an FFT window. Our block uses this sliding window on N_{win} samples to calculate the FFTs. The window advances by steps of n_{step} samples. To extract the track of an entire symbol, we thus need to review $2^{SF} + N_{win}$ samples.

OCT aims at a real-time decoding of LoRa collisions. To do so, OCT divides each symbol duration SD into several segments according to the time offset between the frames. We implemented OCT with a minimal resolution of 1/8-th of SD, that is 0.128 ms for $SF = 7$ and $BW = 125$ kHz. The decoder needs to perform at most as many FFT within every SD as there are collided frames.

CIC was not implemented in our framework, as the authors of CIC already made their code available in both Python and Matlab.

All these algorithms have in common the computation of several FFTs. This allows us to compare them depending on how many FFTs they perform, and on the size of these FFTs. The space complexity also mainly depends on the number of samples that a receiver has to review. Table I summarizes the complexities of each algorithm. We use the following notations: N is the size of each FFT (which depends on the algorithm), n_{sym} is the number of symbols in the collided frames, and n_c is the number of frames in collision. Note that in the case of CIC, the space complexity is $n_{sym} \times N$ according to the code published in [10], but it could be reduced to N by considering only the samples during an SF for each FFT.

SIC-based algorithms for LoRa need to store the samples during the entire superposition (at least $n_{sym} \times 2^{SF}$ samples), so as to do the subtraction and recover a weaker signal. We believe the SIC-approach needs to be implemented by storing all the samples and analyzing them several times, in order to be compliant with our framework (and more generally, with the GNU Radio design).

D. Extra characterizations of LoRa signals

Noise is considered as an Additive Gaussian White Noise (AGWN) with zero mean [14]. In our framework, such noise can be simulated by the "Noise Source" and added to the superposed signals. The relative amplitude is computed according to Equation 2:

$$A_{noise} = A_{ref} \times 10^{-\frac{SNR}{20}} \tag{3}$$

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

Carrier frequency offset (CFO) is not negligible in a real transmission. To simulate CFO in the base band signal in our framework, it is possible to add a frequency offset in the "LoRa Encoding" block as part of the node abstraction. The CFO is translated into a time offset in the synchronization by the "LoRa Encoding" block.

Sample time offset (STO) can occur when the re-sampler in the receiver side is not perfectly synchronized with the original

TABLE I
COMPLEXITIES OF THE IMPLEMENTED COLLISION RESOLUTION ALGORITHMS.

Algorithm	Number of samples	FFT size	Time complexity of FFT per symbol	Space complexity of FFT per symbol
LoRa	2^{SF}	$N = 2^{SF}$	$\mathcal{O}(N \log N)$	N
CHOIR	2^{SF}	$N = 10 \times 2^{SF}$	$\mathcal{O}(N \log N)$	N
FTrack	various ($\leq 2^{SF}$)	$N = 2^{SF}$	$n_{step} \times \mathcal{O}(N \log N)$	$N + N_{win}$
OCT	various ($\leq 2^{SF}$)	$N = 2^{SF}$	$n_c \times \mathcal{O}(N \log N)$	N
CIC	various ($\leq 2^{SF}$)	$N = 2^{SF}$	$n_c \times \mathcal{O}(N \log N)$	$n_{sym} \times N$

signals. To simulate the STO in our framework, we can simply configure the interpolation parameter in the "LoRa Encoding" block to generate the signals with a higher sample rate, and re-sample the signal with sample-level offsets.

Large networks are usually difficult to simulate as they require creating and interconnecting several blocks, which is tedious and error-prone. To facilitate this, we provide a script with an optional configuration file to automatically generate a complex network. Suppose that 100 nodes are deployed around a LoRa gateway, following a uniform distribution between 500 and 1000 meters. The path loss model is a log distance model. The noise amplitude A_{noise} equals to the amplitude of the weakest signal A_{signal}. Such a scenario can be described as follows:

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

IV. SIMULATION RESULTS

In this section, we use our framework to evaluate the collision resolution algorithms under different scenarios. Our goal is not to identify the best algorithm, but rather to show that they can be compared in an homogeneous setup. We also try to highlight results that are unknown (*e.g.*, because the algorithms were not systematically tested with negative SNR) or inconsistent with previous works.

A. Scenario 1: impact of the duty-cycle

In the first scenario, we evaluate the ideal performance of different decoding algorithms with different duty-cycle constraints (or, equivalently, with different traffic load) without noise. The network consists of one gateway and a varying numbers of nodes. All the nodes have the same relative distance to the gateway, which is the worst-case for the algorithms based on power level. For the FTrack algorithm, we set n_{step} to 8 and a sliding window of SD. Each node periodically sends at least five frames of 20 random bytes. The time on air of the frames is 57.3 ms for SF7 and 1.34 s for SF12. We set the duty-cycle to 1% and 10%. We add a random waiting time between each transmission in order to desynchronize the nodes. The maximum waiting time is set to 100 times the time on air for 1% duty-cycle, which is 5.7 s and 135 s for SF7 and SF12 respectively, and 10 times the time on air for 10% duty-cycle, which is 0.57 s for SF7 and 13.5 s for SF12. Therefore, the simulation time is 57 s and 1350 s for 1% duty-cycle, 5.7 s and 135 s for 10% duty-cycle. The results are averaged over 20 simulations.

