New Rateless Codes for Receiver With Limited Memory

Bohwan Jun, Pilwoong Yang, Jong-Seon No
Department of Electrical and Computer Engineering
Seoul National University
Seoul 08826, Korea
{netjic, yangpw}@ccl.snu.ac.kr, jsno@snu.ac.kr

Hosung Park
School of Electronics and Computer Engineering
Chonnam National University
Gwangju 61186, Korea
hpark1@jnu.ac.kr

***Abstract*—In this paper, we propose a new class of rateless codes which have improved intermediate performance using batched zigzag decoding over symbol erasure channels. When generating a batch, an encoder shifts selected input symbols before performing bit-wise XOR operation in order to make a batch zigzag decodable. Solving a batch by zigzag decoding algorithm gives improved intermediate performance. We show that the proposed codes outperform LT codes and zigzag decodable rateless codes with respect to coding overhead when symbol erasure rate is low and available buffer is limited.**

I. Introduction

Rateless codes are a class of forward error correction (FEC) codes which generate and transmit a limitless number of encoded symbols over erasure channels. Since an encoder generates a linear combination of a random number of symbols chosen from total k input symbols, a decoder can recover all input symbols successfully by receiving arbitrary $k\gamma_{succ}$ encoded symbols, where γ_{succ} is called a coding overhead and for capacity achieving case, it is equal to one. Therefore, rateless codes provide an efficient way of transmission in the point to multipoint channel. The most famous rateless codes are Luby transform (LT) codes [1], which are the first practical rateless codes.

Encoding procedure of LT codes can be described in two steps. First, an encoder samples a degree d from degree distribution and selects d input symbols uniformly and independently. Secondly, it performs bit-wise XOR operation. A decoder of LT codes stores all received encoded symbols in the storage called buffer and iterates the following two steps until all input symbols are successfully recovered. First, find encoded symbols with current degree one and move to the another storage named ripple. Secondly, select an arbitrary symbol in the ripple and perform bit-wise XOR operation of newly recovered input symbols to their neighboring encoded symbols. During the decoding steps, size of the ripple changes and it determines the performance of LT codes. To minimize coding overhead, various ripple evolutions are proposed. In [1], Luby proposed ideal Soliton distribution (ISD) and robust Soliton distribution (RSD) in order to keep ripple constant to one and larger than one, respectively. On the other hand, decreasing ripple evolution is proposed in [2] and it is shown that the performance significantly improves with the decreasing ripple size distribution (DRSD).

Recently, rateless codes based on zigzag decoding were proposed in [3] and [4]. When generating encoded symbols, they used shift operation before performing bit-wise XOR operation. Shift values of d input symbols are selected uniformly at random from $[0, s_{max}]$ for [3] and distinctly from $[0, d-1]$ for [4], respectively. Both codes showed improved performance respect to coding overhead.

However, LT codes and zigzag decodable rateless codes suffer from poor intermediate performance. Thus, these rateless codes have to store a lot of encoded symbols in the buffer. We consider a system such that a receiver is able to store only $k\beta$ encoded symbols in the buffer. Hence, when its buffer is full, a decoder has to discard encoded symbol randomly for new encoded symbol. Therefore, if $\beta < 1$, performance degrades severely. There were previous works to improved recovered input symbol ratio, z for $0 \leq \gamma \leq 1$, where z and γ are the ratios of recovered input symbols and received encoded symbol of k in [6]–[9], respectively. However, their drawbacks are high coding overhead, $O(k^2)$ additional complexity for encoding, and using feedbacks. To resolve this problem, we propose a new rateless code with improved intermediate performance without degradation in coding overhead and usage of feedbacks.

II. Preliminaries

In this section, we briefly overview zigzag decodable (ZD) codes. These codes use not only bit-wise XOR but also shift operation. As a result, the length of encoded symbol is slightly larger than the original length of input symbol, l. Input symbols are represented as polynomial form, i.e., $\mathbf{s}(z) = (s_1(z), \ldots, s_k(z))$, where the ith input symbol is

$$s_i(z) = s_{i,1} + s_{i,2}z + \cdots + s_{i,l}z^{l-1}$$

for $i = 1, 2, \ldots, k$.

