Revolving Codes: Overhead and Computational Complexity Analysis

Niloofar Yazdani, *Member, IEEE,* and Daniel E. Lucani, *Senior member, IEEE,*

***Abstract*—Revolving codes (ReC) are a new family of network codes that reduce signaling overhead and maintain high decoding probability, which is key in applications with small payloads, e.g., IoT, Industry 4.0. However, they have only been studied using simulations. We present i) the first exact mathematical model for the total overhead and decoding probability of a set of ReCs, and ii) efficient encoders and decoders. Our analysis shows that ReC decoders' number of multiplications scales as $O(g)$ where g is the number of data packets being combined. In practice, this translates in several orders of magnitude fewer multiplications, half the number of additions, and lower transmission overheads compared to RLNC.**

***Index Terms*—Network coding, complexity, overhead.**

I. Introduction

NETWORK coding (NC) is a networking technique that can increase a network's throughput and reliability [1]. Random Linear Network Coding (RLNC) [2] provides a practical and distributed approach to NC, where network nodes transmit random linear combinations of previously received packets by using coefficients chosen uniformly at random from a finite field. However, RLNC's integration into large scale systems and products has been curbed by a number of factors. First, RLNC's computational complexity grows significantly as the number of data packets being combined (g) grows. In fact, the number of encoding and recoding operations grow as $O(g^2)$ while its decoding operations grow as $O(g^3)$. Second, the signaling overhead to communicate the random coefficients used in each coded packet grows with g. This signaling overhead is higher for the packets with small payload. This high complexity and high overhead of RLNC can increase energy consumption in network and end devices, e.g., battery-powered Wireless Sensor Networks (WSN) [3], may create additional latency [4], reduce the efficiency of applications with short-packet transmissions, e.g., Internet of Things (IoT), Industry 4.0, or even become the bottleneck on high-speed wireless networks [5]. Although reducing g is an alternative, this reduction increases the transmission overhead due to a higher probability to generate linearly dependent packets and reducing the overall benefits of RLNC [6].

Sparse coding techniques such as sparse RLNC [7] have been proposed to decrease the computational complexity at the cost of larger communication overhead [5]. However, sparse techniques typically introduce additional complexity in the recoding process to maintain a sparse structure. Techniques that provide simple recoders and reasonable signaling overhead have been proposed, e.g., BATS [8], but which operate effectively for transmission of large number of data, or batch codes [9] that perform particularly well for generation size of 256 and above. Fulcrum codes [10] are an alternative that increase processing speed of encoders and decoder by a constant factor and reduce the signaling overhead for each transmitted packet to roughly 1-bit per original packet.

More recently, revolving codes (ReC) [11] were proposed as an alternative approach to reduce the signaling overhead per packet, overhead due to linearly dependent packets while preserving a simple RLNC recoder (similar to Fulcrum codes). ReC was shown to outperform RLNC by as much as two orders of magnitude in terms of total overhead and deliver two orders of magnitude lower overhead due to linearly dependent packets compared to Fulcrum codes [11]. However, the work in [11] only analyzed ReC's performance by simulations and did not prove performance mathematically. In this paper, we characterize a family of ReCs using a Markov chain, which fully describes its decoding probability and overhead performance. Moreover, we propose encoding and decoding algorithms that exploit ReCs code structure. We show that the number of finite field multiplications for the decoder are reduced from $O(g^3)$ in RLNC to $O(g)$ in ReC. Furthermore, addition and multiplication operations in the encoder are reduced by half. For small payloads, our ReC decoder requires 251 and 2 times less multiplications compared to RLNC and Fulcrum with 4 extra dimensions, respectively.

II. Basics of Revolving Codes

A $ReC(q, b, t)$ is a revolving code [11] over a finite field $\mathbb{F}_q$, $q = 2^m$, which flips b bits per coefficient with respect to a common value selected per transmitted coded packet and signals the coding coefficient information by transmitting the q-bit common value and t bits per coefficient. For a generation of g data packets $P_1, ..., P_g$, the k-th coded packet is given by $X_k = \sum_{i=1}^{g} c_{k,i} P_i$ where $c_{k,i}$ is the coding coefficient of the i-th data packet (P_i). For ReC, $c_{k,i}$ is a combination of a common value (CV_k) selected for X_k and an individual value for each coefficient ($IV_{k,i}$). CV_k is chosen uniformly at random from elements in $\mathbb{F}_q$. Each $IV_{k,i}$ is a random element of b bits (e.g., $b = 1, 2$ in [11]). Let $IV_{k,i}^{*i-1}$ be $IV_{k,i}$ shifted circularly in a register of $\log_2(q)$ bits, $i-1$ times by b bits. Then, $c_{k,i} = CV_k + IV_{k,i}^{*i-1}$. In contrast, RLNC chooses $c_{k,i}$ uniformly at random from $\mathbb{F}_q$. An $ReC(2^m, b, t)$ can be recoded by intermediate nodes in the network using RLNC recoders over $\mathbb{F}_2$ [11].