Figure 3 shows the throughput for a varying number of nodes, for SF7 (on the left) and SF12 (on the right), for four algorithms and two possible duty-cycles (1% or 10%). Our results show that the setup with 1% duty-cycle does not generate enough collisions: collision resolution algorithms have similar performance as LoRa. LoRa saturation is not reached with 100 nodes at 1% duty-cycle, while it is reached at about 30 nodes with 10% duty-cycle. It can be seen that FTrack performs the best in all setups, which is not consistent with [10]. Indeed, the original paper for FTrack [8] mentions that they use FTrack to decode the frames with similar power levels and that the capture effect can be leveraged to decode frames with different power levels. In [10] however, FTrack is evaluated with collided frames of different power levels though, which degrades the results.

B. Scenario 2: impact of the SNR

In this scenario, we set the duty-cycle to 10% and deploy 50 nodes in the network for guaranteed collisions. The nodes are randomly deployed in a ring between 500 m and 1000 m around the gateway. The relative receiving power levels are calculated through a log distance loss model. The path loss L in dB is $L = L_0 + 10nlog_{10}(\frac{d}{d_0})$, where $n = 3.76$ is the path loss exponent, $d_0 = 1$ is the reference distance in meter, $L_0 = 7.7$ is the path loss at reference distance, and d is the distance between a node and the gateway. We vary the strength of the noise so that the SNR varies from 20 dB to -10 dB (recall that a negative SNR means that the LoRa signals are weaker than the noise). We also test two sliding windows for FTrack: $N_{win} = 2^{SF}$ and $N_{win} = 2^{SF}/2$.

Figure 4 shows the throughput for a varying SNR, for SF7 (on the left) or SF12 (on the right), for four algorithms, 50 nodes, and 1% duty-cycle. Our results show that FTrack with full FFT window performs the best with SF7, while OCT reaches the best performance with SF12. The performance of most algorithms appears to be slightly impacted by the SNR, except for CHOIR, and for FTrack on full window with SF12. At SF7, throughput degradation can be seen from SNR=0 dB and lower for CHOIR, FTrack with full window, and OCT. At SF12, such degradation appears from SNR=10 dB for CHOIR and FTrack with full window, and from SNR=5 dB for OCT. Note that the impact of the SNR on the decoding algorithms was only known for LoRa and for CIC.

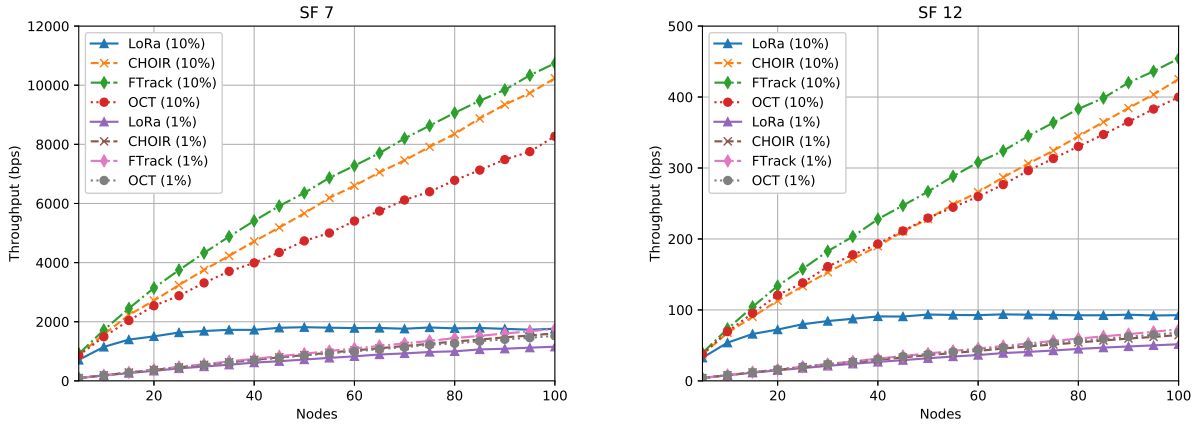

Fig. 3. Study of the throughput as a function of the number of nodes, with 1% or 10% duty-cycle, and with SF7 (on the left) and SF12 (on the right).

