TnB: Resolving Collisions in LoRa based on the Peak Matching Cost and Block Error Correction

Raghav Rathi and Zhenghao Zhang
rrathi@fsu.edu,zzhang@cs.fsu.edu
Florida Sate Univesity, Tallahassee, Florida 32306, USA

ABSTRACT

LoRa has emerged as one of the main candidates for connecting low-power wireless IoT devices. Packet collisions occur in LoRa networks when multiple nodes transmit wireless signals simultaneously. In this paper, a novel solution, referred to as TnB, is proposed to decode collided LoRa signals. Two major components of TnB are Thrive and Block Error Correction (BEC). Thrive is a simple algorithm to resolve collisions by assigning an observed signal to a node according to a matching cost that reflects the likelihood for the node to have transmitted the signal. BEC is a novel algorithm for decoding the Hamming code used in LoRa, and is capable of correcting more errors than the default decoder by jointly decoding multiple codewords. TnB does not need any modification of the LoRa nodes and can be adopted by simply replacing the gateway. TnB has been tested with real-world experimental traces collected with commodity LoRa devices, and the results show that TnB can increase the median throughput by 1.36× and 2.46× over the state-of-the-art for Spreading Factors (SF) 8 and 10, respectively. Simulations further show that the improvement is even higher under more challenging channel conditions.

CCS CONCEPTS

• **Networks → Wireless access points, base stations and infrastructure**.

KEYWORDS

LoRa, Multi-packet collisions, Error correction

ACM Reference Format:
Raghav Rathi and Zhenghao Zhang. 2022. TnB: Resolving Collisions in LoRa based on the Peak Matching Cost and Block Error Correction. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3555050.3569132

1 INTRODUCTION

LoRa [7] has emerged as a strong candidate for Low-Power Wide-Area Networks (LPWAN), where a large number of nodes connect to a gateway over long distances with wireless links. In LoRa, nodes may transmit packets at the same time, causing collisions. There have been increasing interests in enhancing LoRa by decoding collided packets, which have achieved significant gains over the original LoRa [12, 15, 18, 20, 22–24, 26–28].

In this paper, *TnB* is proposed, which is a novel solution to decode collided LoRa packets. Two main components of TnB are *Thrive* and *Block Error Correction (BEC)*. When a collision occurs, the received signal contains multiple *peaks*, where each peak is generated by a node. In order to decode the collided packets, the LoRa receiver should find the *owner* of each peak, where the owner refers to the node that transmitted the peak. Thrive is a simple yet effective algorithm for finding the owners of the observed peaks. In LoRa, the Hamming Code is used for Forward Error Correction, which is simple but offers only limited error correction capabilities. For example, with Coding Rate (CR) 4, the Hamming code has 4 data bits and 4 parity bits per codeword, and the default decoder can correct only 1-bit errors. BEC is an algorithm for decoding the same Hamming code in LoRa but can correct more errors than the default decoder. For example, with CR 4, BEC can correct all 1-symbol and 2-symbol errors, and even over 96% of 3-symbol errors. Thrive and BEC do not need any modification of the LoRa nodes, allowing a network operator to simply replace the gateway and enjoy immediate performance improvements. The computation complexity of Thrive and BEC are both moderate. TnB has been tested with experimental traces collected with commodity LoRa devices, and the results show that TnB can increase the median throughput by 1.36× and 2.46× over the state-of-the-art for Spreading Factors (SF) 8 and 10, respectively. Additional evaluation with simulations show that TnB achieves even higher gains when the wireless channel is more challenging with stronger multi-path and higher fluctuations. The source code of TnB is available at [5].

Thrive is based on the well-known fact that a peak *thrives*, i.e., is the highest, when the signal is processed with the specific parameters of the owner of the peak. Thrive jointly considers three features that can distinguish a node from others, namely, the symbol boundary, the Carrier Frequency Offset (CFO), and the height of other peaks observed from the same node, and therefore is more effective than those consider only a subset of the features. The key novelty of Thrive is to *calculate a matching cost that extracts the information embedded in these features based only on the height of the peaks, therefore enjoying a low computation complexity.* BEC corrects more errors than the default decoder by exploiting a special dependency of the codewords in LoRa. That is, a demodulation error will lead to errors in multiple codewords at the same location. The key novelty of BEC is to *decode such codewords jointly, which allows BEC to achieve an error correction capability beyond the traditional bound based on the minimum Hamming distance of the codewords.*

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 discusses the background of LoRa. Section 4 gives an overview of TnB. Section 5 explains Thrive. Section 6 explains BEC. Section 7 explains the synchronization and CFO estimation. Section 8 describes the evaluation. Section 9 concludes the paper.

2 RELATED WORK

LoRa packet collision resolution has attracted increasing interest in recent years [21]. Choir [15] distinguishes peaks from different nodes by the fraction CFOs, which likely lead to unique fractional peak locations. mLoRa [24] leverages the time offset of the transmitted packets and successively recovers and subtracts the collision-free signal. FTrack [27] detects the interfering chirp from the discontinuity of the frequency track. CoLoRa [23] is based on the fact that a misaligned chirp generates peaks at the same location in two consecutive symbols, where the peak height ratio of the two peaks is proportional to the amount of misalignment. Nscale [22] processes the received signal with a modified downchirp, which leads to different effects on the peaks from different nodes. SCLoRa [18] distinguishes the peaks based on the power of the peaks and the change of the peaks when the processing window slides to the left or the right. Pyramid [28] is based on the observation that the peak height of a node increases then decreases when the signal is processed with a sliding window, where the highest height is achieved when the window matches the actual symbol. AlignTrack [12] is based on a similar observation but processes the signal with the symbol boundaries of the detected packets. PCube [26] identifies the signals from different nodes in the spatial domain with multiple antennas. CIC [20] cancels the interference and leaves only the target peak.

TnB is significantly different from the work listed above in multiple aspects. First, TnB assigns peaks to the nodes by calculating a matching cost based on multiple unique features of the node, namely, the symbol boundary, the CFO, and the peak height history, while the existing work exploits only a subset of the features. Second, TnB makes a unique contribution in the error correction decoding of LoRa, which has not been explored. Third, the peak assignment algorithm in TnB is very simple and does not involve costly computations or a large number of antennas.

There have been other attempts to improve LoRa. OPR [11] exploits multiple gateways to recover a lost packet, while TnB runs at a single gateway. BICM Decoding [16] and List Decoding [17] enhance the error correction of LoRa; however, they are designed for single-node transmissions and cannot be applied to signals with collisions because strong signals are processed first which could be from other nodes. TnB is designed for unmodified LoRa nodes and is therefore different from those add own application layer codes [13, 19].

3 BACKGROUND OF LORA PHY

The LoRa *Spreading Factor* (SF) is an integer that can be from 6 to 12. The *upchirp*, denoted by C, is a complex vector of length 2^{SF} with unit amplitude but linearly increasing frequency. The conjugate of C is denoted as C' and is called the *downchirp*. A LoRa packet consists of LoRa *symbols* transmitted back-to-back, where a symbol is a cyclically shifted version of C. For example, the top of Fig. 1(a) shows the real part of a symbol with SF 8, which is C shifted by one location. A symbol modulates SF bits of data, because C can be shifted by h locations where $h \in [0, 2^{SF} - 1]$.

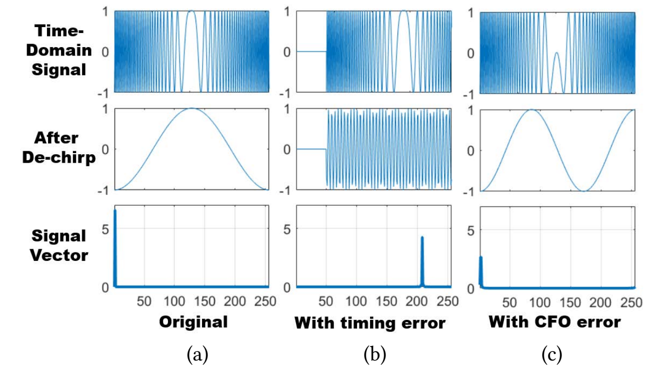

Figure 1: (a). Modulation and demodulation of a LoRa symbol. (b) Sensitivity of the peak height to timing error. (c) Sensitivity of the peak height to CFO.

At the receiver, a received symbol is denoted as β, which is also a complex vector of length 2^{SF}. The receiver first *de-chirps* β by computing $\gamma = \beta \odot C'$, where $\odot$ denotes the element-wise multiplication of two vectors. If the transmitted symbol shifts C by h locations, γ is a sinusoid that completes h cycles in the symbol time, as shown in the middle of Fig. 1(a). The *signal vector*, denoted as Y, is defined as $Y = |FFT(\gamma)| \odot |FFT(\gamma)|$, which is a vector with a *peak* at location h, as shown at the bottom of Fig. 1(a). The receiver can therefore infer the value of the transmitted data. When there are multiple receiving antennas, the signal vector is the summation of the individual signal vectors of all antennas.

To demodulate a symbol, the receiver needs to find the correct *symbol boundary*, which refers to the start of the symbol, and cancel the CFO. The peak height can be reduced with a mismatching symbol boundary and residual CFO. A case with misaligned symbol boundary is shown in Fig. 1(b), where, as only part of the symbol is used in the calculation, the peak is noticeably lower. A case with residual CFO is shown in Fig. 1(c), where the residual CFO leads to an additional 0.5 cycles in γ and a much lower peak.

At the sender, the data from the upper layer is encoded by an (8,4) Hamming code, followed by procedures such as whitening. At the receiver, the reverse process is applied to the demodulated bits, including de-whitening and error correction. The generator matrix of the Hamming code is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

A codeword can be generated by multiplying the data, which is a 1 by 4 binary vector, with the matrix. A complete codeword is a 1 by 8 binary vector, where the first 4 bits are the data bits and the remaining bits are the parity bits. The Coding Rate (CR) is an integer between 1 and 4 and is the number of parity bits transmitted per codeword. If the CR is between 2 and 4, the first CR parity bits are transmitted. An exception is when the CR is 1, in which case the parity bit is the checksum of the 4 data bits. For example, if the data is '1001,' the complete codeword is is '10011100', which is the summation of rows 1 and 4 of the generator matrix. If the CR is 3, the transmitted codeword is '1001110.' The Hamming code

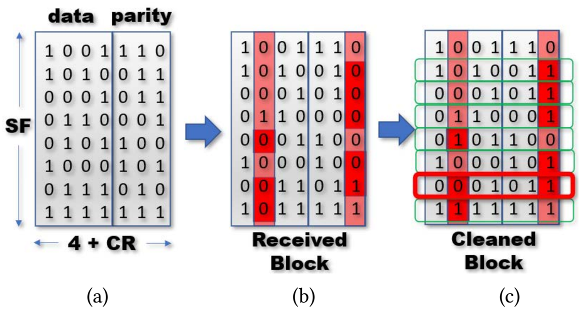

Figure 2: A code block in LoRa with SF 8 and CR 3.

guarantees detecting $t-1$ errors if the minimum Hamming distance between the codewords is t, and correcting t errors if the minimum Hamming distance is $2t+1$. Therefore, CR 1 and CR 2 offer 1-bit error detection, while CR 3 and CR 4 offer 1-bit error correction.