III. Mathematical Analysis of $ReC(q, 1, 1)$

To develop the mathematical model and the encoding/decoding algorithms of $ReC(q, 1, 1)$, the core step is to expand $X_k = \sum_{i=1}^{g} c_{k,i} P_i$ as

$$X_k = CV_k \cdot \sum_{i=1}^{g} P_i + \sum_{i=1}^{g} IV_{k,i}^{*i-1} P_i. \quad (1)$$

Niloofar Yazdani and Daniel E. Lucani are with DIGIT and Department of Engineering, Aarhus University, Denmark.{n.yazdani, daniel.lucani}@eng.au.dk

This work was supported in part by the SCALE-IoT Project (Grant No. 7026-00042B) granted by the Danish Council for Independent Research, the Aarhus Universitets Forskningsfond Starting Grant Project AUFF- 2017-FLS-7-1.

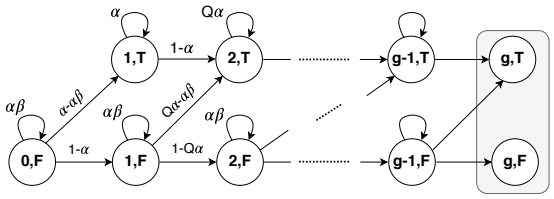

Fig. 1: Markov chain model of decoding.

These two components are linearly independent except for the case where all coefficients are zero. This is due to the fact that $c \cdot \sum_{i=1}^{g} P_i \neq \sum_{i=1}^{g} IV_{k,i}^{*i-1} P_i$ for a non-zero constant c in the finite field. For example, if all individual values are chosen as 1, then $\sum_{i=1}^{g} IV_{k,i}^{*i-1} P_i = P_1 + 2P_2 + 4P_3 + ... + 1^{*g-1} P_g$.

This means that $CV_k \cdot \sum_{i=1}^{g} P_i$ provides one degree of freedom that cannot be achieved by the second component (C2) of Eq. (1), i.e., $\sum_{i=1}^{g} IV_{k,i}^{*i-1} P_i$, which uses only a very limited set of coefficient values. However, this degree of freedom can only be used once during the decoding process. At the decoder, the benefit of the first component (C1) of Eq. (1) is key the first time that the vector of individual values (i.e., C2) of a coded packet is linearly dependent to the vector of individual values of the previous coded packets at the decoder. However, after the C1 has been used during the decoding process at some point, it will not bring any benefits in the future if the C2 is linearly dependent of previously received coded packets.

Markov Chain Model: We consider an absorbing Markov chain [12] with states $\mathcal{S} = \big((0,F),(1,F),(1,T),(2,F),(2,T),...,(g,F),(g,T)\big)$ as in Fig. 1. Each pair represents the number of linearly independent packets received and whether the C1 has been used in the decoding previously. If a state is of the form $(.,T)$ it has already been used and $(.,F)$ otherwise. The start state is $(0,F)$, while the *absorbing* states (i.e., when data can be decoded) are (g,F) and (g,T). For simplicity, we can consider them as a single absorbing state.

Transition probability: Let us define $Q = 2^b = 2$, $\alpha = \frac{1}{Q^g}$, $\beta = \frac{1}{q}$, and $p_{ss'}$ as the transition probability from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$. Then,

$$p_{(x,A)(x',A')} = \begin{cases} \alpha.\beta & x' = x,\ A' = A = F \\ 1 - Q^x\alpha & x' = x+1,\ A' = A = F \\ Q^x\alpha - \alpha\beta & x' = x+1,\ A = F,\ A' = T \\ Q^{x-1}\alpha & x' = x,\ A' = A = T \\ 1 - Q^{x-1}\alpha & x' = x+1,\ A' = A = T \end{cases} \quad (2)$$

The first term represents the probability of $CV_k = 0$ and $IV_{k,i} = 0$ for all i's, which is the only case where C1 and C2 in Eq. (1) would be linearly dependent. Since the probability that an incoming coded packet has a C2 in Eq. (1) that is linearly dependent with respect to the C2 of the x previously received packets is given by $Q^x\alpha$, then $p_{(x,F)(x+1,F)} = 1 - Q^x\alpha$, which corresponds to the second term. The third term comes from the fact that $p_{(x,F)(x+1,T)} = 1 - \Big(p_{(x,F)(x,F)} + p_{(x,F)(x+1,F)}\Big)$. The fourth and fifth terms are the resulting of using the C1 in (1) once before during the decoding process. Thus, only $x-1$ independent linear combinations can be attributed to come from the C2 in Eq. (1).