We define a batch as a set of encoded symbols generated from a subset of the input symbols. A size and degree of batch $\mathbf{X}(z)$ are the number of the encoded symbols in $\mathbf{X}(z)$ and the number of input symbols involved in $\mathbf{X}(z)$, respectively. Then a batch of ZD codes with size T is represented as

$$\mathbf{X}(z) = \begin{bmatrix} x_1(z) \\ x_2(z) \\ \vdots \\ x_T(z) \end{bmatrix} = \begin{bmatrix} z^{v_{1,1}} & z^{v_{1,2}} & \cdots & z^{v_{1,d}} \\ z^{v_{2,1}} & z^{v_{2,2}} & \cdots & z^{v_{2,d}} \\ \vdots & \vdots & \ddots & \vdots \\ z^{v_{T,1}} & z^{v_{T,2}} & \cdots & z^{v_{T,d}} \end{bmatrix} \mathbf{B}(z),$$

where $\mathbf{B}(z)$ is a column vector which has d distinct input symbols as entries and $v_{i,j}$ is a nonnegative integer for $i = 1, \ldots, T$, $j = 1, \ldots, d$ and represents shift amount of jth

neighbor of ith encoded symbol. Each row of the generator matrix corresponds to d tuple coding vector $v_i = (v_{i,1}, \ldots, v_{i,d})$ for $i = 1, \ldots, T$. And a set of coding vectors is called batched coding vector, i.e., $\mathbf{v} = (v_1, \ldots, v_T)$. Let s_{max} be the maximum value in all batched coding vectors.

Definition 1: A batch is *zigzag decodable* if it can be successfully decoded by the following steps.

1) Find an encoded symbol in the batch such that the leftmost unrecovered bit is XORed by only one input bit. Using this bit, we can recover the leftmost unrecovered bit of input symbol immediately.
2) The recovered bit in Step 1 is XORed with all encoded symbols in the batch.
3) Continue Steps 1 and 2 iteratively until all bits of the corresponding input symbols are recovered.

III. Proposed Rateless Codes

In this section, we propose a new class of rateless codes. A key feature of the proposed codes is that they generate infinite number of batches with variable sizes. The size of batch T depends on degree, which is sampled from given degree distribution, $\Psi(x) = \sum_{d=1}^{k} \Psi_d x^d$, where Ψ_d is the probability that a batch is degree d. When $2 \le d \le s_{max}+1$, we set $T = d$, and otherwise, we set $T = 1$. Thus, when $T > 1$, T encoded symbols are generated with same neighbors simultaneously. Hence, it implies relation between $\Psi(x)$ and overall degree distribution of encoded symbol $\Omega(x)$. We have

$$\Psi_d = \begin{cases} \frac{\Omega_d}{d}\theta & \text{if } 2 \le d \le s_{max}+1, \\ \Omega_d \theta & \text{otherwise,} \end{cases} \tag{1}$$

where $\theta = \left(1 - \sum_{d=2}^{s_{max}+1} \frac{d-1}{d}\Omega_d\right)^{-1}$.

An encoder generates batches with carefully designed batched coding vector, which will be described in the following subsection. Hence, when symbol erasure rate is low, a decoder may receive all encoded symbols in the same batch without any erasures. Neighboring input symbols of zigzag decodable batches are recovered immediately, which leads to improvement on recovery ratio during $0 \le \gamma \le 1$.

Another feature is additional memory $\mathcal{I}'$ at the encoder, which stores whether ith input symbol has been selected or not for generating previous batches with $T > 1$. Since input symbols not in $\mathcal{I}'$ are picked preferentially, it is guaranteed that all input symbols are selected at least once. Now, we describe the rateless codes with design parameters $(k, l, \Psi(x), \mathbf{s}(z), s_{max})$ in the next subsections.

