Divide and Code: Efficient and Real-time Data Recovery from Corrupted LoRa Frames

***Abstract*—Due to power limitations and coexistence in ISM bands, up to 50% of the Long Range(LoRa)-frames are corrupted at low signal strengths (≈ -115dBm) and the built-in redundancy schemes in LoRa-Wide Area Network (LoRaWAN) cannot correct the corrupted bytes. To address this, higher Spreading Factors (SF) are used resulting in wasted energy, increased traffic load, and highly compromised effective data rate. Our on-field experiments showed a high correlation in corruption of close-by frames. We propose a novel Divide & Code (DC) scheme for LoRaWANs as an alternative to using higher SF. DC pre-encodes LoRa payloads using lightweight and memoryless encoding. After receiving a corrupted frame, DC uses a combination of likely patterns of errors, Time Thresholds (TT), and splitting payloads into subgroups for batch processing to recover frames effectively and maintain low complexity and timely operation. Implementing DC on our LoRa-testbed, we show it outperforms vanilla-LoRaWAN and Reed-Solomon codes in decoding and energy consumption. We show that our schemes decode up to 80.5% of corrupted payloads on SF10 by trying only 0.03% of all patterns of error combinations. TT keeps processing times below 2 ms with only minor reductions in the decoding ratio of corrupted payloads. Finally, we show that introducing 30% redundancy with DC results in minimum energy consumption and high decoding ratio at low SNRs.**

I. Introduction

Long Range (LoRa) is one of the prominent technologies for providing easy, low-power, and inexpensive Internet access to IoT devices using sub-GHz ISM bands. LoRa-PHY (referred to as LoRa from now on) is a proprietary modulation scheme developed by SemTech [1]. LoRa Wide Area Network (LoRaWAN) is an open standard [2]. Through a series of configurable parameters –including transmission power, carrier frequency, channel bandwidth, Spreading Factor (SF), Coding Rate (CR) –LoRaWAN trades-off data rate, which can reach up to 46.88 kbps, and operational range, reaching up to 5 km and 20 km in dense urban and rural environments, respectively [3], [4]. The SF dictates the number of raw bits of information per transmitted symbol (e.g., SF8 = 8 bits = 2^8 = 256 chips). The topology of a LoRa network (LoRaWAN) is depicted in Fig. 1, wherein a LoRa-frame is received directly by at least one gateway (GW), which forwards it to a network server over the Internet. The server removes any duplicate frames and sends the data to the application server. The MAC layer of LoRaWAN uses unslotted Aloha, wherein any frame is transmitted unconditionally upon generation.

A. Frame Corruption in LoRaWAN

Causes. LoRa network operates in the ISM bands thus it has to coexist with other ISM-based technologies. Therefore, LoRa devices must regulate their transmission power and duty cycle to reduce radio interference and also to conform with the regulations for using ISM bands. The combination of long transmission range –which increases the number of hidden devices– and low transmission power results in LoRa-frames being corrupted. Their received signal strength (RSS) is low due to multipath fading, shadowing, and scattering, and in Non-Line of Sight (NLoS) scenarios RSS can be as low as ≈ -115 dBm. In addition, the ALOHA type of MAC allows simultaneous/asynchronous transmissions increasing the number of collisions leading to frame corruption.

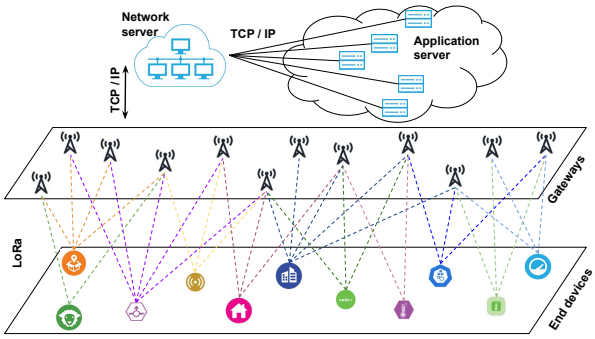

Figure 1: A Typical LoRaWAN network

Consequences. Most IoT applications involve data-driven systems of sensors, thus data corruption heavily compromises their seamless operation. LoRaWAN does not employ Automatic Repeat reQuest (ARQ) with ACK-messages since it is practically infeasible in networks involving thousands of LoRa-devices due to the duty-cycle limitation of 1% (that applies to both gateways and devices) and the absence of unicast between devices and the GW. If the ACKs were employed the devices would retransmit increasing the traffic in the network, and leading to increased interference which in turn results in even more corrupted frames. Data from The Things Network (TTN), which is a large-scale open LoRa network [5] involving 229 gateways and data from 488 devices, showed that only 3.47% frames were received by GWs using ACKs [6].

Data Recovery in LoRaWAN. LoRa-modems utilize Forward Error Correction (FEC) to detect and correct corrupted symbols. Soft Hamming codes [7] are used to create data-redundancy with different CR represented by $\frac{4}{4+x}$, where x is extra coded bits for every 4 bits and $x \in \{1, 2, 3, 4\}$. All four CRs can detect single errors, but 4/7 and 4/8 can further correct single errors or detect double errors. Moreover, 2 B, and 4 bits of Cyclic Redundancy Check (CRC) can be added at the end of the payload and header, respectively, to support the error detection.

Statistics on LoRa frame corruption. Xia *et al.* claimed that the reception of LoRa-frames relies primarily on detecting the frame-preambles, which –if not detected– usually lead to complete loss of the frame [8]. Further, Marcelis *et al.* found that up to 53% of the frames were lost at distances of 6 km

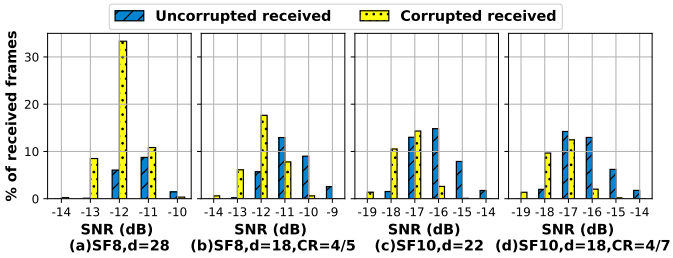

Figure 2: Portion of corrupted and correctly received frames out of 5000 transmitted frames, d bytes of payload

between devices and GWs [9]. However, the vast majority of their frames, *i.e.,* 95%, were transmitted on SF12. Although the above would necessitate the adoption of erasure coding mechanisms, Rahmadhani *et al.* [10], and Yazdani *et al.* [11] showed considerable amounts of received frames ending up corrupted. Specifically, they reported frame corruption of 32% and 50% in SF8, using multiple GWs and single-GW, respectively. Marcelis *et al.* proposed a novel use of convolutional and fountain codes to recover data from the frames that were already received [9] but not recovering the frames. Therefore, we delved deep into the characterization of frame corruption under various scenarios (more details in § II) and we observed up to 49.72% and 29.06% of frames being received corrupted on SF8 and SF10, respectively (see Fig. 2).

B. Constraints and Contributions

Accounting for the above observations and expecting a large portion of the projected 41 billion IoT devices by 2027 [12] to use LoRaWAN, recovering corrupted LoRa-frames is highly critical –not only to preserve the battery life of the already constrained LoRa-devices but also for the efficient utilization of the spectrum. Thus, we focus on **how to efficiently recover corrupted LoRa-frames and avoid dropping them, in real-time, independent of the built-in error correction and any other FEC at the application layer**. To address the question, we need to consider the constraints herein: (1) The gateways and the existing network infrastructure should remain unchanged. (2) No coordination through ACKs, since it increases the message overhead and extra listening times costing energy. (3) Any introduced byte redundancy due to encoding should be minimal, especially for high SFs, because the allowed payload size is limited (*i.e.,* 51 B for SF10-SF12). (4) The encoding should be memoryless, *i.e.,* using block codes to ensure the freshness of data. DaRe [9] corrects the frames based on the reception of subsequent frames, thus it is unsuitable for time-critical applications and it only recovers data but not the corrupted frames. (5) The decoding should be fast enough to deliver real-time reception, e.g., before reception of a future frame.