Expected time to absorption: The expected number of coded packets received before decoding is equivalent to the expected absorption time [12]. Starting from a particular transient state, for any state s, μ_s is the expected number of transitions from s until absorption, and $\mu_s = 1 + \sum_{s' \in \mathcal{S}} p_{ss'}\mu_{s'}$. Note that $\mu_{(g,T)} = 0$ and $\mu_{(g,F)} = 0$. The expected time to absorption from any state can fully characterized by

$$\mu_{(x,F)} = \frac{1 + p_{(x,F)(x+1,T)}\mu_{(x+1,T)} + p_{(x,F)(x+1,F)}\mu_{(x+1,F)}}{1 - p_{(x,F)(x,F)}},$$
$$\mu_{(x,T)} = \frac{1 + p_{(x,T)(x+1,T)}\mu_{(x+1,T)}}{1 - p_{(x,T)(x,T)}}. \quad (3)$$

The expected number of coded packets received before decoding results from calculating $\mu(0,F)$.

Since ReC is intended to use large finite fields, but operate in part as small ones [11], we assumed the finite field is large enough to have a minimum of one useful common value. For instance, for $q = 2^8$, if the vector of individual values for the current coded packet is linearly dependent to the vector of individual values of the previous coded packets, the probability of none of them having a useful common value is at least as low as $\frac{1}{2^8 \cdot 2^8}$ which is negligible. In Sec. IV, we will show that the cost of using a high finite field is not that important because of the low number of multiplications.

Overhead Analysis: We consider two key metrics. First, the dependency overhead ratio, which captures the overhead of transmitting additional coded packets due to linear dependencies at the receiver, is $RO_D\% = \frac{\mu(0,F)-g}{g} \times 100\%$. Fig. 2 (a) shows that the RO_D computed by simulations and by our analytical model for $ReC(2^8,1,1)$ for different g values have a good match. Second, the total overhead computed as $O_T = g \cdot \lceil \frac{g+m}{8} \rceil + (\mu(0,F) - g) \cdot (p + \lceil \frac{g+m}{8} \rceil)$ [Bytes], for p bytes of payload per data packet. This metric captures the signaling overhead. Fig. 2 (b) shows the total overhead ratio, $RO_T\% = \frac{O_T}{g \cdot p}$ for packet size of 1.6 kB and $q = 2^8$.

As shown in Fig. 2, $ReC(2^8,1,1)$ outperforms Fulcrum with two extra dimensions, i.e., $r = 2$, in terms of dependency and total overhead by up to 1.47 times and 1.39 times, respectively. Fulcrum with $r = 4$ outperforms $ReC(2^8,1,1)$. However, as it will be shown in Sec. IV, it is achieved at the cost of higher complexity for decoding. $ReC(2^8,1,1)$ outperforms RLNC in terms of total overhead by up to 7.8 times for $g > 16$. While, it provides up to several orders of magnitude better dependency and total overhead than sparse RLNC with sparsity of 0.9 (90% of coefficients are zero). Finally, $ReC(2^8,1,1)$ outperforms sparse RLNC with sparsity of 0.5 for most of the region in terms of total overhead by up to 13.5 times. Note that dependency overhead for $ReC(2^8,2,2)$ is as low as the one for RLNC over $q = 2^8$.

IV. Encoding and Decoding Algorithms

A. Encoder

Considering Eq. 1, the encoder carries out four steps indicated in Table. I. First, it computes an addition of all data packets, each with S symbols in $\mathbb{F}_q$ of payload, each symbol drawn from the finite field. Second, it multiplies this result by the CV_k selected for coded packet X_k to generate the C1 in (1).

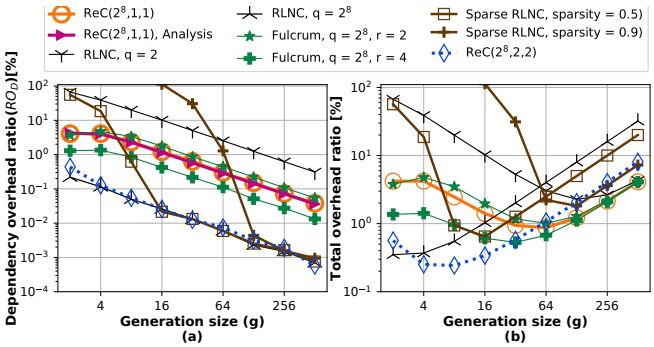

Fig. 2: Dependency and total overheads.

TABLE I: $ReC(2^m, 1, 1)$: Encoding Operations.