LoRa organizes the codewords in *blocks*, where each block is a SF by $4+CR$ binary matrix. In a block, each row is a codeword and each column contains the bits to be transmitted by one symbol. As an example, a code block is shown Fig. 2(a), where the SF is 8 and the CR is 3. Let the *received block* be the block received by the receiver, potentially with some symbols corrupted. Fig. 2(b) shows the received block, where symbols 2 and 7 have been corrupted. Note that, as the error values are random, a corrupted symbol usually does not flip all bits in a column. The default decoder replaces each row of the received block with a codeword that is closest to the row, i.e., with the minimum Hamming distance, producing the *cleaned block*. Fig. 2(c) shows the cleaned block, where, in all rows except row 7, the number of errors are 1 or 0, which can be corrected by the default decoder. Row 7 however has 2 errors, which is beyond the error correction capability of the Hamming code. The default decoder "snaps" row 7 to the codeword with the minimum Hamming distance, which differs with row 7 in column 3, producing an error. Note the BEC can decode this block correctly, as will be explained in Section 6.1.

A LoRa packet starts with the preamble, followed by the Physical Layer (PHY) header, then the payload. The preamble allows the receiver to detect the packet, which typically starts with 8 upchirps, followed by 2 symbols called the *sync symbols*, then 2.25 downchrips. The PHY header consists of 8 symbols and uses CR 4, from which the receiver can learn the CR and the length of the payload.

4 OVERVIEW OF TNB

TnB consists of four components, as shown in Fig. 3. The first is the packet detection component, which takes the received time-domain signal as input and detects packets, at the same time finding the symbol boundary and the CFO of each packet. The second is the signal calculation component, which takes the list of detected packets, as well as the time-domain signal, as input, and calculates the signal vectors of each packet, where the signal vectors of a particular packet are calculated by aligning to its estimated symbol boundary and correcting the CFO according to its estimated CFO. The core of TnB are the third and fourth components, namely, Thrive and BEC. Thrive takes the signal vectors as input, and assigns peaks to the packets. BEC takes the peak locations of each packet as input, and decode them into data bits. Thrive and BEC can be jointly used as in TnB, or used separately and combined with other methods, such as combining BEC with CIC [20] in Section 8.

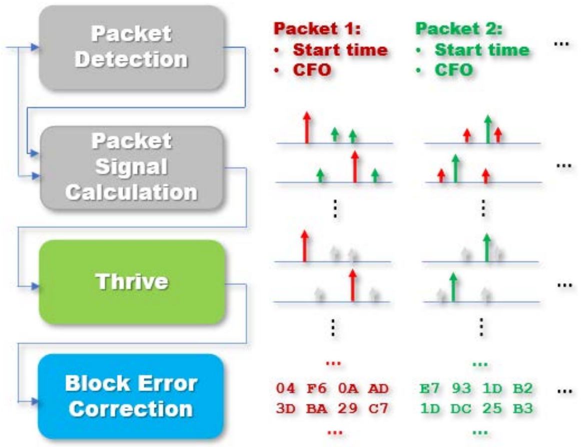

Figure 3: Overview of TnB.

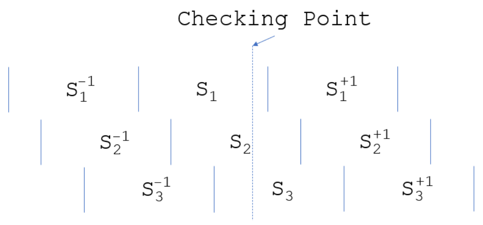

Figure 4: The checking point and symbols in Thrive.

Starting from the first sample of the received signal, every 2^{SF} samples, which is the length of the symbol, is a *checking point*. At each checking point, Thrive examines the symbols that intersect the checking point, and assigns one peak to each symbol. Once the PHY header of a packet has been received, BEC is called to decode the PHY header to learn the CR and the length of the payload. Once the last symbol of the payload has been received, BEC is called to decode the payload. Thrive also reexamines the received signal for a second time to decode packets that failed at the first attempt, because many packets may have been decoded correctly and their peak locations are known and can be masked.

The packet detection component is described in Section 7. The signal calculation component is very straightforward. Thrive and BEC are explained in the following.

5 PEAK ASSIGNMENT WITH THRIVE

Thrive is a simple algorithm for assigning peaks to packets when there are multiple peaks in the signal vector.

5.1 Challenges

A checking point is shown in Fig. 4, which intersects 3 symbols denoted as S_1, S_2, and S_3, respectively, where the symbols are sorted according to their boundaries with the first being S_1. Symbol i belongs to packet i, which is transmitted by node i. Symbols of packet i right before and after S_i are denoted as S_i^{-1} and S_i^{+1}, respectively.

The challenges of peak assignment can be seen in Fig. 5, which shows the signal vectors of the symbols in Fig. 4. As can be seen in

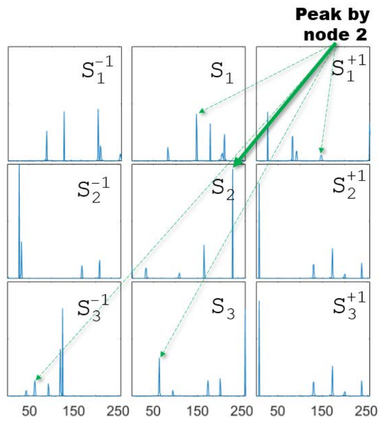

Figure 5: An example of signal vectors and peaks.

the figure, a signal vector contains multiple peaks, while the owners of the peaks are unclear. The number of peaks in a signal vector is more than the number of nodes, because the signal from one node generates a peak not only in its own signal vector, but also in the signal vectors of other nodes if the symbols overlap. For example, node 2 generates a peak in S_2. However, as S_2 overlaps with S_1 and S_1^{+1}, the same signal also generates in peaks in the signal vectors of S_1 and S_1^{+1}, which are at different locations and are of different heights. The same signal also generates peaks in the signal vectors of S_3^{-1} and S_3.

As mentioned earlier, Thrive jointly considers the symbol boundary, the CFO, and the peak height history. While these features have been exploited to various degrees in the past, combining them further improves the distinguishability of the peaks. The challenge, however, is how to combine them effectively without incurring high computation cost.

5.2 Core Ideas

Thrive is based on the observation is that the height of a peak is highly sensitive to the symbol boundary and the CFO, and is also highly correlated with peaks in nearby symbols from the same node. *Therefore, reversely, by examining how the peak height varies in the signal vectors of different nodes and how it differs from those in nearby symbols, the owner of the peak can be identified.* As the signal vectors have been found by the packet signal calculation component, no additional heavy computations are needed in Thrive.

To elaborate, first, note that the peak height is reduced if the signal is processed with incorrect symbol boundary and CFO, as shown in Fig. 1(b) and Fig. 1(c). Therefore, as long as the nodes have different symbol boundaries and CFOs, the signal from a node likely generates the highest peak in its own signal vector, rather than in those of other nodes. To exploit this observation, let *siblings* refer to the set of peaks in the signal vectors of different nodes generated by the same transmitted symbol. For example, the 5 peaks pointed by the arrows in Fig. 5 are siblings. From the receiver's point of view, as the highest peak among all siblings is in the signal vector of node 2, the peak matches the best with the parameters of node 2, and therefore node 2 is most likely the owner of the peak. The *sibling cost* of a peak is therefore defined based on its relative height

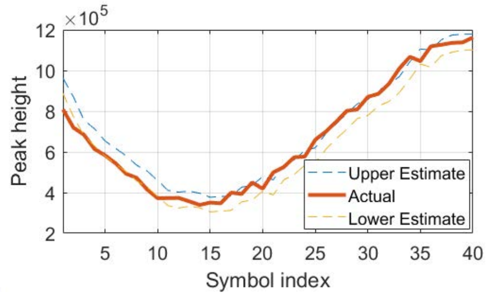

Figure 6: The peak height history of a packet, along with the upper and lower estimates.

among its siblings. The higher the peak, the lower the cost, and the more likely the node to which the signal vector belongs is the owner of the peak.

The peak height history can be very useful, because the signal powers of the nodes likely differ, resulting in different peak heights; at the same time, the peaks from the same node should bear some similarities. An example is shown in Fig. 6 for a packet, where, although the signal fluctuates, the peak height still follows some trend. The *history cost* of a peak measures the deviation of the peak from the expected peak height of a node based on the past observations of the node. The smaller the deviation, the lower the cost, and the more likely the node is the owner of the peak. The expected peak height is computed by a curve-fitting algorithm capable of tracking the changes caused by channel fluctuations, bootstrapped by the peaks in the preamble.

Lastly, the sibling cost and the history cost are linearly combined into the *matching cost*, which represents the likelihood of a node to be the owner of a peak.

5.3 Details of Thrive

The details of Thrive are explained in the following for a generic checking point.

5.3.1 Notations. Let M be the number of symbols intersecting the checking point. Let τ_i be the time difference between the boundaries of S_i and S_1. Let δ_i be the CFO difference between S_i and S_1, where the CFO is measured by the number of cycles the CFO sinewave completes in a symbol. Let $\alpha_i = \tau_i + \delta_i$. For each $1 \le i \le M$, Thrive runs a peak finding algorithm [29] to find the peaks in the signal vector of S_i, denoted as $\{P_{i,1}, P_{i,2}, ...\}$, where the maximum number of peaks in a symbol is currently $2M$. The height of $P_{i,h}$ is denoted as $\eta_{i,h}$. The matching cost of $P_{i,h}$ is the summation of $w_{i,h}$ and $F_{i,h}$, which denote the *sibling cost* and the *history cost*, respectively.

5.3.2 Identifying the Siblings. A potential challenge is to identify the set of siblings. Fortunately, in LoRa, the following fact holds: *if a symbol or part of the symbol overlaps with both S_i and S_k and produces peaks at locations a and b, respectively,* $a = \mod\{b + \alpha_i - \alpha_k - 1, 2^{SF}\} + 1$. Therefore, it is possible to track a peak in all symbols where it may emerge and find its siblings based on the locations of the peaks.

5.3.3 Peak Cost Calculation. As shown in Fig. 5, a peak transmitted by node i in symbol S_i may also appear in $2(M-1)$ symbols, namely,

S_k and S_k^{+1} for $1 \le k < i$, and S_k^{-1} and S_k for $M \ge k > i$. Denote the maximum peak height of all siblings of $P_{i,h}$ as H^*. Note that in the signal vectors of some nodes, the sibling may be too weak to be identified as a peak; in this case, the height of the sibling is the value of the signal vector at the expected location of the sibling. The sibling cost of $P_{i,h}$ is

$$w_{i,h} = (1 - \frac{\eta_{i,h}}{H^*})^2. \quad (1)$$

To calculate the history cost, let A_i and D_i be the estimated peak height and peak height deviation when processing S_i, respectively. Thrives uses a curve-fitting algorithm [8] to fit the height of the peaks that have been observed for packet i so far. A_i is the value of the fitted curve at S_i^{-1}, and D_i is the median of the differences between the actual and fitted data. The *upper* and *lower estimates* are $U_i = A_i + 4D_i$ and $L_i = \max\{0, A_i - 4D_i\}$, respectively, which are shown in Fig. 6, along with the actual height of the peaks. The history cost of $P_{i,h}$ is:

$$F_{i,h} = \begin{cases} \omega(1 - \frac{U_i}{\eta_{i,h}})^2 & \text{if } \eta_{i,h} > U_i \\ 0, & \text{if } U_i \ge \eta_{i,h} \ge L_i \\ \omega(1 - \frac{\eta_{i,h}}{L_i})^2, & \text{otherwise} \end{cases}, \quad (2)$$

where ω is an empirical parameter to control the importance of the history cost. Currently, $\omega = 0.1$. As TnB decodes a packet for a second time if the first attempt was not successful, during the second attempt, the curve fitting algorithm runs on all peaks, while A_i is the value of the fitted curve at S_i, and D_i is the median of the differences between the actual and fitted data.