A. Encoder

Batched coding vectors for the proposed codes can be classified into Types 1 and 2 for $T = 1$ and $T > 1$, respectively. For Type 1, let $v' = (v_1, v_2, \cdots, v_d)$ be a coding vector such that v_j is selected from $[0, s_{max}]$ uniformly at random for $j = 1, \ldots, d$. Find the minimum value of v', i.e., $v_{min} = \min_{j \in \{1,\ldots,d\}}\{v_j\}$. Then subtract the value from all elements. Finally, a generated batched coding vector of size one is $\mathbf{v} = (v) = (v_1 - v_{min}, \cdots, v_d - v_{min})$. This generation is the same as that in [3].

And for Type 2, generating procedure of $\mathbf{v} = (v_1, \ldots, v_T)$ can be divided into three steps. First, sample an index g from $[1, T]$ uniformly at random. Secondly, using g, generate first $T-1$ coding vectors. For $i \in \{1, \ldots, T-1\}$, the gth element $v_{i,g}$ is fixed to zero and the other elements $v_{i,j}$ for $j \in [1, d] \setminus \{g\}$ are filled sequentially with $i-1$ right cyclic shifted version of $(1, \ldots, T-1)$. That is, $v_1 = (1, 2, \ldots, g-1, 0, g, g+1, \ldots, T-1)$, $v_2 = (T-1, 1, \ldots, g-2, 0, g-1, g, \ldots, T-2)$, …, $v_{T-1} = (2, 3, \ldots, g, 0, g+1, g+2, \ldots, 1)$. Lastly, for a remaining coding vector v_T, fix g'th element with zero and fill the elements with $\{1, \ldots, T-1\}$ sequentially, where $g' = (g+1) \bmod T$. It can be represented as $v_T = (1, 2, \ldots, g, 0, g+1, g+2, \ldots, T-1)$. This generation is the same as that in [4] and [5] except for removing randomness of v_T. Note that a batch generated by Type 2 batched coding vector is zigzag decodable.

Now overall encoding procedure of the proposed codes is given in Algorithm 1.

Algorithm 1: Encoding procedure
Input: $k, l, \Psi(x), \mathbf{s}(z), s_{max}$
Initialization: $\mathcal{I} \leftarrow \{1, \ldots, k\}$, $\mathcal{I}' \leftarrow \emptyset$, $i \leftarrow 1$
Step 1) Sample a degree d from degree distribution $\Psi(x)$.
Step 2)
- First phase: If $2 \le d \le s_{max}+1$, $T \leftarrow d$. Otherwise, $T \leftarrow 1$.
- Second phase: $T \leftarrow 1$.

Step 3)
- First phase: If $T > 1$, $r \leftarrow |\mathcal{I} \setminus \mathcal{I}'|$. If $r \ge d$, select distinct d indices of input symbols $\mathcal{I}_{sel} = \{i_1, \ldots, i_d\}$ from $\mathcal{I} \setminus \mathcal{I}'$ uniformly at random. Otherwise, select distinct r indices of input symbols $\mathcal{I}_{sel}^{(1)} = \{i_1, \ldots, i_r\} \leftarrow \mathcal{I} \setminus \mathcal{I}'$ and select the remaining $d-r$ indices $\mathcal{I}_{sel}^{(2)} = \{i_{r+1}, \ldots, i_d\}$ from $\mathcal{I}'$ uniformly at random. $\mathcal{I}_{sel} \leftarrow \mathcal{I}_{sel}^{(1)} \cup \mathcal{I}_{sel}^{(2)}$. Change to the second phase at the beginning of the next iteration. $\mathcal{I}' \leftarrow \mathcal{I}' \cup \mathcal{I}_{sel}$.
  Otherwise, select distinct d indices of input symbols from $\mathcal{I}$ uniformly at random. $\mathcal{I}_{sel} \leftarrow \{i_1, i_2, \ldots, i_d\}$.
- Second phase: Select distinct d indices of input symbols from $\mathcal{I}$ uniformly at random. $\mathcal{I}_{sel} \leftarrow \{i_1, i_2, \ldots, i_d\}$.