#	Operation	Add's	Mult's	Times Used
1	$X = \sum_{i=1}^{g} P_i$	$S(g-1)$	0	1
2	$X_{k,1} = CV_k.X$	0	S	1
3	$X_{k,2} = \sum_{i=1}^{g} IV_{k,i}^{*i-1} P_i$	$S(g\mu - 1)$	$Sg\mu\gamma$	1
4	$X_{k,1} + X_{k,2}$	S	0	1

Third, it computes the C2 in (1) by multiplying the packets only with the non-zero $IV_{k,i}^{*i-1}$'s and adding those together. We denote μ as the ratio of non-zero $IV_{k,i}^{*i-1}$'s when generating the k-th coded packet. We also define γ as the ratio of $IV_{k,i}^{*i-1}$'s that cannot be one i.e., $i \neq 1$. Thus, $\gamma = 1 - \frac{\lceil \frac{g}{m} \rceil}{g}$. Fourth, it adds the C1 and C2 generated in steps 2 and 3. Step 1 is done once for generating all encoded packets.

Lemma 1. *The expected number of additions and multiplications for $ReC(2^m, 1, 1)$ encoding are $\mathcal{A}_{ReC,e} = \frac{S}{2}g^2 + Sg - S$ and $\mathcal{M}_{ReC,e} = \frac{S}{2}\gamma g^2 + Sg$, respectively, with $0 < \gamma < 1$.*

Proof. Follows a counting argument using Table I and using the fact that $E[\mu] = 1/2$. Note that we consider the most common case of having g packets. □

The following two corollaries come from the fact that the total number of additions and multiplications for RLNC are $Sg^2 - Sg$ and Sg^2, respectively [4]. Using the same counting argument, Fulcrum's encoder with r extra dimensions requires a total of $Sg^2 + (r - S)g - r + S$ and rgS additions and multiplications, respectively.

Corollary 1.1. *For $g \gg S$, (a) $ReC(2^m, 1, 1)$ reduces the number of additions and multiplications by a factor of 2 and $\frac{2}{\gamma}$, respectively, with respect to RLNC; (b) $ReC(2^m, 1, 1)$ reduces the number of additions by a factor of 2, with respect to Fulcrum assuming $r \ll S$; (c) Due to considering a systematic outer code for Fulcrum, Fulcrum reduces the number of multiplications by a factor of $\frac{\gamma g}{2r}$, compared to $ReC(2^m, 1, 1)$.*

Corollary 1.2. *For $g \ll S$, (a) $ReC(2^m, 1, 1)$ reduces the number of additions and multiplications by a factor of $\frac{g^2-g}{\frac{g^2}{2}+g-1}$ and $\frac{g^2}{\frac{\gamma}{2}g^2+g}$, respectively, compared to RLNC; (b) $ReC(2^m, 1, 1)$ reduces the number of additions and multiplications by a factor of $\frac{g^2-g+1}{\frac{g^2}{2}+g-1}$ (assuming $r \ll g$) and $\frac{r}{1+\frac{\gamma g}{2}}$, respectively, with respect to Fulcrum.*

B. Decoder

The decoder experiences two cases: (Case 1) all vectors of individual coefficients are linearly independent until data is decoded, and (Case 2) at least one vector of individual coefficients during the reception process was linearly dependent. In the latter, we use the C1 in (1) to provide an independent linear combination overall for the receiver the first time a linearly dependent individual coefficient vector is received.

Our decoder will exploit the structure of (1) to consider the C1 as an independent variable during the first stages of decoding. This means that a system with g original packets, will generate a decoder that acts as if there were g+1 variables. In other words, the system of linear equations considered for decoding is of the form

$$\begin{bmatrix} IV_{1,1}^{*0} & IV_{1,2}^{*1} & \dots & IV_{1,g}^{*g-1} & CV_1 \\ \vdots & & \ddots & & \\ IV_{g,1}^{*0} & IV_{g,2}^{*1} & \dots & IV_{g,g}^{*g-1} & CV_g \end{bmatrix} \times \begin{bmatrix} P_1 \\ \vdots \\ P_g \\ P_1 + \dots + P_g \end{bmatrix} = \begin{bmatrix} X_1 \\ \vdots \\ X_g \end{bmatrix}. \quad (4)$$

The goal is to (a) isolate the effect of CV_k of each coded packet X_k, and (b) take advantage of the structure of the individual values, which only modify a limited number of bits. The decoder is exploiting the structure by thinking of the problem as having a *virtual precoding* while there is not a real precoding mechanism. The decoder will still need to decode g elements. The specific bits modified in each coefficient is the same across different coded packets. This can simplify the operations during decoding to only apply additions in the $\mathbb{F}_{2^m}$.