5.3.4 Peak Assignment.

Prior to the peak assignment, the known peaks and their siblings are found and masked, where a peak is known if it is in the preamble part of a packet, or if the packet has been decoded correctly. The peak assignment algorithm is a simple heuristic that determines the assignment of a selected symbol in each iteration. In each iteration, it first finds the minimum matching cost of the peaks in all remaining symbols. If there is only one symbol that has a peak with the minimum cost, this symbol is selected; otherwise, the symbol that has the fewest peaks with the minimum cost is selected; if there are still ties, an arbitrary choice is made. The selected symbol is assigned a minimum cost peak, after which the siblings of the peak are masked and the selected symbol removed.

5.3.5 Complexity.

At a checking point that intersects M symbols, Thrive needs to run the peak finder and the curve fitting algorithms for at most M times each, as well as running the peak assignment algorithm for at most M iterations, because each symbol is assigned a peak in each iteration and the decisions are never revisited. The peak cost needs to be calculated for at most $2M^2$ peaks, where the calculation for each peak involves finding its siblings and calculating the cost in constant time according to Eq. 1 and Eq. 2.

6 BLOCK ERROR CORRECTION (BEC)

BEC is an algorithm that decodes the same Hamming code in LoRa but achieves much higher error correction capabilities than the default decoder, as summarized in Table 1.

Table 1: Decoding Capability Comparison

CR	Default Decoder	BEC
1	Detects 1-bit error	Corrects 1-symbol error
2	Detects 1-bit error	Corrects 1-symbol error
3	Corrects 1-bit error	Corrects 1-symbol error and almost all 2-symbol errors
4	Corrects 1-bit error	Corrects 1 and 2-symbol errors and over 96% of 3-symbol errors

6.1 Core Ideas

BEC decodes code blocks in LoRa instead of individual codewords, because errors in a block are correlated: a corrupted symbol leads to errors in the same column of the block. BEC examines the differences between the received block and the cleaned block, because the differences are either the true errors, or are related to the true errors. In the following, the core ideas of BEC are explained for CR 3, because other CRs are similar.

Fig. 7 is an example continuing with Fig. 2, where the differences between the received block and the cleaned block are shown in red at the top of the figure. As mentioned earlier, due to the randomness of errors, a corrupted symbol rarely flips every bit in the corresponding column. Therefore, even when there are multiple corrupted symbols, there often exist rows with only one error, which is within the error correction capability of the Hamming Code and can be corrected by the default decoder. In Fig. 7, rows 2, 3, 4, 5, 6, and 8 have only one error and the differences between the received block and the cleaned block are either in column 2 or column 7, which are the true error columns.

When the number of errors is beyond the error correction capability of the Hamming code, the default decoder produces a decoding error, which is still mathematically related to the true errors. In Fig. 7, as row 7 has two errors, the default decoder flips the wrong bit in column 3. However, this is not a random action, because the default decoder will always flip the bit in column 3 if there are errors in columns 2 and 7. Therefore, column 3 is referred to as the *companion* of columns 2 and 7. Fundamentally, this is because a binary vector with '1's only in columns 2, 3 and 7 is a valid codeword. As a result, flipping the bit in column 3 of row 7 also produces a codeword, which is closer to row 7 than that by flipping the bits in columns 2 and 7.

As the differences between the received block and the cleaned block occur in columns 2, 3, and 7, BEC can determine that there must be 2 or more error columns, because otherwise, the default decoder is capable of correcting all errors and the difference shall occur all in the same column. With CR 3, BEC attempts to correct up to 2 error columns. The companion introduces ambiguities, because it is unclear which columns are the true error columns and which is the companion. Note that, as column 2 is the companion of columns 3 and 7, and column 7 is the companion of columns 2 and 3, the same situation can be observed if the true error columns are 2 and 3, or 3 and 7. To resolve this ambiguity, BEC generates 3 *BEC-fixed blocks* as potential solutions. Basically, BEC tests every combination of 2 potential error columns. As shown in Fig. 7, the BEC-fixed block for every combination is obtained by first masking the columns in the received block, then replacing each row with a codeword that matches in the remaining columns. BEC relies on the packet-level

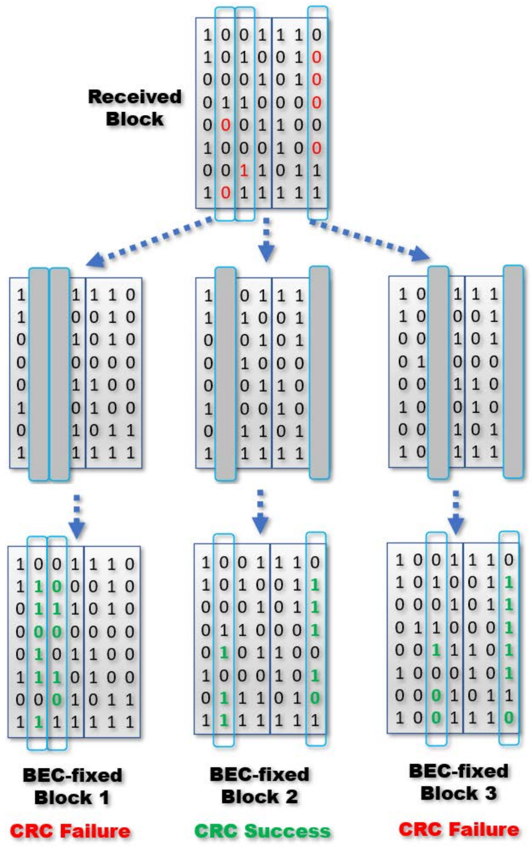

Figure 7: BEC corrects the errors in Fig. 2.

Cyclic Redundancy Check (CRC) to identify the correct BEC-fixed block, which should be the only one that leads to a CRC pass.

BEC applies only to packets with a small number of blocks, because the number of CRC calculations grows exponentially with the number of blocks. Fortunately, LoRa packets are typically small: a packet with 16 bytes has only 3 to 5 blocks depending on the SF and CR. Also, it could occur that the differences between the received block and cleaned block do not appear in a true error column. In almost all such cases, the differences occur in one true error column and the companion of the true error columns, so that BEC will still test the actual true error columns as one of the potential solutions and correct the errors.

6.2 Preliminaries and Definitions

The following are some of the notations used in BEC:

- R: the received block
- Γ: the cleaned block
- ϕ_i: the set of rows in which R and Γ differ by i bits
- Ξ: the set columns in which the rows in ϕ_1 differ between R and Γ
- r_i: row i of a matrix
- c_k: column k of a matrix
- Π: a set of columns
- $||$: the size of a set
- $V(\Pi)$: a binary row vector with the same length of a codeword, where a column is 1 if the column is in Π

In Fig. 2 and Fig. 7, $\phi_0 = \{r_1\}$, $\phi_1 = \{r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$, and $\Xi = \{c_2, c_3, c_7\}$.

For a set of columns Π where $|\Pi|$ is less than the minimum Hamming distance of the code, a *companion*, denoted as Π', is defined as the columns, which, when combined with Π, makes a codeword. That is, $V(\Pi \cup \Pi')$ is a codeword. For example, for the code shown in Fig. 2 and Fig. 7, the companion of $\{c_2, c_7\}$ is $\{c_3\}$. Clearly, $|\Pi| + |\Pi'| = CR$. Note that when there are errors in every column in Π, if $|\Pi| < |\Pi'|$, the default decoder flips the bits in Π; if $|\Pi| > |\Pi'|$, the default decoder flips the bits in Π' or in other companions of Π; if $|\Pi| = |\Pi'|$, the choice is arbitrary. Π may have one or multiple companions depending on the CR and $|\Pi|$. In particular, when the CR is 4 and $|\Pi| = 2$, as explained in Section A.1, Π has 3 possible companions. In this case, Π, along with its companions, are called a *companion group*.

6.3 Repair Methods

BEC employs a number of methods to *repair* R to produce the *BEC-fixed blocks*. There are a total of 4 repair methods, denoted as Δ', Δ_1, Δ_2, and Δ_3.

Δ' applies only to CR 1 and its notation style is slightly different, because CR 1 is a special case. To repair R with a column is to use the checksum of the other 4 columns to replace this column in R.

Δ_1 is used most often and has been shown in Fig. 7. To repair R with a set of columns, say, Π, BEC first masks these columns. A row in R, say, R_i, is *repairable*, if it matches one of the valid codewords, say, θ, in the remaining columns. In this case, to repair R_i is to replace it with θ. R is repairable only if every row is repairable.

Δ_2 applies mainly to CR 4 for correcting 2-column errors when Ξ contains one column, say, c_{k_1}. In this case, BEC assumes c_{k_1} is a true error column, and attempts to repair the rows in ϕ_2. A row in ϕ_2, say, R_i, is repairable, if it differs only in one column, say, c_{k_2}, with a valid codeword, say, θ, after the bit in c_{k_1} is flipped. In this case, to repair R_i is to replace it with θ. c_{k_2} is called the *column of mismatch*. R is repairable only if all rows in ϕ_2 are repairable with the same column of mismatch.

Δ_3 applies only to CR 4 for correcting 2-column errors when Ξ is empty. BEC attempts to repair R with two columns, say c_{k_1} and c_{k_2}. A row in ϕ_2, say R_i, is repairable, if it matches a codeword, say, θ, after the bits in c_{k_1} and c_{k_2} are flipped. In this case, to repair R_i is to replace it with θ. R is repairable only if all rows in ϕ_2 are repairable.

The complexity of Δ' is clearly low because it involves only the calculation of the checksum. Owing to the simplicity of the (8, 4) Hamming code, the complexities of the rest of the repair methods are also low, because the main computation is to compare each modified row with all 16 codewords, where the total number of comparisons is bounded by $16SF$.

6.4 Decoding CR 1

With CR 1, BEC attempts to correct up to 1-column errors. If the parity check passes in every row, BEC returns, assuming there is no error. Otherwise, BEC attempts to repair R with each column according to Δ', and produces 5 BEC-fixed blocks, as there are 5 columns in R,

6.5 Decoding CR 2

With CR 2, a row in R and the corresponding row in Γ differ by at most one bit. BEC first examines if $|\Xi| = 0$, i.e., R and Γ are identical, and if so, BEC returns, assuming there is no error. If $|\Xi| \geq 1$, BEC attempts to decode 1-column errors. As explained in Section A.2, if $|\Xi| \geq 3$, there must be more than one error column and BEC returns with decoding failure. If $|\Xi| = 1$, BEC first finds the companion of the column in Ξ and adds it to Ξ, then applies Δ_1 to repair R with each column in Ξ, producing a BEC-fixed block for each if the repair is successful. The same process is applied if $|\Xi|$ is already 2.

6.6 Decoding CR 3

With CR 3, a row in R and the corresponding row in Γ differ by at most one bit. BEC first examines if $|\Xi| = 0$, and if so, BEC returns, assuming there is no error. Otherwise, BEC examines if $|\Xi| = 1$, i.e., the differences between R and Γ all occur in a single column, and if so, BEC returns, assuming there is only one error column, because the default decoder can correct one-bit errors. If $|\Xi| \geq 2$, BEC attempts to decode 2-column errors. As explained in Section A.2, if $|\Xi| \geq 4$, there must be more than two error columns and BEC returns with decoding failure. If $|\Xi| = 2$, BEC finds the companion of the two columns in Ξ, which is another column, and adds it to Ξ. BEC attempts all 3 combinations of two columns in Ξ to repair R with Δ_1, producing a BEC-fixed block in each case if the repair is successful. The same process is applied if $|\Xi|$ is already 3, as shown in Fig. 7.

6.7 Decoding CR 4

For CR 4, a row in R and the corresponding row in Γ differ by at most two bits. Similar to CR 3, BEC returns without further processing if R and Γ are identical, or if the differences between R and Γ all occur in a single column. Otherwise, BEC first attempts to decode 2-column errors, and, if fails, 3-column errors.