Step 4) If $T > 1$, generate Type 2 batched coding vector. Otherwise, generate Type 1 batched coding vector.
Step 5) For $j \leftarrow 1, \ldots, T$ do
  Generate encoded symbol x_j using $\mathcal{I}_{sel}$ and coding vector $v_j = (v_{j,1}, \ldots, v_{j,d})$.
  $x_j(z) \leftarrow z^{v_{j,1}} \cdot s_{i_1}(z) + \cdots + z^{v_{j,d}} \cdot s_{i_d}(z)$
  End for
  $\mathbf{X}_i(z) \leftarrow [x_1(z) \quad x_2(z) \quad \cdots \quad x_T(z)]^\intercal$
Step 6) $i \leftarrow i+1$. Go to Step 1).

B. Decoder

Similar to [3], all received encoded symbols are represented as bit-wise Tanner graph $\mathcal{G}_b = (V_b, C_b, E_b)$, where V_b and C_b denotes a set of input bit nodes and encoded bit nodes, respectively. And E_b denotes a set of edges (v_b, c_b) for $v_b \in V_b$ and $c_b \in C_b$. Since we use shift operation, length of an encoded symbol is larger than l, i.e., $l' \le l + s_{max}$. Thus, $|C_b|$ and $|E_b|$

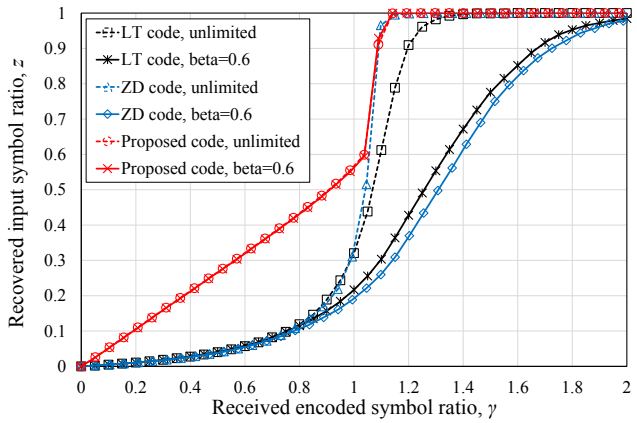

Fig. 1. Comparison of performance with respect to recovered input symbol ratio, z of γ between the proposed and the conventional codes for $k = 200$ at symbol erasure rate $\epsilon = 0.2$ when buffer size is unlimited and limited to $k\beta$, where $\beta = 0.6$.

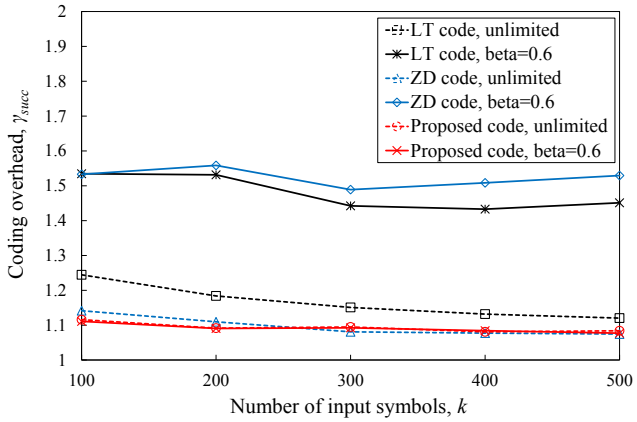

Fig. 2. Coding overhead for LT code, ZD code, and the proposed code for $k \in \{100, 200, 300, 400, 500\}$ when buffer size is unlimited and limited to $k\beta$, where $\beta = 0.6$. All symbol erasure rates are fixed to 0.2.

increase slightly. However, if s_{max} is fixed and the length of an input symbol l increases, then this loss becomes negligible.