Our Contributions. To this end, we introduce **DC**, **Divide & Code**[1] – a novel coding scheme for LoRaWAN frame recovery. DC uses lightweight block codes to pre-encode the payload of a LoRa-frame and adds a limited number of encoded bytes before LoRa-FEC is applied, to provide robustness. The decoder of DC: **(a)** limits the decoding time to acceptable values; **(b)** segments large payloads to save decoding time; and **(c)** prioritizes the decoding of certain bytes in a frame based on their probability of being corrupted. Specifically, (1) We measure frame-corruption in LoRa networks and analyze patterns of (bursts of) errors and the correlation among errors within a frame (see § II). (2) We design a novel, memoryless coding scheme, called DC, that can work with the current LoRa standard transparently. DC recovers corrupted frames efficiently and in real-time well beyond the built-in error-correcting capability (see § III). (3) We define specific decoding algorithms depending on frame size and using the characteristics of frame corruption in LoRaWAN (see subsection III-C). (4) We provide a probabilistic analysis of successful decoding in DC (see § IV). (5) Through simulations and real-world experiments using our testbed of LoRa-modules, we compare DC not only to the built-in error-correcting scheme of LoRaWAN but also to Reed Solomon codes and the recent scheme ReDCoS [11]. Furthermore, we dictate the ratio among data bytes and error-correcting bytes that optimizes consumption and decoding ratio (see § V).

[1]This is similar to the Divide & Conquer principle.

II. Characterization of Error Pattern

Before we develop more effective, faster, and energy-efficient data recovery algorithms, we first attempt to characterize frame corruption in LoRaWAN. We find the conditions under which the majority of frame corruption is observed. Further, we evaluate which parts of LoRa-frames are more prone to corruption, analyze various aspects of symbol corruption and understand how errors occur and whether there is a pattern to it.

Data Collection. We conduct experiments on our testbed using SX1261 LoRaWAN modules. We vary the transmitter-receiver distance between $100-300$ m, using low transmission powers (0 dBm) under (N)LoS conditions, employing different types of isotropic antennas ($0-2$ dBi gain). The devices operate at 868.1 MHz, using the bandwidth of 125 kHz on SF8 and SF10. The following four cases are studied: (SF8, d = 28, no CR), (SF8, d = 18, CR = 4/5), (SF10, d = 22, no CR), and (SF10, d = 18, CR = 4/7) where d is the number of bytes within the payload. 5,000 frames with random data are transmitted for each case. Note that we used relatively small frames for our on-field experiments (Fig. 2) considering the data from TTN, which shows 75.3% of the frames having at most 30 B payload [6].

Frame corruption. As seen in Fig. 2, the frames received with SNR values between $[-10,-13]$ dB and $[-16,-19]$ dB for SF8 and SF10, respectively, have high probabilities of being corrupted. These values correspond to RSSI of around -115 dBm. Frames received with SNR values above this range are correctly decoded and below this range are lost completely. For example, 33.32% of the 5000 frames (*i.e.,* 1675) that were transmitted on SF8 with 28 B payload, *i.e.,* d = 28, were received corrupted when SNR was -12 dB. For high SFs, the total number of corrupted frames is reduced due to more robust encoding, *i.e.,* 49.72% frame corruption for SF8 compared to 29.06% for SF10 as shown in Fig. 2(a) and Fig. 2(c), respectively. We did not focus on SF11 and SF12 because frame loss is mostly observed instead of corruption, as Marcelis *et al.* have confirmed [9]. With inbuilt redundancy using CR, part of the corrupted bytes could be retrieved as seen in Fig. 2(b) and Fig. 2(d). However, the sheer amount

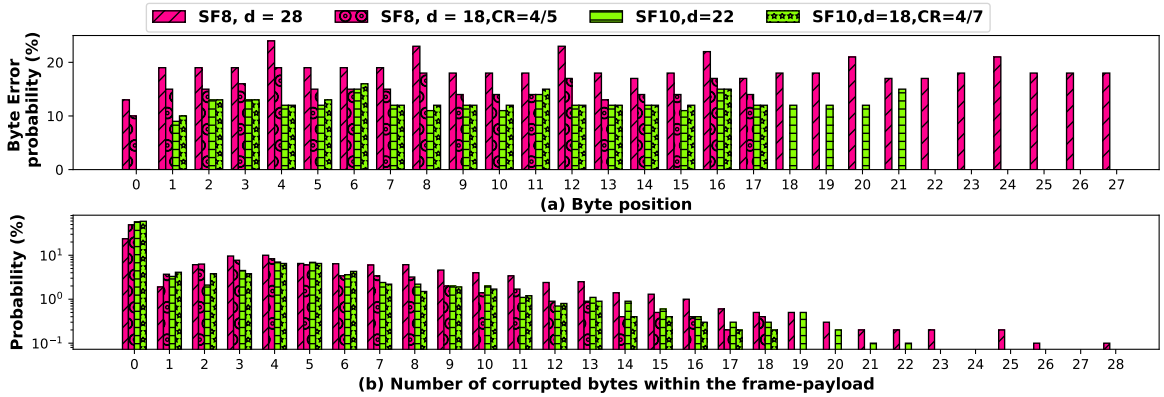

Figure 3: (a) Byte Error Rate (β) versus byte position, (b) Probability of number of corrupted bytes.

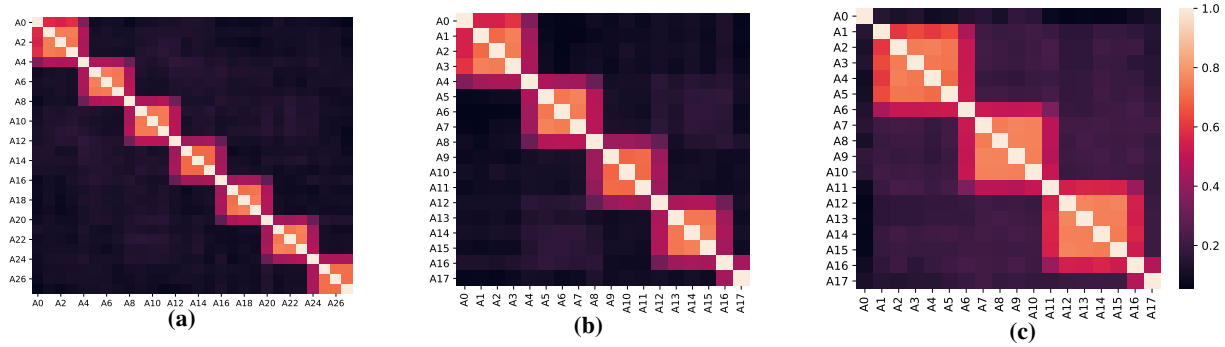

Figure 4: Heat-map of showing correlation between corrupted frames: (a) SF8, $d = 28$, (b) SF8, $d = 18$, CR = 4/5, (c) SF10, $d = 18$, CR = 4/7.

of corrupted (bursts of) bytes render the simple LoRa-FEC unable to correct them, still leading to 32.82% and 25.78% of frames being corrupted at SF8 and SF10, respectively.