6.7.1 2-Column Errors. If there are 2 error columns, as explained in Section A.2, $|\Xi| \leq 2$. Therefore, BEC attempts to decode 2-column errors only if $|\Xi| \leq 2$. First, if $|\Xi| = 0$, which is very rare, for every row in ϕ_2, say row i, where R_i and Γ_i differ in two columns, BEC finds the companion group of the two columns, which contains 4 pairs. If every row in ϕ_2 yields exactly the same companion group, BEC produces a BEC-fixed block for every pair in the group by using the pair to repair R with Δ_3. If $|\Xi| = 1$, let the column in Ξ be c_k. BEC attempts to repair R with Δ_2 using c_k and produces one BEC-fixed block if the repair is successful. If $|\Xi| = 2$, BEC repairs R with Δ_1 using the two columns and produces one BEC-fixed block if the repair is successful. The attempt to decode 2-column errors fails if no BEC-fixed block is produced.

6.7.2 3-Column Errors. If there are 3 error columns, as explained in Section A.2, $|\Xi| \leq 4$. Therefore, BEC attempts to decode 3-column errors only if $|\Xi| \leq 4$. If $|\Xi| = 0$, however, BEC returns with decoding failure, because it is beyond the capability of BEC.

If $|\Xi| = 1$, suppose the column in Ξ is c_{k_1}. BEC applies Δ_2 using c_{k_1}. If there are indeed 3 error columns, based on Lemma 3 in Section A.7, there must be either 2 or 3 distinct columns of mismatch for rows in ϕ_2 after the repair. In the former case, denote the columns as c_{k_2} and c_{k_3}. BEC finds the companion of c_{k_1}, c_{k_2}, and c_{k_3}, denoted as c'. In the latter case, denote the columns as c_{k_2}, c_{k_3}, and c_{k_4}, and based on Lemma 3, c_{k_4} must be the companion of c_{k_1}, c_{k_2}, and c_{k_3}; therefore, the two cases are equivalent. BEC attempts to repair R with Δ_1 using all 4 combinations of 3 columns and produces a BEC-fixed block in each case.

If $|\Xi| = 2$, BEC first makes 6 attempts to repair R with Δ_1, where in each attempt it uses Ξ along with a column not in Ξ. If there are indeed 3 error columns, based on Lemma 1 and Lemma 2 in Section A.3, among the 6 attempts, regardless of whether or not the two columns in Ξ are the true error columns or not, there will be exactly 2 attempts that can repair R. Denote the two combinations of columns that can repair R as $(c_{k_1}, c_{k_2}, c_{k_3})$ and $(c_{k_1}, c_{k_2}, c_{k_4})$, respectively, where c_{k_1} and c_{k_2} are in Ξ. Note that if Ξ contains two true error columns, one of the two combinations are the true error columns; otherwise, i.e., if Ξ contains the companion of the true error columns, either $(c_{k_3}, c_{k_4}, c_{k_1})$ or $(c_{k_3}, c_{k_4}, c_{k_2})$ must be the true error columns. Therefore, BEC makes two more attempts with Δ_1 using $(c_{k_3}, c_{k_4}, c_{k_1})$ and $(c_{k_3}, c_{k_4}, c_{k_2})$, and produces a total of 4 BEC-fixed blocks.

If $|\Xi| = 3$, BEC adds the companion of Ξ to Ξ, which is another column. BEC then attempts to repair R with Δ_1 using each combination of 3 columns in Ξ. If there are indeed 3 error columns, one BEC-fixed block is produced in each attempt. The same repair process is applied if $|\Xi|$ is already 4.

Note that, if there are more than 3 error columns, the repair may fail. Even if the repair appears successful, the packet-level CRC will still eventually fail.

Table 2: Summary of BEC

CR	# of Err. Columns	Approx. Err. Prob.	Repair Complexity	# of CRC
1	1	0	$5\ \Delta'$	5
2	1	0	$2\ \Delta_1$	2
3	1	0	NA	NA
	2	2^{-SF}	$3\ \Delta_1$	3
4	1	0	NA	NA
	2	0	$\leq 4\Delta_3$	≤ 4
	3	≤ 0.04	$\leq 9\Delta_1$	4

6.8 Decoding Performance and Complexity

Table 2 summaries the decoding performance of BEC, the proof of which can be found in the Appendix. Table 2 also shows the complexity of BEC for decoding one block, where the computation mainly include applying the repair methods and CRC calculations. The number of CRC calculations for a block is exactly the number of BEC-fixed blocks. The type of the repair method and the number of times it is applied depend on the CR and the number of error columns, which, in most cases, should be clear from the description of the decoding process. With CR 4 and 3 error columns, the bound is $9\Delta_1$, because the highest computation occurs when $|\Xi| = 2$, in which case BEC first assumes that there are two error columns and applies Δ_1 once, which will fail, then applies Δ_1 8 times, assuming there are 3 error columns.

6.9 Packet Decoding

A packet is decoded by assembling the BEC-fixed blocks of different blocks into a *repaired packet* and testing the packet level CRC. To limit the computation complexity, the number of CRC calculations is limited by a parameter denoted as W. If the potential number of repaired packets exceeds W, only W packets are randomly selected and tested. W is currently 125, 16, 16, and 16 when the CR is 1, 2, 3, and 4, respectively. W is higher when the CR is 1, because more BEC-fixed blocks are generated, a price for transmitting less overhead. However, it was found that when the CR is 1, changing W to 25 reduces the number of decoded packets by less than 5%.

7 PACKET DETECTION

In TnB, packet detection consists of 4 steps, where the main innovation is the last step for the estimation of the fractional symbol boundary and CFO.

In step 1, TnB detects packets with a similar approach as that in [20] by finding peaks at the same locations in consecutive symbols, which should be generated by the upchirps and the downchirps in the preamble.

In step 2, a preliminary estimate of the start time of a detected preamble is calculated and the preamble is removed if it does not produce peaks at expected locations. For example, if the preamble is detected with the downchirp, the preliminary start time should be selected such that the downchirp peaks are at location 1. Under this constraint, the upchirp peaks should be close to location 1, where the relaxation is determined by the maximum allowable CFO, because the CFO has not been corrected at this point. To avoid errors that are multiples of T, where T is the length of a symbol, multiple tests are performed with adjustments of $-2T$, $-T$, 0, T, and $2T$ added to the preliminary estimate of the start time. A preamble is discarded only if it fails all 5 tests.

In step 3, coarse timing and CFO estimations are calculated for each detected preamble according to [25]. That is, let x_1 and x_2 be the locations of the upchirp peaks and the downchirp peaks, respectively. The preamble start time and the CFO are adjusted by $\tau\lfloor(x_1 - x_2)/2\rceil$ and $f\lfloor(x_1 + x_2)/2\rceil$, respectively, where $\lfloor\rceil$ denotes rounding a number to the nearest integer, τ is the sample time, and $f = 1/T$.

In step 4, fractional timing and CFO are estimated, which are fractions of τ and f, respectively. For simplicity, adjustments of $\delta_t\tau$ and $\delta_f f$ are denoted as δ_t and δ_f, respectively, where δ_t and δ_f are real numbers. Due to challenges caused by collisions, TnB uses a search that evaluates a function for different combinations of δ_t and δ_f and selects the combination that achieves the maximum. The search is optimized and evaluates only 36 combinations when $U = 8$, where U denotes the Over-Sampling Factor (OSF), which is the number of samples taken at the receiver between two transmitted samples at the sender.

To be more specific, for a received symbol β, let the *complex signal vector* be $FFT(\beta \odot C')$. For any δ_f and δ_t, let $Q(\delta_t, \delta_f)$ be the total peak energy in the preamble, where the energy is computed by adding the complex signal vectors of the preamble and computing the energy at the peak location in the summation vector. The complex signal is used because it preserves the phase information, so that the summation at the peak location is weak if the fractional CFO is not canceled. Let $Q^*(\delta_t, \delta_f)$ be $Q(\delta_t, \delta_f)$ if both the upchirp peaks and downchirp peaks are at location 1; otherwise, let $Q^*(\delta_t, \delta_f)$ be 0.

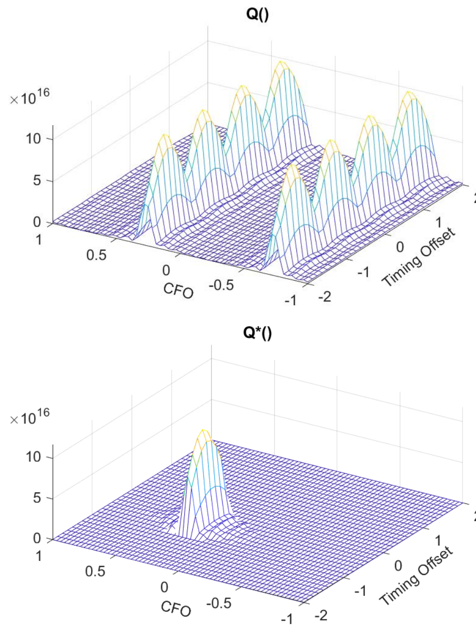

Figure 8: $Q()$ and $Q^*()$ of a packet transmitted by a commodity LoRa device [3].

The search consists of 3 phases. In Phase 1, the search evaluates 17 points along a line where $\delta_t = 0$ and δ_f is from -1 to 0 at a step of $f/16$. Suppose $Q()$ achieves the maximum at $(0, \delta_f^*)$. In Phase 2, the search evaluates a total of 10 points along two lines. On both lines, δ_t is from -1 to 1 at a step of $1/2$. On one line, $\delta_f = \delta_f^*$; on the other line, $\delta_f = \delta_f^* + 1$. Suppose $Q^*()$ achieves the maximum at $(\hat{\delta_t}, \hat{\delta_f})$. In Phase 3, the search evaluates $U + 1$ points along a line where δ_t is from $\hat{\delta_t} - 1/2$ to $\hat{\delta_t} + 1/2$ at a step of $1/U$ and $\delta_f = \hat{\delta_f}$. Suppose $Q^*()$ achieves the maximum at $(\tilde{\delta_t}, \tilde{\delta_f})$. $\tilde{\delta_t}$ and $\tilde{\delta_f}$ are used as the estimated fractional timing and CFO, respectively.

The search is based on the fact that when the timing is accurate and the CFO has been fully canceled, the total energy of the peaks is the highest and the upchirp peaks and downchirp peaks are all at location 1. By exploiting the nature of $Q()$, the computation complexity is significantly reduced compared to a naive approach that may evaluate all possible combinations of δ_f and δ_t. The top of Fig. 8 shows $Q()$ of a packet transmitted by a commodity LoRa device [3], where it can be seen that along any line where δ_t is fixed, $Q()$ achieves high values when δ_f is correct, or when δ_f is off by ± 1. Therefore, in Phase 1, the search simply is along the line where $\delta_t = 0$, which will find either the correct fractional CFO, or off by ± 1. In Phase 2, the correct fractional CFO is found by evaluating $Q^*()$, because the peaks will not be at location 1 if the fractional CFO is off by 1 or -1, as can be seen at the bottom of Fig. 8. In Phase 3, the search is along the line with the correct fractional CFO to pick the best fractional timing offset.