Example: Let us consider a simple example with $g = 4$ over $\mathbb{F}_{2^8}$ to illustrate this (Fig. 3. (a)). If $CV_1 = 100$ (i.e., a binary sequence of $(01100100)_2$) and $IV_{1,1} = IV_{1,2} = IV_{1,4} = 1$ and $IV_{1,3} = 0$. The standard view of coefficients would be $c_{1,1} = 101$, $c_{1,2} = 102$, $c_{1,3} = 100$, and $c_{1,1} = 108$, as $IV_{1,1}^{*0} = 1 = (00000001)$, $IV_{1,2}^{*1} = 2 = (00000010)$, $IV_{1,3}^{*2} = 0 = (00000000)$ and $IV_{1,4}^{*3} = 8 = (00001000)$ because $c_{k,i} = CV_k + IV_{k,i}^{*i-1}$ as stated in Sec. II. If we use Eq. (4), this gives us the first row of Fig. 3. (a). In a similar way, if $CV_2 = 120$ and $IV_{2,1} = IV_{2,3} = IV_{2,4} = 1$ and $IV_{2,2} = 0$, we get the second row of Fig. 3. (a).

After receiving $g = 4$ coded packets, we can initiate the decoding process of Fig. 3. (a). To eliminate the effect of the left-most column, a bit-by-bit XOR of row 1 to rows 2 and 3 is sufficient. Although this operation changes the values of the last column (corresponding to the common value), the remaining columns maintain values that are either zero or the corresponding non-zero value of the specific column. For example, column 2 has value 2 and column 3 has value 4. If we eliminate the effect of the second column in rows below and later on rows above in backward elimination using the second row as pivot, we see a similar behavior. This continues for all columns. For the second step, we eliminate the effect of each row on rows above in a backward path. For the third step in Fig. 3. (a), there are only non-zero elements in the diagonal of the first four columns. We can then multiply by the inverses of each corresponding element in the diagonal. Then, we perform an XOR bit-by-bit of all four rows. This yields an equation of the form $\sum_{k=1}^{4} P_k(1 + \sum_{k=1}^{4} CV'_k)$, where the addition is a bit-by-bit XOR and CV'_k is the modified value of the last column after the elimination process. For example, $CV'_2 = 46$. This allows us to solve for the value of $\sum_{k=1}^{4} P_k$ and eliminate the effect of the last column. Because $\sum_{k=1}^{4} P_k$ can be considered

TABLE II: $ReC(2^m, 1, 1)$: Decoding operations for case 1.

#	Operation	Add's	Mult's	Times Used
1	$R_j \leftarrow R_j + R_k$	$g+1+S-k$	0	$(g-k).\mu$
2	$R_k \leftarrow R_k + R_j$	$1+S$	0	$(g-k).\mu$
3	$R_k \leftarrow c'_{k,k}.R_k$	0	$(1+S)$	γ
4	$R = \sum_{i=1}^{g} R_i$	$1+(g-1).$ $(1+S)$	0	1
5	$R \leftarrow c'_{g+1}.R$	0	S	1
6	$R_i \leftarrow R_i + dR$	S	S	1

TABLE III: $ReC(2^m, 1, 1)$: Decoding operations for case 2.

#	Operation	Add's	Mult's	Times Used
1	$R_j \leftarrow R_j + R_k$	$g+1+S-k$	0	$(g-k).\mu$
2	$R_k \leftarrow R_k + R_j$	$S+2$	0	$(g-1-$ $k).\mu$
3	$R_k \leftarrow c'_{k,k}.R_k$	0	$(2+S)$	γ
4	$R_g \rightarrow c'_{g+1}.R_g$	0	S	1
5	$R_k \rightarrow R_k + c'_{g+1} \cdot R_g$	S	S	1
7	$R_g \leftarrow R_g + R_k$	$S+1$	0	1
8	$R_g \leftarrow c'_g \cdot R_g$	0	S	1
9	$R_k \leftarrow R_k + c'_{k,g+1} R_g$	S	S	μ

as the value for another row with elements equal to $[00001]$. This yields to the first case's decoding.

For the second case, the process corresponds to Fig. 3. (b). The early steps are similar. The key difference is that the final row does not have a non-zero element in the first four columns after the elimination process. Thus, its effect cannot be eliminated on other rows at step 2. In this case, finding $\sum_{k=1}^{4} P_k$ consists only in multiplying the last row by the inverse of the remaining element (i.e., multiply by 52^{-1}). After this, we can eliminate the effect on the right-most column in all rows. Then, we can convert the 1 element in the right-most column (last row) into a a row with all ones in the first four columns and zero in the right-most column (i.e., re-interpreting $\sum_{k=1}^{4} P_k$). We proceed to finalize the elimination process until obtaining an identity matrix in the first four columns.

The formal algorithm for the general case is detailed in algorithm 1, where we use R_x to denote the x-th row of the coding coefficient matrix. We also use $R_{x,y}$ and $R'_{x,y}$ to denote the y-th entry of R_x and its inverse, respectively.