Byte error probability vis-á-vis position. Fig. 3(a) depicts the probability of a byte being corrupted based on its position in a payload for the above cases. For SF8, the first byte has the lowest probability of being corrupted. For SF10, the first and the second bytes have the lowest probability of being corrupted. At SF8, except for the first byte, there is a pattern where 3 bytes have an average probability of being corrupted (~18% for $d = 28$ and ~14% for $d = 18$) and the fourth byte has a higher probability (~22% for $d = 28$ and ~18% for $d = 18$). This pattern repeats per 4 B. At SF10 except for the first and the second bytes, there is a similar pattern where there is 4 bytes with an average probability of being corrupted (~12%) and 1 byte with a higher probability (~15%). This pattern repeats per 5 bytes.

Number of corrupted bytes. Fig. 3(b) shows the probability of receiving a payload with different number of corrupted bytes for the previous 4 cases. Apart from the case of no corruption (0 bytes), a peak is observed for 3, 4 (4, 5) corrupted bytes per received payload in SF8 (SF10) confirming our observations regarding bursts of errors in Fig. 3(a).

Correlation among corrupted bytes. We arrange received data bytes (of corrupted frame) as a row having d columns corresponding to a byte/symbol. Each element takes a value 0/1; 1 if the associated byte is corrupted, and 0 otherwise. We define A_i as the i-th column associated with the i-th byte of the received payload where $i \in \{0, 1, ..., d-1\}$. The Pearson correlation coefficient between any A_i and A_j depicts the strength of linear correlation between the given two columns. It ranges from -1 to 1 where correlation coefficients of 0 and ± 1 depict no correlation and a high correlation, respectively. Fig. 4 visualizes the correlation matrix as a heatmap for (SF8, $d = 28$), (SF8, $d = 18$, CR = 4/5), and (SF10, $d = 18$, CR = 4/7). As depicted in Fig. 4, there is a pattern for each SF. Considering SF8, for any $\alpha \in \mathbb{Z}$ and $\beta \in \mathbb{Z}$, considering $\alpha' = (\alpha \oplus \theta)$ and $\xi = (\alpha - \alpha' + \gamma)$ we have:

$$\rho(A_\alpha, A_\beta) \begin{cases} \geq 0.3 \text{ if } & \alpha' = \gamma \ \& \ \beta \in [\alpha - \theta, \alpha + \theta] \\ \geq 0.3 \text{ if } & \alpha' \neq \gamma \ \& \ \beta \in [\xi, \xi + \theta] \\ < 0.3 & \text{otherwise,} \end{cases} \quad (1)$$

where ($\theta = 4$, $\gamma = 0$) for SF8, ($\theta = 5$, $\gamma = 1$) for SF10, and $\oplus$ is the modulo operation. For SF10, A_0 is an exemption as the correlation coefficient between A_0 and any other byte is less than 0.3. The previously observed patterns are only dependent on the SF and not the CR used, or payload size.

We verified this pattern in various environments, locations, transmission distances, SFs, and with different devices to ensure external noise was not the cause of the pattern. We believe that these patterns are due to the mechanisms and steps involved in decoding the incoming LoRa messages on SX1261. It seems that for every 4th or 5th byte the hardware buffer gets refreshed or due to internal noise that pulls the RSSI further down. SemTech does not describe publicly the decoding schemes on SX1261 at the hardware level.

III. Proposed Idea

Divide & Code (DC) scheme for LoRaWAN pre-encodes the payload of frames using lightweight schemes. This is indepen-

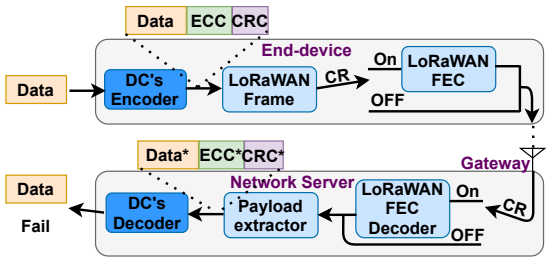

Figure 5: The encoder and decoder of corrupted frame.

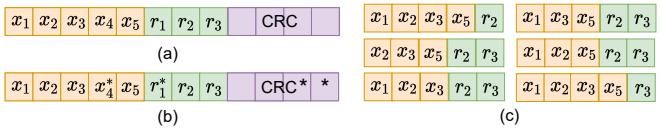

Figure 6: Data retrieval example for Case 3: (a) Transmitted payload, (b) Received payload (corrupted bytes marked with star), (c) All k-combinations generating correct CRC.

dent of LoRa-FEC, which is added afterwards. DC exploits the frame-corruption patterns of LoRaWAN to improve the speed of- and, packet recovery.

A. Encoder

Although DC can use different CRCs and ECCs, we focus on CRC-32 and Reed-Solomon codes (RS) [13]. In particular, we use systematic $RS_{256}(n,k)$ over the finite field 2^8 with n and k being the byte-size of the codeword and the message, respectively. We define t and l as the number of error-correcting bytes and CRC check bytes, respectively, where $n = k + t$. The encoded data is generated by attaching t RS error-correcting bytes to the original data. The payload of the LoRaWAN frame is generated by concatenating the encoded data and a CRC of length $l = 4$ bytes calculated over the encoded data. If CR is on, LoRa's FEC is added to the frame before transmitting (see Fig. 5).

B. Decoder

DC's decoder is located at the network server, as in Fig. 5, but it could be placed at GW if it is computationally powerful. Potentially corrupted encoded data and the CRC are parsed at the DC's decoder. The DC's decoder recovers the data if one of the following cases is true:

Case 1 The received encoded data and CRC are not corrupted.

Case 2 The received CRC is not corrupted and at least k bytes out of the $k+t$ are correct in the received encoded data.

Case 3 A minimum of H bytes of the received CRC are not corrupted (to verify the recovered CRC) and a minimum of $k+1$ bytes of the received encoded data are not corrupted. H is determined by the user (usually $H \geq 2$), depending on the size of CRC, and determines the strictness of CRC verification.

From the given set of $k+t$ elements, the decoder tries all the k-combinations, *i.e.,* a total of $C(k+t,k) = \frac{(k+t)!}{k!t!}$ combinations. For each k-combination, the decoder calculates the remaining t bytes of the encoded data by decoding the RS coded data, encoding the recovered data, and then calculating the associated CRC. DC's decoder is detailed in Algorithm 1, where T is the maximum allowed computation time, after which the decoding procedure stops (see subsubsection III-C3).

Example 1. *Let us consider $k = 5$, $t = 3$, $H = 2$. Then, the data bytes $\{x_1, x_2, x_3, x_4, x_5\}$ are mapped to the encoded data $\{x_1, x_2, x_3, x_4, x_5, r_1, r_2, r_3\}$, and transmitted. Obviously if all the received bytes are uncorrupted, data is retrieved (Case 1). But let us assume the gateway receives $\{x_1, x_2^*, x_3, x_4, x_5^*, r_1^*, r_2, r_3\}$ where a corrupted byte is shown with a star and CRC is uncorrupted. Then, the decoder tries different k-combinations. Once, it selects the uncorrupted bytes $\{x_1, x_3, x_4, r_2, r_3\}$, it generates the correct encoded data and the correct CRC, which is verified using the received CRC. Hence, data is retrieved (Case 2). Now we assume 2 bytes of the received CRC are corrupted and the received encoded data is $\{x_1, x_2, x_3, x_4^*, x_5, r_1^*, r_2, r_3\}$ as shown in Fig. 6. Then, every time the decoder selects a k-combination from the set of the correct bytes $\{x_1, x_2, x_3, x_5, r_2, r_3\}$, the same correct CRC is generated, i.e., $C(6,5) = 6 = k+1$ times, see Fig. 6 (c). This CRC has $H = 2$ bytes which are equal to their counterparts from the received CRC, and hence is verified. Thus, the data can be retrieved (Case 3).*

C. Proposed Schemes for Realtime Decoder

In Algorithm 1, we propose three different schemes to decode in realtime as described hereafter.