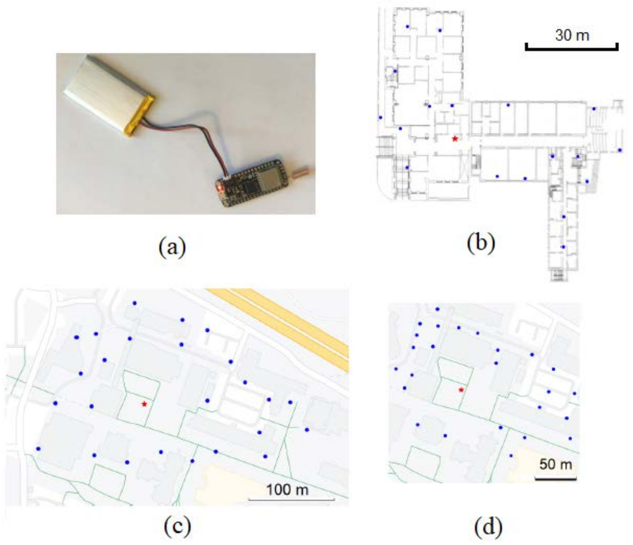

Figure 9: (a) The LoRa node. (b). Indoor. (c). Outdoor 1. (d). Outdoor 2.

Table 3: Experiment Parameters

Carrier Frequency	915 MHz
Bandwidth	125 kHz
Over-Sampling Factor (OSF)	8
Packet Size	16 bytes
Duration	30 seconds
SF	8, 10
CR	1, 2, 3, 4

8 EVALUATIONS

TnB has been implemented and compared with the state-of-the-art. The implementation can be found at [5] and is capable of detecting and decoding packets transmitted by commodity LoRa devices. Part of the implementation related to specific operations in LoRa, such as whitening, CRC calculation, etc., are based on the open-source LoRa implementations at [6, 10, 14].

8.1 Experiment Setup

The LoRa nodes used in the experiment are the Adafruit Feather M0 with RFM95 LoRa Radio 900 MHz [3], one of which is shown in Fig. 9(a). One of the nodes acts as the *starter*. At the beginning of an experiment, the starter transmits a *start message* 3 times to inform the nodes about the configuration of the experiment, including the SF, the CR, the start time of the experiment, the number of packets to be transmitted by each node, and the duration of the experiment. After receiving the start messages, a node transmits packets at randomly selected times during the experiment. A USRP B210 [9] is placed next to the starter to record the samples, which are written to a trace file. Some common parameters in all experiments are shown in Table 3. The packet transmitted by each node has 16 bytes of payload, which includes 4 bytes of header, 10 bytes of data, and 2 bytes of CRC. A node ID and a sequence number have been added to the data part of the packet to distinguish the packets.

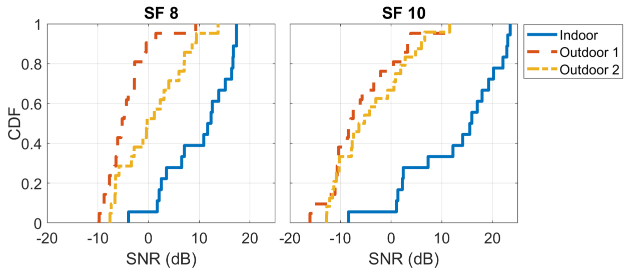

Figure 10: Estimated SNR of the 3 deployments.

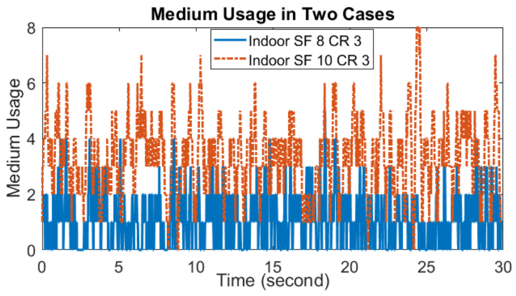

Figure 11: Lower bound of the medium usage in two cases at the highest load.

Three deployments have been tested, which are referred to as Indoor, Outdoor 1 and Outdoor 2, with 19, 25, and 25 nodes, respectively. The layout of the deployments are shown in Fig.9(b), Fig.9(c), and Fig.9(d), respectively, where the star is the location of the starter and the USRP sniffer. In each deployment, it was confirmed that each node could communicate with starter. SF 8 and SF 10 were tested for all CR values. For each combination of SF and CR, 5 network traffic load values were tested from 5 pkt/sec to 25 pkt/sec at a step of 5 pkt/sec. For each traffic load, the experiment was repeated 3 times, called three *runs*. A total of 360 trace files were collected.

Fig. 10 shows the Cumulative Distribution Function (CDF) of the estimated Signal to Noise Ratio (SNR) of the nodes in various deployments. To be more specific, for each deployment and SF, the run with the most number of nodes that have decoded packets is selected. The SNR of a node is estimated based on the peak heights found in its decoded packets. The SNR estimations are different between SF 8 and SF 10, primarily because more weak packets can be decoded with SF 10. The SNR of the same node can also vary, such as by over 5 dB, in one run. Still, it can be seen that the SNR varies in different deployments, and, within the same deployment, the SNRs of the nodes may also differ by more than 20 dB.

Fig. 11 shows the *medium usage* in 2 typical runs at the highest load, where the medium usage of a particular time instant refers to the number of packets on the air at the time and reveals the traffic condition. As the traffic was randomly generated and not all packets were received correctly, the medium usage is a random variable that cannot be known exactly. Fig. 11 shows a lower bound obtained by considering only packets that were correctly decoded by TnB, which is a subset of all transmitted packets. It can be seen that the medium can be very busy both for SF 8 and SF 10, and is more so for SF 10, because the packet is longer with SF 10.

8.2 Compared Schemes

TnB is compared with CIC [20] and AlignTrack [12], which are recent LoRa packet collision resolution schemes, as well as LoRaPHY [6], which is an implementation of the original LoRa packet decoder. CIC and LoRaPHY were tested with their open-source implementations graciously shared by their authors at [4, 6]. The testing of AlignTrack poses some challenges, as an open-source implementation was not available. AlignTrack mainly consists of a peak detection algorithm, a packet detection algorithm, and a peak assignment algorithm, which are described in Sections IV.C, IV.D, and IV.E in [12], respectively. In this paper, the comparison focuses on the peak assignment algorithm, denoted as AlignTrack*, because it is the core and main innovation of AlignTrack. AlignTrack* has been implemented and can swap out Thrive as a component of TnB to be tested, because it solves the same problem as Thrive. It should be mentioned that the peak detection algorithm in TnB is a highly rated open-source peak finder [29]. The packet detection algorithm in TnB allows TnB to outperform CIC; in addition, it also lends the benefit of the fractional CFO information to AlignTrack, because AlignTrack estimates only the coarse CFO. Each scheme was tested with exactly the same traces collected in the experiments. As CIC and AlignTrack* find only the peak locations of the packets, their outputs were decoded by the open-source LoRa implementation [6, 10] into data bits.

8.3 Results

The results are shown in Fig. 12, Fig. 13, Fig. 14, for Indoor, Outdoor 1, and Outdoor 2, respectively. In most cases, a data point is the average of 3 runs. In some cases, however, the number of nodes that responded to the start messages are significantly smaller, which leads to biased results. A simple rule is applied to filter such cases; that is, the result of a run is used, if the number of nodes with decoded packets is at least half of the maximum number of nodes with decoded packets in the same deployment. Only two cases were found with no valid data after the filtering, namely, in Outdoor 2 for the highest load when the SF is 10 and the CR is 1 and 4. In other cases, because some nodes might not have responded, the traffic load values shown in the figures, which assume all nodes responded in the experiment, is higher than the actual traffic load. The comparison is still fair because all schemes process exactly the same traces.

It can be seen that, first, TnB achieves much higher throughput than the compared schemes. At the highest tested load, the median throughput increase of TnB over CIC among all CR values in all experiments are 1.36× and 2.46× for SF 8 and 10, respectively. The highest improvement is 2.59× for the Outdoor 1 deployment with SF 10 and CR 3. Second, the gain of TnB over CIC is significantly higher for SF 10 than SF 8, because the packet duration is longer with SF 10, resulting in more collisions. For the same reason, LoRaPHY still decoded a descent amount of packets for SF 8, but not SF 10. Lastly, the performance of AlignTrack* is similar to CIC for the two outdoor cases with SF 8, but is much lower with SF 10, which will be further investigated in Section 8.4.

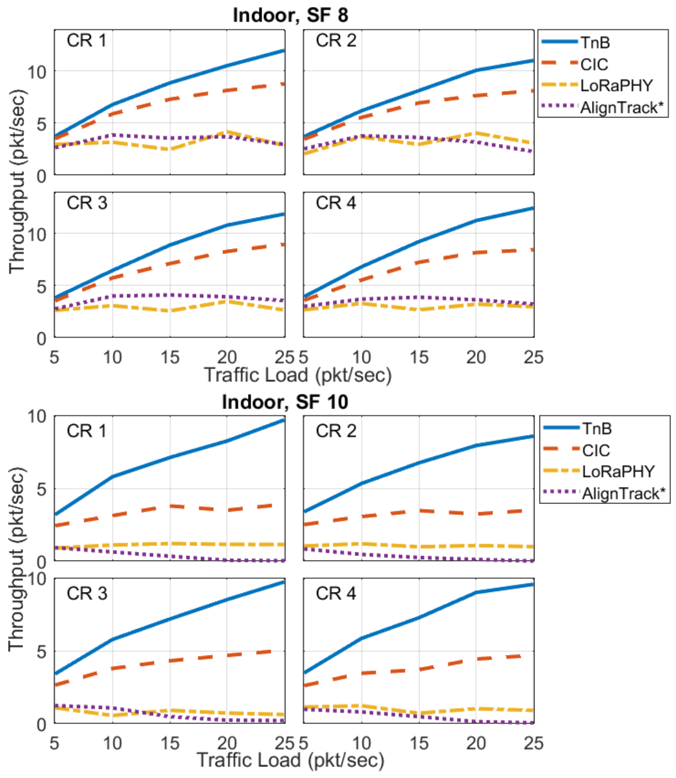

Figure 12: Results of the Indoor deployment.

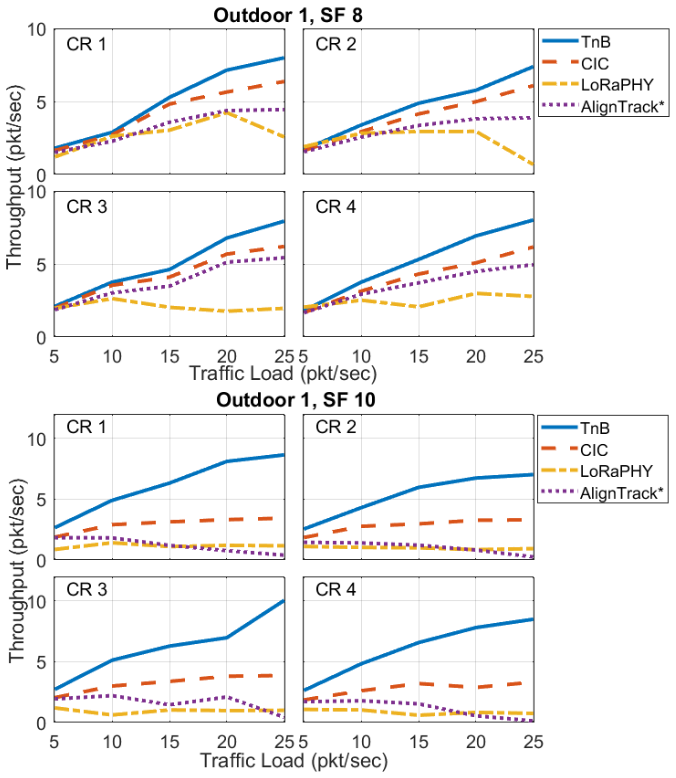

Figure 13: Results of the Outdoor 1 deployment.