Lemma 2. *The mean number of additions and multiplications are*

$$\mathcal{A}_{ReC,d} = \frac{1}{6}g^3 + (\frac{S}{2} + e_1)g^2 + (\frac{3S}{2} + e_2)g - e_3.S, \text{ and} \quad (5)$$

$$\mathcal{M}_{ReC,d} = ((e_4 + \gamma)S + e_5.\gamma)g + e_6.S + e_7, \quad (6)$$

respectively. For case 1, $e_1 = \frac{1}{4}$, $e_2 = \frac{7}{12}$, $e_3 = 1$, $e_4 = e_5 = e_6 = 1$, *and* $e_7 = 0$, *while for case 2,* $e_1 = \frac{1}{2}$, $e_2 = \frac{-2}{3}$, $e_3 = 2$, $e_4 = \frac{3}{2}$, $e_5 = 2$, $e_6 = \frac{1}{2} - \gamma$, *and* $e_7 = -2\gamma$.

Proof. The proof follows by a counting argument, considering the various steps of the algorithms for case 1 and case 2 and using the fact that $E[\mu] = 1/2$. We assume the computational complexity of finding the inverse of a symbol for determining the coefficients is negligible due to using a relatively small lookup table. Note that swapping the rows, if required, is effectively instantaneous if implemented efficiently, e.g., using pointers. We do not perform the operations which outcome is already known by the structure of the algorithm. This includes: a) In step 1, the first $k-1$ entries of R_j are 0 and k-th entry will be 0, b) In step 2, outcome for the first g and $g-1$ entries for cases 1 and 2, respectively, are known. c) In step 3, $c'_{k,k} \neq 1$ by a fraction of γ. Thus, for a fraction of $1-\gamma$, we do not need to do the multiplication. Furthermore, outcome for the first g and $g-1$ entries for cases 1 and 2, respectively, are known. d) In step 4, case 1, We already know the first g entries of $\sum_{i=1}^{g} R_i$ are 1. However, we assume the first g entries of R are 0 and we add 1 to the $(g+1)$-st entry, instead. e) Step 5, case 1, we know the outcome for first $g+1$ entries. f) Step 5, case 2, we know the outcome for first $g+1$ entries.

We also consider using a high $\mathbb{F}_q$ where packets are independent and g packets are enough for decoding. Tables II and III provide a breakdown of the number of operations per step of the algorithm for Case 1 and Case 2, respectively. □

Algorithm 1: $ReC(2^m, 1, 1)$ decoder:

```
Step 1: Forward Elimination
for k = {1, 2, ..., g − 1} do
    if R_{k,k} = 0 then
        R_k ↔ R_{k'}        // Swap the rows. k' > k, R_{k',k} ≠ 0
    for j = {k + 1, ..., g} do
        if R_{j,k} ≠ 0 then
            R_j ← R_j + R_k
Step 2: Backward Elimination
g' = g
if Case 2 then
    g' = g − 1
for k = {1, 2, ..., g − 1} do
    for j = {k + 1, ..., g'} do
        if R_{k,j} ≠ 0 then
            R_k ← R_j + R_k
Step 3: Multiply by Inverse for k = {1, 2, ..., g'} do
    R_k → R'_{k,k}.R_k
if Case 1 then
    Step 4: Compute Addition of all rows
    R ← Σ_{i=1}^{g} R_i
    Step 5: Multiply by Inverse
    R → R'_{R,g+1}.R     // R'_{R,g+1}: (g + 1)-st entry's inverse
    Step 6: Forward Elimination
    for k = {1, 2, ..., g} do
        R_k → R_k + R_{k,k+1} · R
else if Case 2 then
    Step 4: Multiply by Inverse
    R_g → R'_{g,g+1}.R_g Step 5: Forward Elimination
    for k = {1, 2, ..., g − 1} do
        R_k → R_k + c_{k,g+1} · R_g
    Step 6: Replace the first g + 1 entries of R_g from
     0, 0, ..., 0, 1 to 1, 1, ..., 1, 0.
    Step 7: Elimination on R_g
    for k = {1, 2, ..., g − 1} do
        if R_{g,k} ≠ 0 then
            R_g ← R_g + R_k
    Step 8: Multiply by inverse
    R_g ← R'_{g,g} · R_g
    Step 9: Elimination on column g + 1
    for k = {1, 2, ..., g − 1} do
        R_k ← R_k + R_{k,g+1}.R_g
```