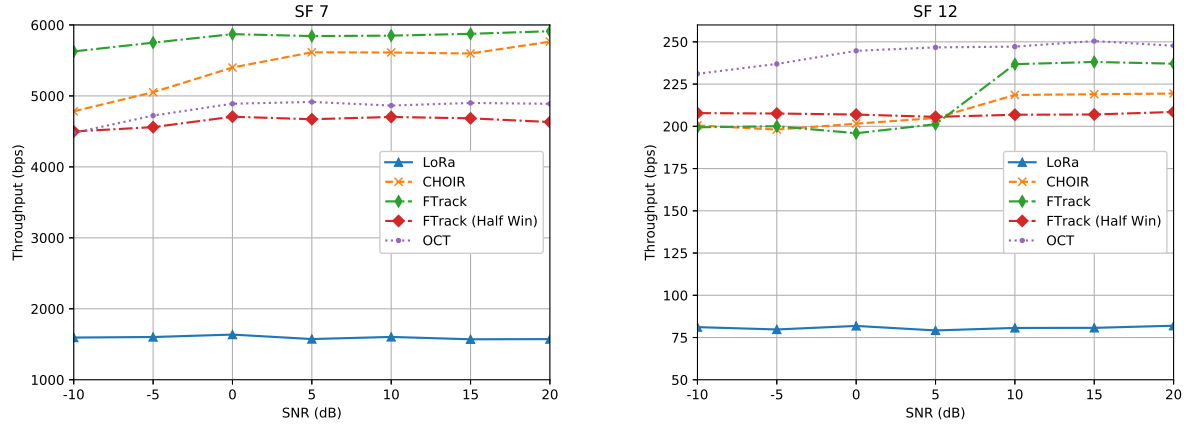

Fig. 4. Study of the throughput as a function of the SNR, for 50 nodes and a duty-cycle of 10%, and with SF7 (on the left) and SF12 (on the right).

V. Conclusion

LoRa technology is used in a large variety of LPWAN applications. Several sophisticated collision resolution algorithms have been proposed in order to increase the throughput of LoRa. However, a common framework to compare these algorithms, or to test new ones, is lacking. In this paper, we described our open-source platform for GNU Radio. We discussed how we implemented several existing collision resolution algorithms, as well as specific features in a realistic LoRa network. We used our framework to compare the performance of several algorithms in various scenarios, and with various metrics including their time and space complexity. As an example, we use the ability of our framework to finely control LoRa signal features in order to study the impact of SNR. We believe that our framework can be used and improved by the community to implement and evaluate new collision resolution algorithms or new signal processing techniques.

References

[1] Semtech Corporation, "AN1200.22 LoRa Modulation Basics," Semtech, Application note Revision 2, 2015. [Online]. Available: http://www.semtech.com/uploads/documents/an1200.22.pdf

[2] S. Park, S. Yun, H. Kim, R. Kwon, J. Ganser, and S. Anthony, "Forestry monitoring system using LoRa and drone," in *International Conference on Web Intelligence, Mining and Semantics (WIMS)*, 06 2018, pp. 1–8.

[3] S. Awadallah, D. Moure, and P. Torres-Gonzáles, "An Internet of Things (IoT) application on volcano monitoring," *Sensors (Basel)*, vol. 19, no. 21, 2019.

[4] A. Lachtar, T. Val, and A. Kachouri, "Elderly monitoring system in a smart city environment using LoRa and MQTT," *IET Wireless Sensor Systems*, vol. 10, 11 2019.

[5] S.-Y. Wang, Y.-R. Chen, T.-Y. C. Chen, C.-H. C. Chang, Y.-H. Cheng, C.-C. H. Hsu, and Y.-B. Lin, "Performance of LoRa-based IoT applications on campus," in *VTC Fall (IEEE Vehicular Technology Conference)*, 2017, pp. 1–6.

[6] P. Robyns, P. Quax, W. Lamotte, and W. Thenaers, "A Multi-Channel Software Decoder for the LoRa Modulation Scheme," in *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security*, Funchal, Madeira, Portugal, 2018.

[7] R. Eletreby, D. Zhang, S. Kumar, and O. Yağan, "Empowering Low-Power Wide Area Networks in Urban Settings," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Aug. 2017.

[8] X. Xia, Y. Zheng, and T. Gu, "FTrack: Parallel Decoding for LoRa Transmissions," in *Proceedings of the ACM 17th Conference on Embedded Networked Sensor Systems*, New York, Nov. 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3356250.3360024

[9] Z. Wang, L. Kong, K. Xu, L. He, K. Wu, and G. Chen, "Online

concurrent transmissions at LoRa gateway," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 2331–2340.

[10] M. O. Shahid, M. Philipose, K. Chintalapudi, S. Banerjee, and B. Krishnaswamy, "Concurrent interference cancellation: decoding multi-packet collisions in LoRa," in *ACM SIGCOMM*, 2021, pp. 503–515.

[11] B. Laporte-Fauret, M. A. Ben Temim, G. Ferre, D. Dallet, B. Minger, and L. Fuche, "An Enhanced LoRa-Like Receiver for the Simultaneous Reception of Two Interfering Signals," in *Proceedings of Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2019.

[12] W. Xiao, N. E. Rachkidy, and A. Guitton, "Recovering Colliding LoRa Frames from Uncertainties Using LoRa Coding," in *Proceedings of the IEEE 46th Conference on Local Computer Networks (LCN)*, 2021.

[13] GNU Radio, accessed Feb. 2022. [Online]. Available: https://www.gnuradio.org

[14] O. Georgiou and U. Raza, "Low Power Wide Area Network Analysis: Can LoRa Scale?" *IEEE Wireless Communications Letters*, pp. 162–165, Apr. 2017.