1) Utilizing the Pattern of CRCs Repetitions (PCR): Regarding Case 3, instead of repeating the correct CRC at least $C(k+1,k) = k+1$ times to retrieve the payload, we show that the repetition of the same CRC 2 times is enough to guarantee that it will be repeated a minimum of $k+1$ times. Thus, we stop the procedure earlier saving time and computations. Let us take an example.

Example 2. *In Fig. 6, we assume that the decoder has tried two k-combinations of $\{x_1, x_2, x_3, x_5, r_2\}$ and $\{x_1, x_3, x_5, r_2, r_3\}$. If they generate the same CRC, then the generated encoded data is also equal. Thus, the generated r_3 by the first combination must be equal to the selected r_3 in the second combination. Hence, any k-combination out of the concatenation of these two k-combinations, i.e., $\{x_1, x_2, x_3, x_5, r_2, r_3\}$ will generate the same CRC.*

Accordingly, in Algorithm 1, line 30 we check if a CRC is repeated 2 times, not $k+1$ times. In general, if the decoder tries all the k-combinations, a given CRC can be repeated $(k+i,k)$ times –not any other value in between– where $i \in \{0,1,2,..,t\}$. If the encoded data is uncorrupted, there exists only one CRC which is repeated $C(k+t,k)$ times. Otherwise, normally there are plenty of CRCs which are repeated only $C(k,k) = 1$ time, and a few CRCs which are repeated $C(k+i,k)$ for $i \in \{1,2,..t\}$. Fig. 7 depicts an example of the number of CRC repetitions when the decoder tries all the k-combinations considering corrupted encoded data. Among $CRC\#3, CRC\#5$, or $CRC\#7$, the one that has H bytes in common with the received CRC is considered the correct CRC. Increasing H decreases the possibility of false decoding, as it employs stricter CRC verification.

2) Utilizing the Pattern of Errors (PE): As mentioned in subsection III-B, the decoder tries up to $C(k+t,k)$ different k-combinations. For selecting k bytes for generating each k-combination, a Boolean vector, V, of size $k+t$ is generated in which the t leftmost elements are ones, and

Algorithm 1: DC's decoder:

Input: D_r: received data, E_r: received error correcting bytes, CRC_r: received CRC
Output: D: retrieved data, failed decoding

1 $timer = 0$ // Keeps the time in ms, increases with time
2 $pc = 0$ // Permutation counter
3 $CRC\_t \leftarrow \{\}$ // Pairs of (CRC:#repetition)
4 $V \leftarrow [0].size(k+t)$ // A Boolean vector
5 V[k:k+t-1] = 1 // Set the last t elements
6 Step 1: Check if the received encoded data is uncorrupted.
7 $x \leftarrow find\_CRC(D\_r, E\_r)$
8 **if** $x == CRC\_r$ **then**
9 $\quad return(D = D\_r)$ // End: SUCCESS (Case 1)
10 $D_{e,r} \leftarrow concatenate(D_r, E_r)$
11 **while** $pc < (C(k+t,k))$ **do**
12 $\quad$ Step 2: Using time threshold (TT).
13 $\quad$ **if** $timer \geq T$ **then**
14 $\quad\quad return(\text{“}Failed\ Decoding\text{”})$ // End: Failed
15 $\quad$ Step 3: Generate an uncorrupted encoded data candidate: Try a k-combination
16 $\quad y \leftarrow [].size(k+t)$
17 $\quad y \leftarrow select\_k\_bytes(D_{e,r}, V)$ // The indices of the zero elements in V indicates the selected bytes
18 $\quad y \leftarrow calculate\_remaining\_t\_bytes(y)$
19 $\quad x \leftarrow find\_CRC(y)$
20 $\quad$ Step 4: Check if the candidate matches the received CRC
21 $\quad$ **if** $x == CRC\_r$ **then**
22 $\quad\quad D = pick\_first\_k\_bytes(y)$
23 $\quad\quad return(D)$ // End: SUCCESS (Case 2)
24 $\quad$ Step 5: Save/Check the recovered CRC.
$\quad\quad$ // Check key x availability in CRC table CRC_t
25 $\quad$ **if** $!find\_key(x, CRC\_t)$ **then**
26 $\quad\quad CRC\_t[x] = 1$
27 $\quad$ **else**
28 $\quad\quad CRC\_t[x] += 1$
29 $\quad\quad$ Step 6: Check CRC repetition (PCR).
30 $\quad\quad$ **if** $CRC\_t[x] == 2$ **then**
31 $\quad\quad\quad h \leftarrow find\_\#similar\_bytes(x, CRC\_r)$
32 $\quad\quad\quad$ Step 7: Verify the recovered CRC.
33 $\quad\quad\quad$ **if** $h \geq H$ **then**
34 $\quad\quad\quad\quad D = pick\_first\_k\_bytes(y)$
35 $\quad\quad\quad\quad return(D)$ // End: SUCCESS (Case 3)
36 $\quad$ Step 8: Permute V.
37 $\quad$ **if** $pc < (k+1)$ **then**
38 $\quad\quad V = rotate\_left(V)$ // PE: Rotate by 1 byte
39 $\quad$ **else**
40 $\quad\quad V = lex\_permute(V)$ // Permute to reverse lexicographic order, skip those tested by PE
41 $\quad$ pc++
42 Step 9: return failure if algorithm has not ended yet by a return
43 $return(\text{“}Failed\ Decoding\text{”})$ // End: Failed

the remaining elements are zero. The indices of the zero elements in this vector represent the indices for the current k-combination. To generate the next k-combination, the Boolean vector is permuted to the reverse lexicographical order [14] (Algorithm 1. line 40). Accordingly, finding the correct k-combinations might take a long time. To alleviate this issue, we utilize the insights from **§ II** to make this search process more efficient. Our goal is to select k bytes while t bytes are left unselected. Eq. 1 represents the correlation between corrupted bytes. As observed in Fig. 4, adjacent bytes have a higher probability of being corrupted. Therefore, the decoder should prioritize them. For example, in SF8 if A_3 is corrupted, there is a relatively higher probability that $A_0, A_1, A_2, or A_4$ are corrupted as well. Afterwards, if data is not decoded yet, the decoder tries all the remaining combinations in reverse lexicographical order. To select the few k-combinations, initially, the t rightmost elements are set to zero (Algorithm 1. line 5). To generate the next k-combination, the Boolean vector is rotated

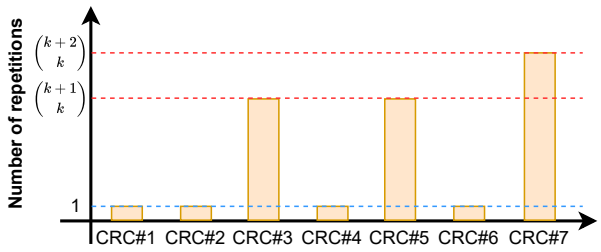

Figure 7: Number of CRC repetitions if decoder tries all k-combinations considering corrupted encoded data.

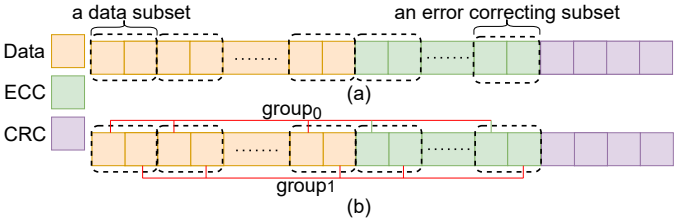

Figure 8: Decoding large payloads (a) Making subsets of bytes, (b) Proposed idea for large payloads.

to the left by 1 byte. The process stops when all the leftmost elements are ones, i.e., after k rotations. This technique is called PE (Algorithm 1. line 38), hereafter. Accordingly, PE checks only $k+1$ k-combinations initially which is much lower than the total number of k-combinations. For instance, considering $k = 20$ and $t = 4$, the PE checks only 21 k-combinations initially while there are 10626 of them.