Corollary 2.1. *$ReC(2^m, 1, 1)$'s decoder has $O(g)$ multiplications.*

Using a similar counting argument for RLNC as in [13], we show that RLNC has a total of $\mathcal{A}_{RLNC,d} = (\frac{1}{3})g^3 + (S - \frac{1}{2})g^2 + (-S + \frac{1}{6})g$ additions and $\mathcal{M}_{RLNC,d} = \frac{1}{3}g^3 + Sg^2 - \frac{1}{3}g$ multiplications. Using the same counting argument as in this paper for the combined decoder for Fulcrum, it has a total of

$$\begin{bmatrix}1&2&0&8&100\\1&0&4&8&120\\1&2&4&8&50\\0&2&4&8&14\end{bmatrix}\xrightarrow{1}\begin{bmatrix}1&2&0&8&100\\0&2&4&0&28\\0&0&4&0&86\\0&0&0&8&14\end{bmatrix}\xrightarrow{2}\begin{bmatrix}1&0&0&0&118\\0&2&0&0&74\\0&0&4&0&86\\0&0&0&8&14\end{bmatrix}\xrightarrow{3}\begin{bmatrix}1&0&0&0&118\\0&1&0&0&46\\0&0&1&0&104\\0&0&0&1&164\end{bmatrix}\xrightarrow{4,5,6}\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}$$

(a)

$$\begin{bmatrix}1&0&4&0&10\\0&2&4&8&12\\1&2&0&8&50\\0&0&4&0&87\end{bmatrix}\xrightarrow{1,2,3}\begin{bmatrix}1&0&0&0&89\\0&1&0&24&91\\0&0&1&0&87\\0&0&0&0&52\end{bmatrix}\xrightarrow{4,5}\begin{bmatrix}1&0&0&0&0\\0&1&0&24&0\\0&0&1&0&0\\0&0&0&0&1\end{bmatrix}\xrightarrow{6}\begin{bmatrix}1&0&0&0&0\\0&1&0&24&0\\0&0&1&0&0\\1&1&1&1&0\end{bmatrix}\xrightarrow{7}\begin{bmatrix}1&0&0&0&0\\0&1&0&24&0\\0&0&1&0&0\\0&0&0&25&0\end{bmatrix}\xrightarrow{8,9}\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\end{bmatrix}$$

(b)

Fig. 3: $ReC(2^8,1,1)$ decoding example for cases a) 1: vectors of individual values are independent, b) 2: they are not independent. Numbers on the arrows show the step numbers in Tables II&III. S additional augmented columns are not shown.

$\mathcal{M}_{Fulcrum,d} = g\cdot r\cdot(1+\frac{r}{2}+S)-\frac{1}{6}\cdot r^3-\frac{3}{2}\cdot r^2+(\frac{1}{6}-S)\cdot r$ multiplications, and $\mathcal{A}_{Fulcrum,d} = \frac{1}{6}g^3+(\frac{1}{2}S+\frac{1}{2}r-\frac{1}{4})g^2+(r^2+(-\frac{s}{2}-\frac{1}{2})r-\frac{1}{2}S+\frac{1}{12})g+\frac{1}{3}r^3+(\frac{3}{2}S-\frac{1}{2})r^2+(-\frac{1}{2}S+\frac{1}{6})r$ additions. The proofs of the following corollaries are trivial and omitted due to space restrictions.

Corollary 2.2. *For $g \gg S$, $ReC(2^m,1,1)$'s decoder reduces the number of additions by half and the number of multiplications by a factor of $O(g^2)$, Compared to RLNC.*

Corollary 2.3. *For $g \ll S$, $ReC(2^m,1,1)$'s decoder reduces the number of additions and multiplications by factors of $\frac{g^2-g}{\frac{1}{2}g^2+\frac{3}{2}g-e_3}$ and $\frac{g^2}{(e_4+\gamma)g+e_6}$, respectively, with respect to RLNC.*

Corollary 2.4. *Considering $r \ll g,S$, (a) for either $g \gg S$ or $S \gg g$, $ReC(2^m,1,1)$'s decoder reduces the number of multiplications by a factor of $\frac{r}{1+\gamma}$ compared to Fulcrum; (b) $ReC(2^m,1,1)$ has the same number of additions as Fulcrum for $g \gg S$ while it reduces the number of additions by a factor of $\frac{\frac{g^2}{2}-\frac{g}{2}+\frac{3}{2}g-\frac{r}{2}}{\frac{g^2}{2}+\frac{3}{2}g-e_3}$ for $S \gg g$.*

V. Numerical Results

Fig. 4 shows the number of additions and multiplications of encoding and decoding for RLNC, Fulcrum with $r=2$ and $r=4$, and $ReC(2^8,1,1)$, for $\mu=0.5$, $S=100$ B, e.g., IoT data, and $q=2^8$. In the case of decoding for $ReC(2^8,1,1)$ we show the performance of experiencing Case 1 or Case 2. We use as standard naming: the code name (RLNC, Fulcrum, or $ReC(2^8,1,1)$), the algorithm (e.g., Encoder, Decoder), and the type of operation (Additions, Multiplications). For the ReC decoder, the algorithm indicates also if it is Case 1 or Case 2.