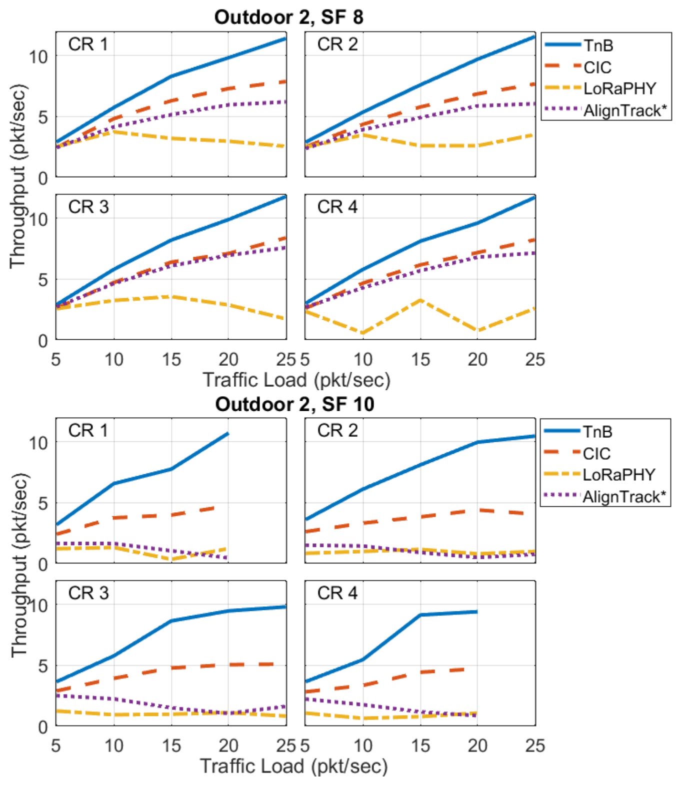

Figure 14: Results of the Outdoor 2 deployment.

8.4 Further Analysis

Further analysis is conducted with the results from experiments at the highest load with valid data in each deployment. First, to understand the source of the gains, TnB is tested with various configurations. To be more specific, *Thrive* refers to TnB without BEC and uses the default Hamming code decoder, and *Sibling* refers to Thrive without the history cost and relying only on the sibling cost. The performances of these configurations are shown in Fig. 15, along with those of CIC for comparison. It can be seen that Thrive is similar to CIC for SF 8, but outperforms CIC for SF 10, suggesting that Thrive is an effective peak assignment algorithm. The median improvement of TnB over Thrive is 1.31×, confirming the contribution of BEC. Sibling does not perform well in certain cases, revealing the importance of the peak history information.

Fig. 16 reveals more details about BEC, which shows the CDF of the number of *BEC rescued codewords* in each decoded packet, which are those decoded by BEC but not decoded correctly by the default decoder. The fraction of packets rescued by BEC can be in the figure, which is the fraction of packets with at least one BEC rescued codeword. In such packets, it can be seen that there can be multiple rescued codewords.

Fig. 17 shows the scatter plots of the Packet Receiving Ratio (PRR) of TnB and CIC in different ranges of SNR, where each marker represents one combination of SF and CR in a deployment within a particular SNR range. It can be seen that higher SNR leads to higher PRRs. Except a few cases, TnB achieves higher PRRs than CIC in all ranges of SNR.

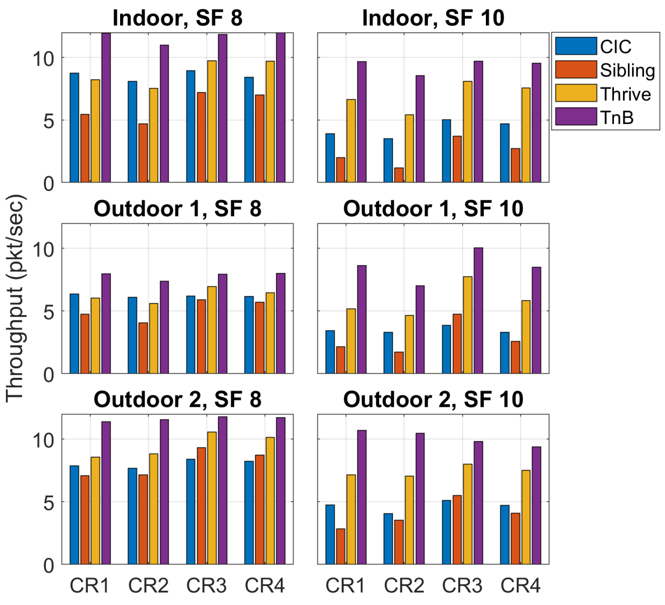

Figure 15: Evaluating the components of TnB.

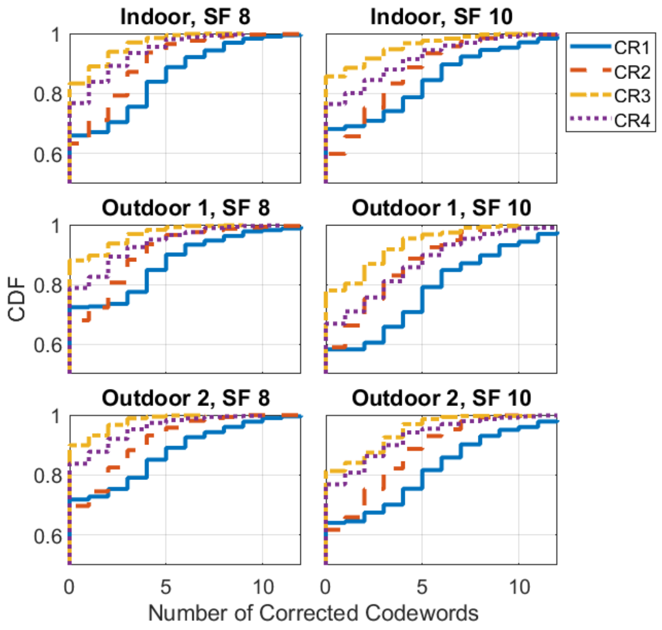

Figure 16: Number of codewords rescued by BEC in a packet.

Fig. 18 shows the percentage of packets decoded by TnB at different *collision levels*, where the collision level of a packet is defined as the highest number of packets it collided with during its transmission. Similar to the medium usage shown in Fig. 11, the collision level is estimated by considering only the packets decoded by TnB, and is therefore a lower bound. It can be seen that less than 15% of decoded packets with SF 8 had no collision, while others typically collide with at least 1 or 2 packets. The majority of packets decoded with SF 10 collided with 4 or more packets.

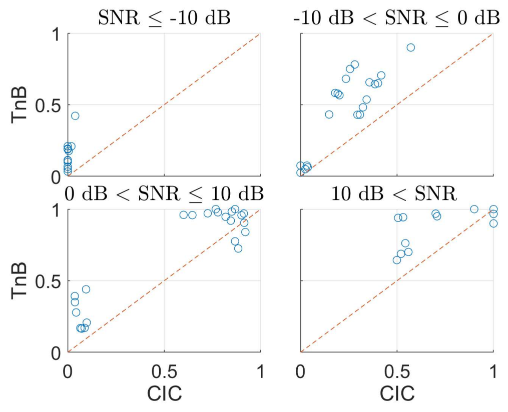

Figure 17: Scatter plots of the PRR at various SNR ranges.

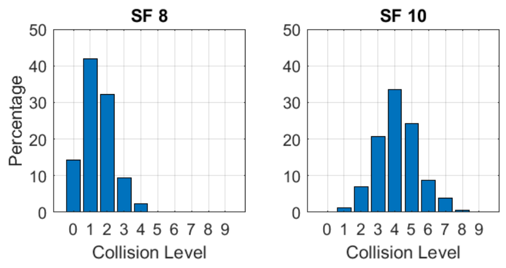

Figure 18: Collision levels of packets decoded by TnB.

Lastly, AlignTrack* did not perform well for SF 10 because it is very sensitive to peaks that may occur accidentally, which are found more with SF 10 than with SF 8. Such accidental peaks may be generated by noise, interference, etc. Typically, they can be seen in one signal vector but not in others, which poses a problem, because AlignTrack* considers a peak to be aligned to a symbol if it is higher in this symbol than in other symbols. As a result, once there is an accidental peak, AlignTrack* finds more than one peak aligned with a symbol, among which one is the correct peak and others the accidental peaks. In such cases, an arbitrary and often incorrect decision has to be made.

8.5 Comparisons with Simulation

Additional evaluation was conducted with simulations, which was intended mainly to examine more challenging channel conditions, i.e., the LTE ETU model [1, 2], which represents channels with strong multi-path and fluctuations, because strong channel fluctuations were not observed in the experiments due to the lack of mobility in the environment. The simulation setup is mostly identical to the experiments, with the same bandwidth, OSF, packet size, and duration. The SNR ranges of SF 8 and SF 10 are $[0, 20]$ dB and $[-6, 14]$ dB, respectively. The CFO of a packet is randomly selected from $[-4.88, 4.88]$ kHz. The delay spread of the ETU channel is 5 μs and the Doppler shift is 5 Hz.

The compared schemes are CIC, AlignTrack*, and TnB. CIC and AlignTrack* are also combined with BEC, denoted as CIC+ and AlignTrack*+, respectively, to test BEC when combined with other methods. Thrive and TnB2ant are also tested, where Thrive has been defined in Section 8.4, and TnB2ant is when the TnB receiver has 2

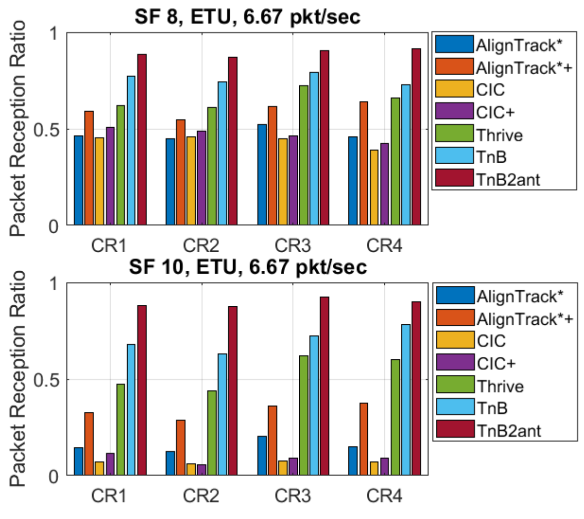

Figure 19: Simulation results in the ETU channel.

antennas, because high channel fluctuations result in a high outage probability for single antenna systems, which can be mitigated by a higher antenna diversity.

The results are shown in Fig. 19, where the traffic load is selected such that the PRR of TnB2ant of at least one CR is above 0.9. It can be observed that TnB2ant achieves PRRs close to or above 0.9 in the ETU channels. TnB, as well as Thrive, achieves much higher gains over CIC in the ETU channels than in the experiments, especially for SF 10, suggesting that CIC may need further enhancements to cope with large channel fluctuations. It can also be seen that BEC can be combined with CIC and AlignTrack* and always improve the performance. AlignTrack* sometimes achieves better performance than CIC in the simulation, an improvement since the experimental evaluation, likely because the interference in the experimental traces leads to more accidental peaks.

9 CONCLUSIONS

In this paper, TnB, a novel solution for decoding collided LoRa packets, is proposed. Two main components of TnB are referred to as Thrive and BEC, respectively. Thrive finds the actual transmitters of the peaks in the received signal vectors, and is mainly based on the fact that a peak is the highest if it is processed with the parameters of its actual transmitter, which is further augmented with the peak history information. BEC decodes exactly the same Hamming code in LoRa, but is capable of correcting much more errors than the default decoder, because its jointly decodes multiple codewords in the same block. TnB has been extensively tested with both real-world experimental traces and simulations, and the results show that TnB significantly outperforms the state-of-the-art. TnB does not require any modifications to the LoRa nodes and can bring immediate benefits to the network operator.

ACKNOWLEDGMENTS

We greatly appreciate our shepherd Zicheng Chi and the reviewers of our paper for their valuable feedback and guidance. This research work was supported by the US National Science Foundation under Grant 1910268.