3) Setting Time Threshold (TT): There is a considerable portion of large frames which cannot be decoded due to lots of corrupted bytes. The time needed for the decoder to dictate that decoding is not possible would be immense as all the $C(k+t,k)$ k-combinations must be tried. Since using the PE most of the payloads are decoded in a short time, we define a time threshold (TT) discarding any payload not decoded before this specified boundary. In § V, we study the effect of changing the T. We call this TT technique.

D. Decoding Large Payloads (DLP)

DLP is a complement to Algorithm 1. Assume DLP is not used unless otherwise stated. Although the main focus of DC is applications with small payloads, transmissions of relatively larger payloads can occur, especially at SF7 and SF8 which allow up to 222 B of payload [2]. Although the encoding is lightweight, the decoding of relatively large payloads can become problematic due to the sheer numbers of different k-combinations that have to be examined. Let us take an example.

Example 3. *Assuming that a payload with $k = 20$, $t = 4$ needs 0.458 s to decode (See § V), we focus on finding the time needed for a payload with $k = 40$, $t = 8$. The number of k-combinations increases by a factor of $\frac{C(48,40)}{C(24,20)} = 35,512$. Further, using Gaussian elimination, the complexity increases by $\frac{40^3}{20^3} = 8$. Thus, the total complexity increases by a factor of 284,096, and hence the expected decoding time would be $284,096 \times 0.14 \approx 113627\ s \approx 31.5\ hours$.*

Solution. In this section, we investigate the solution for the aforementioned problem that leads to immense decoding times. We expand the $k+t$ bytes into subsets with Q bytes

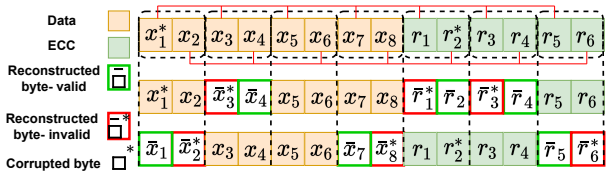

Figure 9: Example of DLP.

where $Q > 1$, $k \oplus Q = 0$, and $t \oplus Q = 0$. The aforementioned conditions guarantee that all subsets have exactly Q bytes and each subset has either data bytes (referred to as data subset, hereafter) or error-correcting bytes (referred to as an error-correcting subset, hereafter) and not both. This consideration is both at the encoder and the decoder. Then, every error-correcting subset is a linear combination of data subsets. This decreases the computational complexity dramatically, as the number of combinations and Gaussian elimination complexity for a payload with k, t, and Q parameters become equal to the ones for $k' = \frac{k}{Q}$, $t' = \frac{t}{Q}$, and $Q' = 1$. However, the above can lead to a decrease in the probability of decoding the data as even one corrupted byte renders its whole corresponding subset useless, and any combination made by the decoder using that subset leads to corrupted results. This worsens by increasing Q for larger payloads, as it increases the probability of having at least one corrupted byte per subset. We account that the i-th byte in a given error-correcting subset is only dependent on the i-th bytes in all data subsets (see Fig. 8(b), red lines). Accordingly, the $k + t$ bytes of the encoded data form Q *groups* where group i for $i \in \{0, 1, ..., Q-1\}$ contains i-th byte of each subset. For instance, group 0 in Fig. 8 includes the first byte of all the subsets. We define $K = \frac{k}{Q}$ and $T = \frac{t}{Q}$ as the number of data subsets and the number of error-correcting subsets, respectively. Then, we try to find uncorrupted groups, individually. The whole uncorrupted encoded data can be found by putting the uncorrupted groups together. DLP uses the same decoding procedure as in Algorithm 1 with the following main modifications: ❶ Decoder tries K-combinations by selecting K subsets out of $K+T$ subsets and generates the remaining T subsets. ❷ DLP keeps track of the CRC repetitions for each group, individually, rather than for the whole encoded data. Thus every time the decoder tries a K-combination, it calculates Q CRCs. ❸ If a CRC for a given group is repeated 2 times (PCR), it is considered a candidate and its corresponding group is saved. ❹ If more than one groups have CRC candidates, these are combined, and a resulting CRC is calculated from the combination. If H bytes of the calculated CRC match the corresponding bytes of the received CRC, the encoded data candidate is correct.

Note that DLP still accounts for PCR, TT, and PE as adjacent subsets have a high probability of being in a similar condition regarding being corrupted especially for low values of Q.

Example 4. *Fig. 9 shows an example where $Q = 2$. x_1 and r_2 are corrupted during the transmission. As $K = 4$, the decoder tries different 4-combinations out of 7 subsets. There are a total of $C(7,4)$ K-combinations. In each trial, the receiver selects $K = 4$ subsets and generates the remaining 3 subsets. As shown in Fig. 9, on the first try, the first, third, fourth, and seventh subsets are selected. This means that x_1^* is one of the selected bytes. Since it is the first byte in its subset, all the first bytes in the generated subsets are corrupted. But, the second byte in the generated subsets, associated with group 1, is uncorrupted. Accordingly, in the second try, the second bytes of the generated subsets are corrupted while the first bytes are uncorrupted. Therefore, although in both tries there was 1 corrupted byte, the first and second tries generated the uncorrupted bytes for groups 1 and 0, respectively.*

Table I
Mapping of the Data Sets

Data set (§ II)	Data set (§ V)
(SF8, $d = 28$)	(SF8, $k = 20$, $t = 4$)
(SF8, $d = 18$, CR = 4/5)	(SF8, $k = 10$, $t = 4$, CR = 4/5)
(SF10, $d = 22$)	(SF10, $k = 10$, $t = 8$)
(SF10, $d = 18$, CR = 4/7)	(SF10, $k = 10$, $t = 4$, CR = 4/7)

DLP recovers the data in the aforementioned three cases in subsection III-B. However, in Case 2, a minimum of K uncorrupted subsets are required, and in Case 3, a minimum of $K+1$ bytes in each group needs to be uncorrupted. The latter corresponds to a minimum of $Q(K+1) = QK + Q = k + Q$ uncorrupted bytes out of the $k+t$ bytes of the received encoded data where the corrupted bytes are uniformly distributed between the groups.

IV. Analysis of DC

We evaluate DC in terms of **decoder throughput, decoding ratio**, and **false decoding ratio** for Byte Error Rates, denoted as β. β is the probability of a byte getting corrupted during the transmission and is calculated as the ratio of corrupted bytes over transmitted bytes. We characterized frame corruption and error pattern in § II. In this section for the sake of simplicity, we assume that β is i.i.d. See § V to analyze the effect of our assumptions. We define the decoder *throughput* as the amount of data (i.e., excluding CRC and error-correcting bytes) processed. The Decoding Ratio (DR) indicates the ratio of decoded over total received payloads. The False Decoding Ratio (FDR) is the ratio of falsely decoded payloads over total. Increasing H reduces the FDR as the recovered CRC must be validated more strictly, i.e., requiring more byte-matches.

Decoding Probability. We define $P(w, e)$ as the probability of having e corrupted out of w given bytes,

$$P(w,e) = C(w,e)\beta^e(1-\beta)^{w-e}, \forall e \in [0,w] \quad (2)$$

Received CRC is uncorrupted: The probability of receiving an uncorrupted CRC is $P(l, 0)$. In this case, DC recovers the data if up to and including t bytes are corrupted out of the $k + t$ bytes which happens with probability $\Sigma_{i=0}^{t} P(k+t, i)$.