IV. Simulation Results

In this section, we show comparisons between the proposed and the conventional codes, e.g., LT code and ZD code [3]. For simulation environment, we fixed $l = 50$ and $s_{max} = 4$ for both the proposed code and ZD code. Decreasing ripple size distribution is applied for LT code, ZD code, and the proposed code. The corresponding parameter (c_1, c_2) is $(1.3, 2.6)$ for $k = 100, 200$, and $(1.7, 2.5)$ for $k = 300, 400, 500$. We fix symbol erasure rate to $\epsilon = 0.2$ for all simulations.

In Fig. 1, the recovered input symbol ratio z of received encoded symbol ratio γ is shown. Although the proposed code and ZD code have similar performance for achieving $z = 1$, we can see that our proposed codes show improved intermediate performance for $0 \leq \gamma \leq 1$. It leads to occupying buffer much less than other codes. Thus, if we limit available buffer size to $k\beta$, where $\beta = 0.6$, performance of both LT code and ZD code degrades severely. However, it is shown that our proposed codes preserve its intermediate performance.

Fig. 2 shows that our proposed code outperforms LT code and ZD code for $k \in \{100, 200, 300, 400, 500\}$ when buffer size is limited, $\beta = 0.6$ when symbol erasure rate is 0.2. With improved intermediate performance, our proposed code rarely discards encoded symbols in the buffer. With batched zigzag decoding scheme, we have 4–12% improvement with respect to coding overhead compared to the conventional LT code using unlimited buffer.

V. Conclusion

In this paper, we proposed rateless coding scheme which has improved intermediate performance due to zigzag decodable batches. While the previous works for rateless codes considering intermediate performance have drawbacks such as degraded coding overhead due to high portion of degree 1 and 2, requiring extra computation for ordering, and usage of feedbacks, our codes require small additional bits for encoded symbols, which cause increasing computational and space complexity. However, if we keep s_{max} small compared to l, they become negligible. We show that the proposed codes outperform LT codes and ZD codes with respect to coding overhead and intermediate performance via simulations when symbol erasure rate is low and available buffer is limited small.

References

[1] M. Luby, "LT Codes," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, Nov. 2002, pp. 271–282.

[2] J. Sørensen, P. Popovski, and J. Østergaard, "Design and analysis of LT codes with decreasing ripple size," *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3191–3197, Nov. 2012.

[3] T. Nozaki, "Fountain codes based on zigzag decodable coding," in *Proc. Int. Symp. on Information Theory and its Applications (ISITA)*, 2014, pp. 274–278.

[4] J. Qureshi, C. H. Foh, and J. Cai, "Primer and recent developments on fountain codes," *Recent Advances in Communications and Networking Technology*, vol. 2, pp. 2–11, 2013.

[5] J. Qureshi, C. H. Foh, and J. Cai, "Optimal solution for the index coding problem using network coding over GF(2)," in *Proc. IEEE Conf. on Sensor, Mesh and Ad Hoc Comm. and Networks (SECON)*, 2012, pp. 209–217.

[6] A. Talari and N. Rahnavard, "On the intermediate symbol recovery rate of rateless codes," *IEEE Trans. Commun.*, vol. 60, no. 5, pp. 1237–1242, May 2012.

[7] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," in *Proc. 2006 Conf. Applications, Technologies, Architectures, Protocols Computer Commun.*, vol. 36, no. 4, pp. 255–266, 2006.

[8] A. Beimel, S. Dolev, and N. Singer, "RT oblivious erasure correcting," *IEEE/ACM Trans. on Netw.*, vol. 15, no. 6, pp. 1321–1332, Dec. 2007.

[9] Y. Cassuto, and A. Shokrollahi, "Online fountain codes with low overhead," *IEEE Trans. Inf. Theory*, vol. 61, no. 6, pp. 3137–3149, Jun. 2015.