REFERENCES

[1] 3GPP TS 36.101. User Equipment (UE) Radio Transmission and Reception. 3rd Generation Partnership Project; Technical Specification Group Radio Access Network. Evolved Universal Terrestrial Radio Access (E-UTRA).

[2] 3GPP TS 36.104. Base Station (BS) radio transmission and reception. 3rd Generation Partnership Project; Technical Specification Group Radio Access Network. Evolved Universal Terrestrial Radio Access (E-UTRA).

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

[4] Concurrent Interference Cancellation (CIC) Implementation. https://github.com/osama4933/CIC.

[5] Implementation of TnB. https://github.com/raghavrathi10/TnB.

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

[7] LoRaWAN 1.1 specification. https://www.lora-alliance.org/ resource-hub/lorawantm-specification-v11.

[8] smoothdata: Smooth noisy data. *MATLAB*.

[9] USRP B210. https://www.ettus.com/all-products/ub210-kit/.

[10] Bassel Al Homssi, Kosta Dakic, Simon Maselli, Hans Wolf, Sithamparanathan Kandeepan, and Akram Al-Hourani. Iot network design using open-source lora coverage emulator. *IEEE Access*, 9:53636–53646, 2021.

[11] Artur Balanuta, Nuno Pereira, Swarun Kumar, and Anthony Rowe. A cloud-optimized link layer for low-power wide-area networks. In *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 247–259. ACM, 2020.

[12] 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, 2021.

[13] Ulysse Coutaud, Martin Heusse, and Bernard Tourancheau. Fragmentation and forward error correction for lorawan small MTU networks. In Christine Julien, Fabrice Valois, Omprakash Gnawali, and Amy L. Murphy, editors, *Proceedings of the 2020 International Conference on Embedded Wireless Systems and Networks, EWSN 2020, Lyon, France, February 17-18, 2020*, pages 289–294. ACM, 2020.

[14] Daniele Croce, Michele Gucciardo, Stefano Mangione, Giuseppe Santaromita, and Ilenia Tinnirello. Impact of lora imperfect orthogonality: Analysis of link-level performance. *IEEE Communications Letters*, 22(4):796–799, 2018.

[15] Rashad Eletreby, Diana Zhang, Swarun Kumar, and Osman Yagan. Empowering low-power wide area networks in urban settings. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 309–321. ACM, 2017.

[16] Tallal Elshabrawy and Joerg Robert. Evaluation of the BER performance of LoRa communication using BICM decoding. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 162–167, 2019.

[17] Sondos Elzeiny, Phoebe Edward, and Tallal Elshabrawy. LoRa performance enhancement through list decoding technique. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6, 2021.

[18] Bin Hu, Zhimeng Yin, Shuai Wang, Zhuqing Xu, and Tian He. SCLoRa: Leveraging multi-dimensionality in decoding collided LoRa transmissions. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–11, 2020.

[19] Paul J. Marcelis, Nikolaos Kouvelas, Vijay S. Rao, and R. Venkatesha Prasad. Dare: Data recovery through application layer coding for lorawan. *IEEE Transactions on Mobile Computing*, 21(3):895–910, 2022.

[20] Muhammad Osama Shahid, Millan Philipose, Krishna Chintalapudi, Suman Banerjee, and Bhuvana Krishnaswamy. Concurrent interference cancellation: decoding multi-packet collisions in LoRa. In Fernando A. Kuipers and Matthew C. Caesar, editors, *ACM SIGCOMM 2021 Conference, Virtual Event, USA, August 23-27, 2021*, pages 503–515. ACM, 2021.

[21] Chenglong Shao, Osamu Muta, Wenjie Wang, and Wonjun Lee. Toward ubiquitous connectivity via LoRaWAN: An overview of signal collision resolving solutions. *IEEE Internet of Things Magazine*, 4(4):114–119, 2021.

[22] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in LPWANs. In *MobiSys '20: The 18th Annual International Conference on Mobile Systems, Applications, and Services, Toronto, Ontario, Canada, June 15-19, 2020*, pages 234–246. ACM, 2020.

[23] Shuai Tong, Zhenqiang Xu, and Jiliang Wang. CoLoRa: Enabling multi-packet reception in LoRa. In *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*, pages 2303–2311. IEEE, 2020.

[24] 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, 2019.

[25] Mathieu Xhonneux, Orion Afisiadis, David Bol, and Jérôme Louveaux. A low-complexity LoRa synchronization algorithm robust to sampling time offsets. *IEEE Internet of Things Journal*, 9(5):3756–3769, 2022.

[26] Xianjin Xia, Ningning Hou, and Yuanqing Zheng. PCube: Scaling LoRa concurrent transmissions with reception diversities. In *ACM Mobicom*, 2021.

[27] Xianjin Xia, Yuanqing Zheng, and Tao Gu. FTrack: Parallel decoding for LoRa transmissions. *IEEE/ACM Transactions on Networking*, 28(6):2573–2586, 2020.

[28] 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*, pages 1–9, 2021.

[29] Nathanael Yoder. peakfinder(x0, sel, thresh, extrema, includeendpoints, interpolate). *MATLAB Central File Exchange*, (https://www.mathworks.com/matlabcentral/fileexchange/25500-peakfinder-x0-sel-thresh-extrema-includeendpoints-interpolate), Retrieved March 16, 2022.

A THEORETICAL FOUNDATION AND ANALYSIS OF BEC

A.1 Properties of Companions

The companion is a key concept for CR 2 or above. For CR 2, the companion of a column is another column and is unique. It can be verified that the companion pairs are c_1 and c_5, c_2 and c_3, c_4 and c_6.

For CR3, if $|\Pi| = 2$, the companion of Π contains only one column and is unique. To see this, suppose Π consists of columns c_{k_1} and c_{k_2}, and has two companions, namely, $c_{k'}$ and $c_{k''}$. Consider vectors $V(c_{k_1}, c_{k_2}, c_{k'})$ and $V(c_{k_1}, c_{k_2}, c_{k''})$, both must be codewords but with Hamming distance only two, a contradiction, as the minimum Hamming distance is 3.

For CR 4, if $|\Pi| = 3$, the companion of Π contains one column and is unique. As mentioned earlier, if $|\Pi| = 2$, Π has 3 possible companions. For example, the companions of $\{c_1, c_2\}$ are $\{c_6, c_8\}$, $\{c_3, c_5\}$, and $\{c_4, c_7\}$, because there are 3 codewords with weight 4, namely, [11000101], [11101000], and [11010010], where bits 1 and 2 are both '1'. The same argument can be applied to other pairs of columns.

A.2 Properties of Ξ

Except for the special and simple case of CR 1, the decoding process depends on Ξ. This is because a column in Ξ is either a true error column, or the companion of the true error columns.

For CR 2, note that if there is a single error column, say, c_k, the companion of c_k is another column denoted as $c_{k'}$. And, if a row has an error in c_k, the default decoder flips either the bit in c_k or in $c_{k'}$. Therefore, $|\Xi|$ is either 1 or 2.

For CR 3, consider the case with two error columns, denoted as c_{k_1} and c_{k_2}, and suppose their companion is $c_{k'}$. In any row i, there are 4 possible cases:

(1) No error in both c_{k_1} and c_{k_2}: R_i and Γ_i are identical
(2) No error in c_{k_1} but error in c_{k_2}: R_i and Γ_i differ in c_{k_2}
(3) Error in c_{k_1} but no error in c_{k_2}: R_i and Γ_i differ in c_{k_1}
(4) Errors in both c_{k_1} and c_{k_2}: R_i and Γ_i differ in $c_{k'}$

Therefore, with two error columns, $|\Xi| \leq 3$.

For CR 4, the case with two error columns, say, c_{k_1} and c_{k_2}, can be analyzed similarly as that for CR 3, except when there are errors in both c_{k_1} and c_{k_2}, because R_i and Γ_i shall differ in c_{k_1} and c_{k_2}, or in the columns of one of their companions, and therefore, no column will be added to Ξ and $|\Xi| \leq 2$. For CR 4, when there are three error columns, $|\Xi| \leq 4$, because the companion of any 3 error columns is one column.

A.3 Properties of Δ_1

Clearly, Δ_1 shall succeed with Π if Π is the set of true error columns. In the following, Δ_1 in other cases are analyzed, in which Ω denotes the set of true error columns and $\Omega = \{c_{k_1}, c_{k_2}, \ldots, c_{k_n}\}$.

Lemma 1. *If the companion of Ω is a column denoted as $c_{k'}$, Δ_1 shall succeed if Π consists of all but one true error column, as well as $c_{k'}$.*

Proof. Without Loss Of Generality (WLOG), suppose R is repaired with $\{c_{k'}, c_{k_2}, \ldots, c_{k_n}\}$. For any row i, suppose the original transmitted codeword is θ. If there is no error in c_{k_1}, clearly, after masking, the remaining bits in R_i match θ; otherwise, the remaining bits match $\theta' = \theta \oplus V(c_{k_1}, c_{k_2}, \ldots, c_{k_n}, c_{k'})$, which is another codeword. □

Lemma 2. *If the companion of Ω is a column denoted as $c_{k'}$, and if the minimum Hamming distance of the code is 3 or more, Δ_1 shall not succeed if Π consists of all but one true error columns, as well as another column $c_{k''}$ other than $c_{k'}$.*

Proof. WLOG, suppose R is repaired with $\{c_{k''}, c_{k_2}, \ldots, c_{k_n}\}$. Consider a row i where R_i has an error in c_{k_1}. WLOG, further suppose the original transmitted codeword, denoted as θ, is the all 0 codeword. If the claim is not true, suppose R_i matches a codeword θ'' in the remaining columns after masking. Clearly, $\theta'' \neq \theta$. Also, $\theta'' \neq \theta'$, where $\theta' = V(c_{k_1}, c_{k_2}, \ldots, c_{k_n}, c_{k'})$, because θ' is 1 in $c_{k'}$, while θ'' must be 0 in $c_{k'}$, as $c_{k'}$ has not been masked. θ'' can be 1 only in $\{c_{k''}, c_{k_2}, \ldots, c_{k_n}\}$, and therefore has only distance 2 to θ', a contradiction. □

A.4 Independence Assumption

In the rest of this section, when analyzing the decoding error probability, it will be assumed that the bits in the error columns are flipped independently with probability 0.5, referred to as the *independence assumption*. The independence assumption makes the analysis tractable, but is an approximation, because at least one bit must have been flipped in each error column. Fig. 20 shows that the approximations are reasonably close to the actual decoding error probabilities.

A.5 Analysis for CR 3 with 2-Column Errors

Suppose c_{k_1} and c_{k_2} are the two true error columns and their companion is $c_{k'}$. A decoding error occurs when in every row, either there is no error in both c_{k_1} and c_{k_2}, or there are errors in both c_{k_1} and c_{k_2}, such that Ξ contains only $c_{k'}$. BEC shall return prematurely, believing that there is only one error column. The error probability is 2^{-SF} under the independence assumption.

A.6 Analysis for CR 4 with 2-Column Errors

BEC always corrects 2-column errors for CR 4, because any column in Ξ must be a true error column; therefore, the repair when $|\Xi|$ is 1 or 2 must be successful. When $|\Xi|$ is 0, for any row in ϕ_2, the two columns where R and Γ differ must either be the true error columns, or their companion.

A.7 Analysis for CR 4 with 3-Column Errors

BEC is capable of correcting 3-column errors if $|\Xi| > 0$. The cases when $|\Xi| = 2$, 3, and 4 are straightforward. The case when $|\Xi| = 1$ is proved in the following.