Matched bytes between recovered and received CRC greater or equal to H and less than l: The number of corrupted bytes of the CRC is within $[1, l-H]$ which happens with probability $\Sigma_{i=1}^{l-H} P(l, i)$. In this case, DC recovers the data if up to and including $t - 1$ bytes are corrupted out of the $k + t$ bytes, i.e., a minimum of $k + 1$ bytes are intact, which happens with probability $\Sigma_{i=0}^{t-1} P(k+t, i)$.

Matched bytes between recovered and received CRC less than H: Decoding is not possible. Accordingly, the probability of successfully decoding a frame is,

$$\mathbb{P}_D = P(l,0)\Sigma_{i=0}^{t} P(k+t,i) + \Sigma_{i=1}^{l-H} P(l,i)\Sigma_{i=0}^{t-1} P(k+t,i) \quad (3)$$

Decoding Probability of DLP. A subset is corrupted if it has

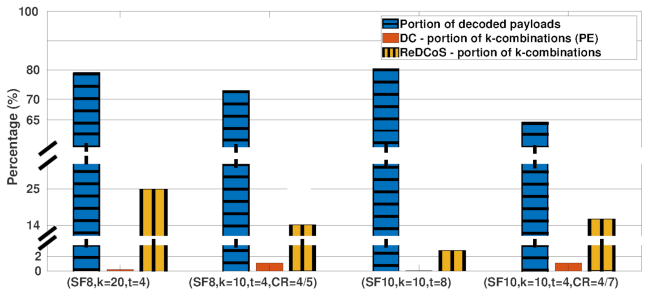

Figure 10: Portion of decoded payloads and tried k-combinations using DC's PE feature compared to ReDCoS

a minimum of one corrupted byte. The probability of a subset getting corrupted during the transmission, denoted as Γ, is:

$$\Gamma = 1 - \{C(Q,0)\beta^0(1-\beta)^{Q-0}\} = 1 - (1-\beta)^Q. \quad (4)$$

The probability of E corrupted out of W given subsets is:

$$\mathcal{P}(W,E) = C(W,E)\Gamma^E(1-\Gamma)^{W-E}, \forall E \in [0,W]. \quad (5)$$

Following a similar approach as for Eq. 3, DC using DLP recovers the data with the probability,

$$\mathbb{P}_D = P(l,0)\Sigma_{i=0}^{T}\mathcal{P}(K+T,i) + \Sigma_{i=1}^{l-H}P(l,i)\Sigma_{i_1=0}^{T-1}\Sigma_{i_2=0}^{T-1}\ldots \\ \ldots\Sigma_{i_Q=0}^{T-1}P(K+T,i_1)P(K+T,i_2)\ldots P(K+T,i_Q), \quad (6)$$

where $\Sigma_{i=0}^{T}\mathcal{P}(K+T,i)$ is the probability of up to and including T subsets are corrupted out of $K+T$ subsets, and $\Sigma_{i_2=0}^{T-1}\ldots\Sigma_{i_Q=0}^{T-1}P(K+T,i_1)P(K+T,i_2)\ldots P(K+T,i_Q)$ is the total probability of every group having a maximum of $T-1$ corrupted bytes out of the $K+T$ bytes.

V. Performance Evaluation

Data set and Set up. We used the data collected by our testbed as described in § II unless stated otherwise. Each data set corresponds to a given k and t value, as depicted in the Table. I. Note that $d = k+t+4$, where 4 bytes are used for the CRC. For our experiments, we use $RS_{256}(k+t,k)$ as the ECC in the DC algorithm with random data in the payload. The encoder and decoder are implemented as a C++ library. The transmitted LoRaWAN frames contain 8 and 2 LoRa bytes for preamble and physical header, respectively. The decoding is done on a Core i7-7820HQ, 2.90GHz network server using a single thread. We consider $H = 2$ and a large time threshold to process all payloads unless stated otherwise.

Schemes for comparison. We compare our technique with No-Added-ECC and RS-ECC. The latter uses a Reed-Solomon code, $RS_{256}(k+t,k)$, to recover the data, if the CRC is not corrupted, up to and including $\lfloor \frac{t}{2} \rfloor$ of corrupted bytes of encoded data.

Effect of PE. As mentioned in subsubsection III-C2, PE tries a limited set of k-combinations with a high probability of decoding the data. PE's portion of k-combinations is equal to $\frac{k+1}{C(k+t,k)}$. Fig. 10 shows the ratio of decoded payloads by PE over a total of decoded payloads considering corrupted payloads. For instance, for SF10, $k = 10$, $t = 8$, PE decodes 80.46% of the corrupted payloads received which are decodable by trying only up to 0.03% of all the k-combinations, and the remaining 19.54% can be decoded using the rest 99.97%

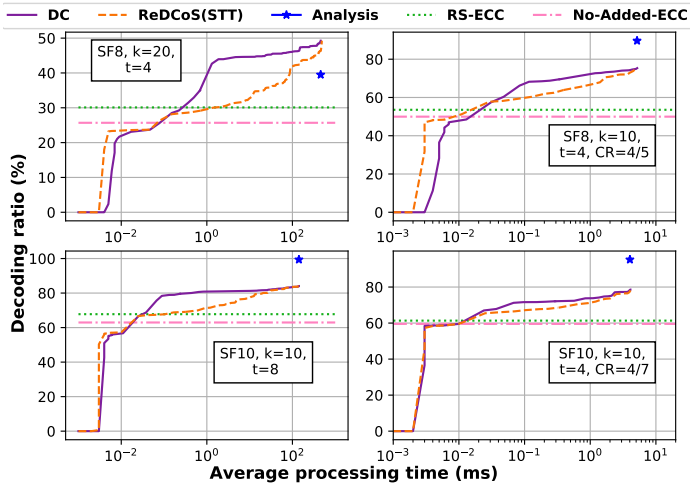

Figure 11: Decoding ratio per processing time using TT.

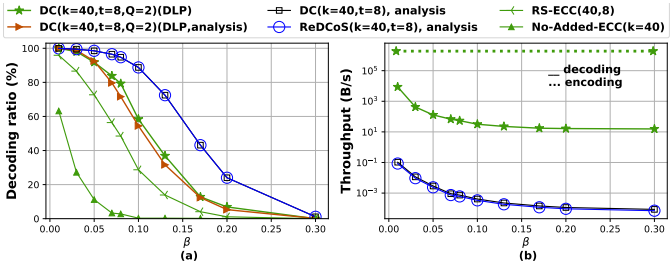

Figure 12: DLP's (a) decoding ratio and (b) throughput

k-combinations. Further, DC outperforms ReDCoS regarding the number of k-combinations needed to achieve the same percentage of decoding. As seen in Fig. 10 ReDCoS tries 125× more k-combinations than DC to decode 80% of payloads at (SF8, $k = 20$, $t = 4$). For all cases, we found FDR=0.

Effect of TT. Fig. 11 compares the decoding ratio of DC using different time thresholds against ReDCoS, RS-ECC, and No-Added-ECC. We proposed a similar approach as TT for ReDCoS (labelled as ReDCoS(STT)) for a fair comparison. After a point, increasing the time threshold does not affect the decoding ratio as the payloads are either decoded or failed to decode. We assume that RS-ECC and No-Added-ECC process the payload instantly. The initial sharp increase in decoding ratio for DC and ReDCoS corresponds to the uncorrupted payloads (instantly decoded). We also observe a sharp increase in decoding ratio for DC due to the use of PE, i.e., trying the most likely k-combinations. DC for SF8, $k = 20$, $t = 4$ outperforms RS-ECC and No-Added-ECC after average processing time of as low as 0.27 ms and 0.083 ms. The average processing time can be further reduced from 458 ms to 1.91 ms at the expense of only 5.4% lower decoding ratio in DC for SF8, $k = 20$, $t = 4$. In contrast, ReDCoS' decoding ratio decreases by 19.7% when reducing the average processing time from 458 ms to 1.91 ms. According to Fig. 11, the decoding ratio deviates at most 16% from the experimental results. For the analysis, we consider the average β.