As depicted in Fig. 4, $ReC(2^8,1,1)$ reduces the number of additions and multiplications for encoding by half and more than half, respectively compared to RLNC, for high enough g values . Compared to Fulcrum with $r=2$ or $r=4$, $ReC(2^8,1,1)$'s encoder reduces the number of additions by half for high enough g values. Since we considered systematic outer code for Fulcurm, its encoder uses fewer multiplications compared to $ReC(2^8,1,1)$. We analyze some key examples for decoding. $Rec(2^8,1,1)$ requires 252 and 734 times less multiplications for $g=256$ and $g=512$, respectively, for decoding compared to RLNC. The number of multiplications for $ReC(2^8,1,1)$, for $g>8$, is as low as Fulcrum with $r=2$ while around 2 times less than Fulcrum with $r=4$.

VI. Conclusions

We provided the first mathematical model for $ReC(2^m,1,1)$ and validated these results using simulations. Moreover, we proposed new encoding and decoding algorithms to reduce the computational complexity. Our proposed encoder decreases the required operations by half compared with RLNC over the same $\mathbb{F}_q$ and g. Our proposed decoder requires only $O(g)$ multiplication operations compared to $O(g^3)$ in RLNC, while also reducing addition operations by half. Given that multiplication operations in $\mathbb{F}_{2^m}$ are more complex than addition operations, this can provide a significant speed-up with respect to RLNC. Compared to Fulcrum with 4 extra dimensions, our proposed decoder reduces the multiplication operations by half. Future work will focus on developing software implementations.

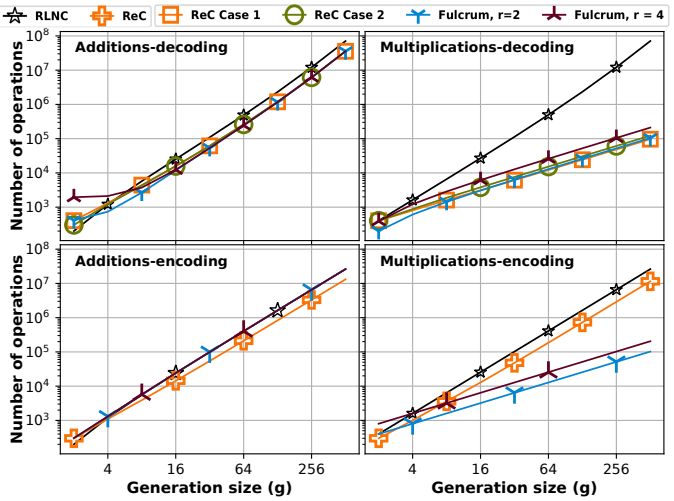

Fig. 4: Number of operations for $q=2^8$ and $\mu=0.5$.

References

[1] M. Wang *et al.*, "Lava: A reality check of network coding in peer-to-peer live streaming," in *IEEE Int. Conf. on Computer Communications (INFOCOM)*. IEEE, 2007, pp. 1082–1090.

[2] T. Ho *et al.*, "A random linear network coding approach to multicast," *IEEE Trans. on Info. Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[3] H. Sehat *et al.*, "An analytical model for rank distribution in sparse network coding," *IEEE Comm. Letters*, vol. 23, no. 4, pp. 556–559, 2019.

[4] L. Nielsen *et al.*, "Latency performance of encoding with random linear network coding," in *European Wireless Conf.* VDE, 2018, pp. 1–5.

[5] J. Heide *et al.*, "Cautious view on network coding—from theory to practice," *Jour. of Comms and Networks*, vol. 10, no. 4, pp. 403–411, 2008.

[6] ——, "On code parameters and coding vector representation for practical RLNC," in *IEEE Int. Conf. on Comms (ICC)*. IEEE, 2011, pp. 1–5.

[7] A. Tassi *et al.*, "Analysis and optimization of sparse random linear network coding for reliable multicast services," *IEEE Transactions on Communications*, vol. 64, no. 1, pp. 285–299, 2015.

[8] S. Yang *et al.*, "Batched sparse codes," *IEEE Transactions on Information Theory*, vol. 60, no. 9, pp. 5322–5346, 2014.

[9] Y. Li *et al.*, "A low-complexity coded transmission scheme over finite-buffer relay links," *IEEE Trans. on Comms*, vol. 66, no. 7, pp. 2873–2887, 2018.

[10] D. E. Lucani *et al.*, "Fulcrum: Flexible network coding for heterogeneous devices," *IEEE Access*, vol. 6, pp. 77 890–77 910, 2018.

[11] N. Yazdani *et al.*, "Revolving codes: High performance and low overhead network coding," in *IEEE Wireless Africa Conf. (WAC)*. IEEE, 2019, pp. 1–5.

[12] D. P. Bertsekas *et al.*, *Introduction to probability*. Athena Scientific Belmont, MA, 2002, vol. 1.

[13] D. E. Lucani *et al.*, "Systematic network coding for time-division duplexing," in *IEEE Int. Symp. on Info. Theory*, 2010, pp. 2403–2407.