Lemma 3. *BEC can correct the errors when there are three error columns and $|\Xi| = 1$.*

Proof. Note that BEC applies Δ_2 to R using the column in Ξ. Clearly, this will not repair R. The purpose is instead to discover the error columns. To see this, suppose there are indeed 3 error columns denoted as c_{k_1}, c_{k_2}, and c_{k_3}, and their companion is $c_{k'}$. First, suppose the column in Ξ is $c_{k'}$. Note that in this case, there is no row with only one error in one of the true error columns, i.e., every row has 0, 2, or 3 errors. As BEC did not return with the assumption that there was only 1 error column, there must be some rows with two errors. Suppose one of such rows, denoted as R_i, has errors in c_{k_1} and c_{k_2}. Clearly, flipping the bit in $c_{k'}$ will change R_i to a codeword except in c_{k_3}. As BEC did not return with the assumption that there were only 2 error columns, there must be at least another row with 2 errors not in c_{k_1} and c_{k_2}. Clearly, in this row, flipping the bit in $c_{k'}$ will reveal another true error column. Therefore, it is guaranteed that at least two true error columns will be revealed. Note that the same argument holds when the column in Ξ is one of the true error columns. □

As the errors are random and can either be 0 or 1 in the error columns in a particular row, when there are 3 error columns, in each row, there are 8 possible combinations. Let Ψ_x denote the probability that exactly x distinct error combinations occur in R. Clearly, $\Psi_1 = (\frac{1}{8})^{SF}$. For $x > 1$, it can be verified that the following recursive relation hold under the independence assumption:

$$\Psi_x = (\frac{x}{8})^{SF} - \sum_{y=1}^{x-1} \binom{x}{y} \Psi_y.$$

The following Lemma explains the decoding error probability.

Lemma 4. *Under the independence assumption, the decoding error probability of CR 4 when there are three error columns is*

$$\Psi_1 + 7\Psi_2 + 9\Psi_3 + 3\Psi_4 + 2^{-SF}.$$

Proof. Suppose c_{k_1}, c_{k_2} and c_{k_3} are the true error columns, and their companion is $c_{k'}$. With 3-column errors, any column in Ξ must be a true error column or $c_{k'}$. A decoding error occurs when BEC returns prematurely, i.e., returns with the assumption that the number of error columns is less than 3.

BEC may return prematurely with the assumption that there is 1 error column, if every row has 3 errors, or if some rows have no error but all other rows have 3 errors. Therefore, the probability is $\Psi_1 + \Psi_2$.

When testing the hypothesis that there are 2 error columns, there are three cases depending on $|\Xi|$. If $|\Xi| = 0$, every row has 0 or 2 errors. BEC will not return prematurely, because the 4 pairs must be from the same companion group, which occurs only if the errors occur in the same two columns in every row, or do not occur.

If $|\Xi| = 1$, first, suppose the column in Ξ is $c_{k'}$. Clearly, there is no row with a single error. In order for BEC to return prematurely at this point, there must be at least one row with two errors. WLOG, suppose the errors occur in c_{k_1} and c_{k_2} in this row. It follows that all rows with two errors must have errors in c_{k_1} and c_{k_2}, because the repair would not appear successful otherwise. There also must be at least one row with 3 errors. As long as these conditions are met, some row may have no error. Therefore, the probability is $\Psi_2 + \Psi_3$. Note that there are two other combinations of two error columns. Second, suppose the column in Ξ is c_{k_1}. There also must be least

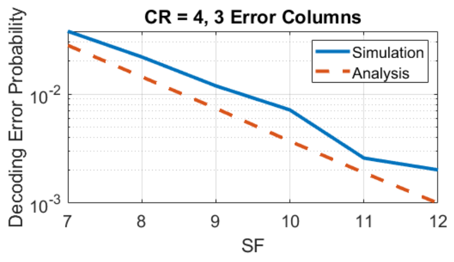

Figure 20: Decoding error probability when the CR is 4 with 3 error columns.

one row with two errors, and all such rows must have errors in the same two columns. There cannot be a row with 3 errors. As a result, in the rows with 2 errors, the errors must occur in c_{k_2} and c_{k_3}, because otherwise there will be less than 3 error columns in total. Therefore, the probability is $\Psi_2 + \Psi_3$. Note that Ξ may also be one of the other two error columns. The total error probability when $|\Xi| = 1$ is therefore $6(\Psi_2 + \Psi_3)$.

If $|\Xi| = 2$, first, note that Ξ cannot contain two true error columns, because masking the two columns will not make R_i identical to a codeword in the remaining columns, where R_i is a row with an error in the error column that has not been masked. Therefore, WLOG, suppose Ξ contains c_{k_1} and $c_{k'}$. In order for BEC to return prematurely at this point, among the rows with two errors, all errors must occur in the same two columns. In addition, the two columns must be c_{k_2} and c_{k_3}, because if one row, say, R_i, has errors in c_{k_1} and c_{k_2}, after masking c_{k_1} and $c_{k'}$, the remaining columns differ either in c_{k_2} or c_{k_3} with the closest codeword. There also must be at least one row with 3 errors, as well as one row with only one error in c_{k_1}. As long as these conditions are met, some row may have no errors. Therefore, the probability is $\Psi_3 + \Psi_4$. Note that the same argument can be applied to the cases when Ξ contains other true error columns. Therefore, the total error probability when $|\Xi| = 2$ is $3(\Psi_3 + \Psi_4)$.

In addition, decoding failure occurs when BEC tests the hypothesis that there are 3 error columns but finds $|\Xi| = 0$. The probability is 2^{-SF} under the independence assumption, because there can only be 0 or 2 errors in each row. □

Fig. 20 shows the decoding error probabilities when the CR is 4 with 3 error columns. It can be seen that: 1) the error probability is less than 0.04 when the *SF* is 7, 2) the error probability decreases as the *SF* increases, and 3) the analysis and the simulation results are reasonably close.

B ARTIFACT APPENDIX

B.1 Abstract

The artifact is the source code of TnB written in Matlab, which decodes wireless signals in LoRa networks. TnB is capable of decoding packets transmitted by commodity LoRa devices, even when multiple packets overlaps in time, i.e., experience collision. Accompanying the source code are trace files collected in the experiments, which can be decoded by TnB into data packets.

B.2 Artifact check-list (meta-information)

- **Algorithm:** TnB consists of a few new algorithms, including the packet detection algorithm, collision resolution algorithm, and error correction decoding algorithm.
- **Data set:** TnB can be tested with the accompanying data set, which contains 24 traces collected from LoRa networks with around 20 nodes using Universal Software Radio Peripheral (USRP). Each trace file is about 150 MB.
- **Run-time environment:** TnB can run on any machine with Matlab R2021b or above installed along with certain required toolboxes.
- **Hardware:** Any machine with 11th Gen Intel(R) Core(TM) i7-1195G7 @ 2.90GHz with 32 GB RAM or similar.
- **Execution:** About 60-120 seconds are needed to process a trace file on a typical machine.
- **Metrics:** The performance of TnB is measured by the number of packets decoded corrected (passed CRC).
- **Output:** The output can be of various forms with information at different levels of details about the decoded packets. The current program shows the total number of packets decoded. Also shown is a list of packets decoded from each node, including the sequence number, the estimated Signal to Noise Ratio (SNR), the start time of the packet in the trace, and the Carrier Frequency Offset (CFO) of the packet.
- **Experiments:** TnB can be tested with the accompanying data set. No experiment is needed.
- **How much disk space required (approximately)?:** The source code is less than 200 KB. The trace file is about 150 MB each. With a total of 24 trace files, the total size of the data set is about 3.6 GB. The trace files can be individually downloaded if there is no need to evaluate all files.
- **How much time is needed to prepare workflow (approximately)?:** Less than 10 minutes.
- **Publicly available?:** The source code and data set are both publicly available.
- **Workflow framework used?:** No workflow framework is used to automate and customize experiments.

B.3 Description

B.3.1 How to access. The source code can be found at
https://github.com/raghavrathi10/TnB
The traces files can be downloaded at
https://doi.org/10.5281/zenodo.7199527

B.3.2 Hardware dependencies. TnB is written in MATLAB, so any hardware that supports MATLAB can be used, such as 11th Gen Intel(R) Core(TM) i7-1195G7 @ 2.90GHz with 32 GB RAM.

B.3.3 Software dependencies. MATLAB R2021b or above along with the following toolboxes: Fixed Point Designer, Communications Toolbox, Signal Processing Toolbox and DSP System Toolbox.

B.3.4 Data sets. The trace files were collected in 3 deployed testbeds, named "Indoor," "Outdoor 1," and "Outdoor 2." For each testbed, 8 traces have been uploaded, one for each combination of Spreading Factor (SF), which is either 8 or 10, and Coding Rate (CR), which is from 1 to 4. The file name is, for example, "`indoor-SF8-CR3`." This set of trace files were selected from a total of 360 trace files, because they contain the most number of decoded packets for each SF and CR combination.

Information about the testbed set up can be found in the Evaluation section of the paper. Some additional details that may be useful are provided below:

- The LoRa device transmits a preamble that starts with 8 upchirps, followed by a symbol with peak at 9, then a symbol with peak at 17, then 2.25 downchirps.
- A transmitted packet starts with 4 bytes of header, followed by 2 bytes as the node ID, 2 bytes as the packet sequence number, 6 bytes of data, then 2 bytes of CRC.
- The signal was sampled by a USRP B210 at 1 Msps, where each sample consists of a real part and an imaginary part, both as 16-bit integers.

B.4 Installation

The user will need to install MATLAB R2021b or above first and then install the required Toolboxes in MATLAB mentioned earlier in "software dependencies."

After unzipping the source file, there should a directory, named `TnB`, which is the source code directory. The main file, named `TnBMain.m`, can be found under the `TnB` directory. The trace data should be downloaded to another directory, which can be called `AEexpdata` and can be at the same level as `TnB`, or according to what the user prefers.

B.5 Evaluation and expected results

To run TnB, the user may simply open Matlab, go to the TnB directory, and type "`TnBMain`" in the command window.

To test different traces, the user may open `TnBMain.m` and modify the first two lines. One is to select the trace, such as:

```
TraceName = '../AEexpdata/outdoor1-SF8-CR4';
```

and the other is to set the corresponding Spreading Factor:

```
SF = 8;
```

After the program finishes, the number of decoded packets is printed, such as:

```
– TnB decoded 278 pkts –
```

The complete list of files and the number of decoded packets are in the following:

- `indoor-SF8-CR1: 368`
- `indoor-SF8-CR2: 334`
- `indoor-SF8-CR3: 383`
- `indoor-SF8-CR4: 394`
- `indoor-SF10-CR1: 302`
- `indoor-SF10-CR2: 263`
- `indoor-SF10-CR3: 290`
- `indoor-SF10-CR4: 295`
- `outdoor1-SF8-CR1: 280`
- `outdoor1-SF8-CR2: 234`
- `outdoor1-SF8-CR3: 250`
- `outdoor1-SF8-CR4: 278`
- `outdoor1-SF10-CR1: 260`
- `outdoor1-SF10-CR2: 234`
- `outdoor1-SF10-CR3: 301`
- `outdoor1-SF10-CR4: 253`
- `outdoor2-SF8-CR1: 375`
- `outdoor2-SF8-CR2: 356`
- `outdoor2-SF8-CR3: 358`
- `outdoor2-SF8-CR4: 366`
- `outdoor2-SF10-CR1: 164`
- `outdoor2-SF10-CR2: 350`
- `outdoor2-SF10-CR3: 294`
- `outdoor2-SF10-CR4: 178`

It should be mentioned that the number of decoded packets may vary slightly, typically by 1-2 packets, for the same trace, when the trace is processes by different machines. This is likely because although TnB is a set of deterministic algorithms, it calls a few functions which may contain randomness.