Effect of DLP. Fig. 12 studies the DLP in terms of decoding ratio, and throughput over 10 different values for Byte Error Rate, β, considering $k = 40$ and $t = 8$. We consider synthetic data to study different β values, i.e., bytes are corrupted at random considering the given β. The decoder throughput is

calculated as in Example. 3. DC-with-DLP provides up to 30.79% and 80.6% better decoding ratio compared to RS-ECC and No-Added-ECC, respectively. The increased probability of having at least one corrupted byte per subset reduces the decoding ratio of DC-with-DLP compared to DC-without-DLP by up to 35.57%. However, this is a small price to be paid for encoding large payloads, which can be seen in the throughput results.

Fig. 12(b) shows that the use of DLP mechanism improves the throughput of the decoder by up to $180,854\times$. Although ReDCoS reports similar values of decoding ratio compared to DC (Fig. 12(a)), it has a throughput similar to DC-without-DLP, i.e., it is unable to process large frames. The encoder's throughput for DC with DLP reaches 1.9 MB/s, while the FDR is zero for all cases with $H = 3$. The simulated results for the decoding ratio of DC using DLP in Fig. 12 are consistent with the analysis in § IV.

False decoding ratio of DC. Throughout our experiments we reported an FDR of 0. In contrast, ReDCoS reports an FDR of 0.13% for $k = 10$ and $t = 8$ and 0.11% for $k = 20$ and $t = 4$ [11], i.e., more than 0.1% of decoded payloads of ReDCoS are falsely decoded.

Finding the Optimum Point. Given a fixed $k+t$, we evaluate the optimum value for $\frac{k}{k+t}$, i.e., which fraction of the encoded payload corresponds to information bytes and which to added redundancy, to deliver a high decoding ratio while keeping energy consumption low. For No-Added-ECC, $k+t$ changes only by changing k. For fairness, No-Added-ECC's results are shown for the same k value as in DC and RS-ECC. Fig. 13 shows the decoding ratio, transmission energy per bit, and the total number of k-combinations utilized. For each case, the on-field experiments are performed for a specific k, see vertical dashed lines, and the rest of values are calculated accordingly. For SF8, $d = 28$, DC provides up to $1.84\times$ and $2.49\times$ better decoding ratio compared to RS-ECC and No-Added-ECC, respectively.

The main source of energy consumption in an end device is data transmission. Energy per transmission can be calculated by multiplying the transmission power by the frame time on-air, which depends on the used SF, CR, and bandwidth [1]. Transmission energy per bit in Fig. 13 is measured as

$$E = \frac{X10^{-6}N_T}{8kN_D}[\mu\text{J}], \tag{7}$$

where X is the energy consumed per transmission, N_T the number of transmitted frames, and N_D the number of decoded payloads. Thus, Eq. 7 indicates the average energy consumed to receive one *data* bit, correctly, by the decoder. If a frame is not correctly received, it is re-transmitted. Eq. 7 includes $\frac{N_T}{N_D}$ to capture this effect of frame losses. DC and RS-ECC include error-correcting bytes, which lead to extra energy consumed compared to No-Added-ECC for a given k. Furthermore, the $8 \cdot k$ data bits in a frame carry the main information. For SF8, $d = 28$ and SF8, $d = 18$, CR = 4/5, with decreasing $\frac{k}{k+t}$, transmission energy initially decreases as the probability of decoding increases. Subsequently, it increases as k decreases while there is a fixed overhead per frame coming from error-correcting bits, preamble, and physical header. The latter one is the dominant factor for SF10, $d = 22$ and SF10, $d = 18$, CR = 4/7. For SF8, $d = 18$, CR = 4/5, DC consumes $1.49\times$ and $1.99\times$ less energy for correctly transmitting each data bit compared to RS-ECC and No-Added-ECC, respectively. Fig. 13 also shows the total number of k-combinations ($= C(k+t, k)$) per case. This is an indication of the throughput of the decoder. The total number of k-combinations peaks at $k = t$, i.e., $\frac{k}{k+t} = 0.5$.

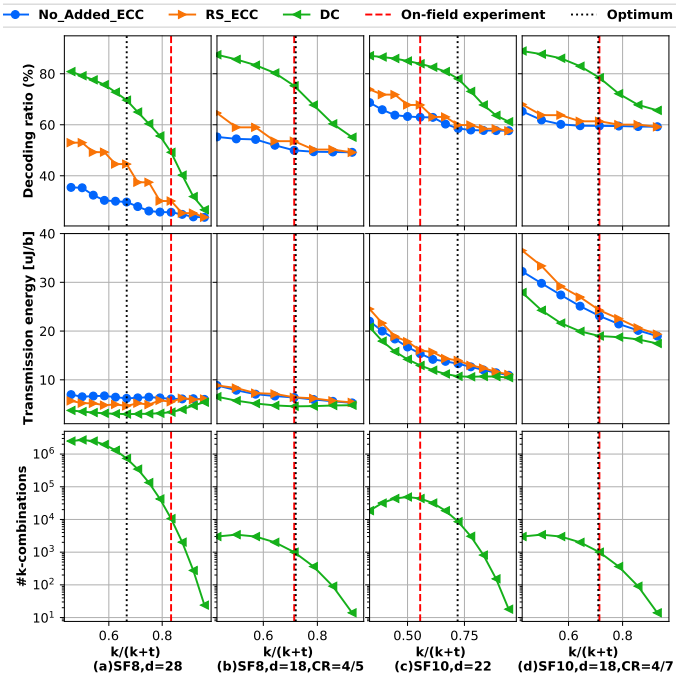

Figure 13: Optimum value for $\frac{k}{k+t}$ considering a fixed $k+t$. As $k+t$ is constant, the payload size for RS-ECC and DC is constant and equal to $d = l + k + t$ for each case. The payload size for No-Added-ECC chooses k with $t = 0$.

Overall, to minimize the transmission energy while keeping a relatively high decoding ratio, $\frac{k}{k+t}$ should be chosen close to 0.7 (with $k+t$ fixed). Therefore, for a fixed d while $l = 4$ bytes are allocated to the CRC, roughly 70% of the remained bytes should be data and 30% error correcting bytes to minimize the transmission energy and keep a high decoding ratio.

VI. Related Works

Marcelis *et al.* introduced *DaRe*, a LoRaWAN application layer coding scheme to retrieve lost data. DaRe combines convolutional and fountain codes, recovering 99% of data using CR = 4/8 at 40% frame loss [9]. Sandell and Raza showed that the benefit of adding more redundancy is bounded due to the interference caused by the increased size of the encoded frames [15]. Thus, DaRe will not work for high frame losses [15]. DC excels in such cases by recovering frames. Indeed, DC coding complements DaRe scheme to retrieve more data by providing more uncorrupted (received) frames at the application layer. DaRe requires reception of multiple frames, whereas DC applies to only one corrupted frame at a time. Montejo-Sánchez *et al.* transmit the encoded redundancy in independent frames, whose number is decided based on range, configuration, and QoS requirements [16]. Elshabrawy and Robert proposed a non-binary Single Parity Check (SPC) code with soft-decision decoding, trading off

the increase in coding gains with increased Time on Air. The optimal application of the SPC code rate enhances the capacity of LoRa networks up to 65% [17]. They showed a reduction in Bit Error Rate (BER) by applying Bit Interleaved Coded Modulation (BICM) considering Rayleigh fading and AWGN channels and gain up to 8 dB in BER [18].

Coutaud *et al.* designed LoRaFFEC wherein frames are encoded using pseudo-random linear combinations of already sent data. Combined with data-fragmentation –to cater to the variety of LoRa-frame sizes– LoRaFFEC manages a data delivery ratio of 98% for channels with 0.4 probability of frame error [19]. Further, Coutaud and Tourancheau propose CCARR, which encodes frames using RS-coding, adapting the size of the added redundancy dynamically [20]. Borkotoky *et al.* suggest windowed and selective coding at the application layer of delay-intolerant LoRaWANs with minimum feedback. Windowed encoding accounts for all the non-delivered and non-expired transmitted symbols, while a selective mechanism chooses a few among them according to feedback [21]. Wang *et al.* focus on convolutionally encoded frames received on high SNR values. They design algorithms assisted by the outcome of CRC: (i) Partial Iterative Decoding-Detecting (P-IDD) retrieves possible errors on bits with the highest log-likelihood of being erroneous, (ii) Soft-Decision Syndrome Decoding (SDSD) technique identifies patterns of errors in frames [22]. Sant'Ana *et al.* propose a hybrid coding scheme, comprised of packet-replication and the use of linear XOR operations for the extension of battery lifetime [23]. Chen *et al.* use Luby Transform (LT) codes on multiple versions of the same frame received by several gateways to recover the correct parts of the frame [24]. Angelopoulos *et al.* apply rateless encoding on frames of k symbols before transmission. Upon reception, they evaluate the algebraic consistency of frames by applying the Algebraic Consistency Rule (ACR) to $k+1$ of their symbols. When corrupted symbols are spotted, they are recovered through an iterative decoding algorithm assisted by CRC [25]. DC builds on this idea to provide a novel, lightweight coding scheme for real systems.

VII. Conclusions

On our LoRa testbed, we measured large portions of corrupted LoRa-frames being received under challenging conditions, e.g., for an SNR$\in [-16,-19]$ dB one-third of LoRa-frames of 22 B at SF10 have errors. Our tests show corruptions in bursts of up to 5 or 6 bytes, with 70%-80% probability for a byte to be erroneous when located next to a corrupted byte. The inbuilt Hamming codes in LoRa cannot repair these frames, leading to data loss and frequent (re)transmissions.

In this work, we introduced Divide & Code, a novel, real-time, coding scheme for LoRaWAN. DC encodes independently and proactively the LoRa-payloads before the CR of LoRa-FEC (or DaRe like schemes) is applied, increasing the robustness of transmitted frames. The decoder of DC incorporates our findings regarding the locations of errors and reduces decoding time and complexity by setting time thresholds and defining subgroups of data bytes within large frames. On-field experiments showed that targeted error correction around erroneous bytes can decode 79.24% of the corrupted frames while trying only 0.20% of k-combinations at SF8 for 20 B payloads. Further, we compared with RS-coding and vanilla LoRaWAN and showed that DC boosts the decoding ratio by $\approx$ 2x and 2.5x while consuming 1.5x and $\approx$ 2x less energy per correctly received data-bit. DC does not require any deviation from the LoRaWAN standard and/or change of infrastructure. DC has no dependency on data from multiple frames to encode/decode, thus supporting time-critical applications. Finally, DC sustains low network traffic, operating without ACKs and adding minimal redundancy per frame.

References

[1] Semtech, "LoRa® and LoRaWAN®: A Technical Overview."
[2] LoRa™ Alliance, "LoRaWAN® Specification v1.1."
[3] B. Reynders, W. Meert, and S. Pollin, "Power and spreading factor control in low power wide area networks," in *IEEE Int. Conf. on Communications (ICC)*, 2017, pp. 1–6.
[4] M. Bor and U. Roedig, "LoRa Transmission Parameter Selection," in *Int. Conf. on Dist. Comp. in Sensor Sys. (DCOSS)*, 6 2017, pp. 27–34.
[5] "The Things Network," https://www.thethingsnetwork.org/.
[6] A. Rahmadhani and F. Kuipers, "Understanding collisions in a lorawan," *SURF Wiki*, 2017.
[7] O. Bernard, A. Seller, and N. Sornin, "Low power long range transmitter," in *Semtech Corporation, Application No. 13154071.8/EP20130154071, Publication No. EP2763321A1*, 2015.
[8] X. Xia, Y. Zheng, and T. Gu, "FTrack: Parallel Decoding for LoRa Transmissions," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys '19. ACM, 2019, p. 192–204.
[9] P. Marcelis, N. Kouvelas, V. S. Rao, and V. Prasad, "DaRe: Data Recovery through Application Layer Coding for LoRaWAN," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.
[10] A. Rahmadhani and F. Kuipers, "When LoRaWAN Frames Collide," ser. WiNTECH '18. ACM, 2018, p. 89–97.
[11] N. Yazdani, N. Kouvelas, R. V. Prasad, and D. E. Lucani, "Energy Efficient Data Recovery from Corrupted LoRa Frames," *arXiv*, 2021.
[12] Bosch, "Market size and connected devices: Where's the future of IoT?"
[13] S. B. Wicker and V. K. Bhargava, *Reed-Solomon codes and their applications*. John Wiley & Sons, 1999.
[14] S. Pemmaraju and S. Skiena, *Computational discrete mathematics: Combinatorics and graph theory with mathematica®*. Cambridge university press, 2003.
[15] M. Sandell and U. Raza, "Application layer coding for iot: Benefits, limitations, and implementation aspects," *IEEE Systems Journal*, vol. 13, no. 1, pp. 554–561, 2019.
[16] S. Montejo-Sánchez, C. A. Azurdia-Meza, R. D. Souza, E. M. G. Fernandez, I. Soto, and A. Hoeller, "Coded Redundant Message Transmission Schemes for Low-Power Wide Area IoT Applications," *IEEE Wireless Comms. Letters*, vol. 8, no. 2, pp. 584–587, 2019.
[17] T. Elshabrawy and J. Robert, "Enhancing LoRa Capacity using Non-Binary Single Parity Check Codes," in *Int. Conf. on Wireless and Mobile Comp., Networking and Comm. (WiMob)*, 2018, pp. 1–7.
[18] ——, "Evaluation of the BER Performance of LoRa Communication using BICM Decoding," in *IEEE ICCE*, 2019, pp. 162–167.
[19] U. Coutaud, M. Heusse, and B. Tourancheau, "Fragmentation and forward error correction for lorawan small mtu networks," ser. ACM EWSN '20. Junction Publishing, 2020, p. 289–294.
[20] U. Coutaud and B. Tourancheau, "Channel Coding for Better QoS in LoRa Networks," in *IEEE WiMob*, 2018, pp. 1–9.
[21] S. S. Borkotoky, U. Schilcher, and C. Raffelsberger, "Application-Layer Coding with Intermittent Feedback Under Delay and Duty-Cycle Constraints," in *IEEE ICC*, 2020, pp. 1–6.
[22] R. Wang, W. Zhao, and G. B. Giannakis, "CRC-assisted error correction in a convolutionally coded system," *IEEE Trans. on Comm.*, vol. 56, no. 11, pp. 1807–1815, 2008.
[23] J. M. d. S. Sant'Ana, A. Hoeller, R. D. Souza, S. Montejo-Sánchez, H. Alves, and M. d. Noronha-Neto, "Hybrid Coded Replication in LoRa Networks," *IEEE TII*, vol. 16, no. 8, pp. 5577–5585, 2020.
[24] G. Chen, J. Lv, and W. Dong, "Exploiting Rateless Codes and Cross-Layer Optimization for Low-Power Wide-Area Networks," in *IEEE/ACM IWQoS*, 2020, pp. 1–9.
[25] G. Angelopoulos, A. P. Chandrakasan, and M. Médard, "PRAC: Exploiting partial packets without cross-layer or feedback information," in *IEEE Int. Conf. on Communications (ICC)*, 2014, pp. 5802–5807.