On LT Codes with Decreasing Ripple Size

Jesper H. Sørensen*, Petar Popovski*, Jan Østergaard*,
*Aalborg University, Department of Electronic Systems, E-mail: {jhs, petarp, jo}@es.aau.dk

***Abstract*—In the orignal work on LT codes by Luby, the ripple size during decoding was designed to be approximately constant. In fact, the Ideal Soliton and the Robust Soliton distributions were designed to yield an approximately unit ripple size and a fixed ripple size greater than one, respectively. In this work, the redundancy within an LT code is characterized, and it is found that a decreasing ripple size during decoding is beneficial. Furthermore, a new design of LT codes with decreasig ripple size is proposed, and it is shown that this design significantly decreases the average amount of redundancy in comparison to existing designs.**

I. Introduction

Rateless codes are capacity achieving erasure correcting codes. Common for all rateless codes is the ability to generate a potentially infinite amount of encoded symbols from k input symbols. Decoding is possible when $(1+\alpha)k$ encoded symbols have been received, where α is close to zero. LT codes were developed by Luby and were the first practical capacity achieving rateless code [1]. A key part of LT codes is the degree distribution. Initially Luby presented the ISD, which achieves optimal behavior by keeping a parameter called the ripple size constantly equal to one throughout the decoding process. A ripple size above one introduces overhead, while decoding fails if the ripple size decreases to zero. For this reason, the ISD is optimal in theory, however, it lacks robustness against variance in the ripple size, which makes it inapplicable in practice. On the other hand, the performance of the RSD, which aims at ensuring a constant ripple size greater than one, is significantly better than that of the ISD, and it is the de facto standard for LT codes.

In this work, we investigate the trade-off between the robustness against variance in the ripple size and the required overhead. That is, the amount of encoded symbols, in excess of k, necessary in order to successfully decode, i.e. αk. We argue that the optimal robust degree distribution for LT codes does not seek a constant ripple size. Rather a degree distribution should ensure a ripple size which decreases during the decoding process. We support this claim by presenting a new degree distribution that outperforms both the RSD and the distribution recently presented in [2].

II. Background on LT Codes

Assume we wish to transmit a data file. This data is divided into k *input symbols*. From these input symbols a potentially infinite amount of encoded symbols, also called *output symbols*, are generated. Output symbols are XOR combinations of input symbols. The number of input symbols used in the XOR is referred to as the *degree* of the output symbol, and all input symbols contained in an output symbol are called *neighbors* of the output symbol. The output symbols of an encoder follow a certain degree distribution, $\pi(i)$, which is a key element in the design of good LT codes. The encoding process of an LT code can be broken down into three steps, which are repeated as many times as needed:

1) Randomly choose a degree i by sampling $\pi(i)$.
2) Choose uniformly at random i of the k possible input symbols.
3) Bitwise XOR the i input symbols to obtain the output symbol.

Decoding of an LT code is based on performing the reverse XOR operations. Initially all degree one output symbols are identified and moved to a storage referred to as the *ripple*. Symbols in the ripple are *processed* one by one, which means that they are removed as content from all buffered symbols through XOR operations. Once a symbol has been processed, it is removed from the ripple and considered decoded. The processing of symbols in the ripple will potentially reduce some of the buffered symbols to degree one, in which case they are moved to the ripple. This is called a symbol *release*. This makes it possible for the decoder to process symbols continuously in an iterative fashion. The decoder alternates between two steps:

1) Identify and add all degree one symbols to the ripple.
2) Process a symbol from the ripple and remove it afterwards.

Decoding is successful when all input symbols are recovered.

III. Results

To assess the evolution of the ripple size of LT codes, we provide the following lemmas, where the proofs can be found in [3].

Lemma 1. *(Release and Ripple Add Probability): The probability that a symbol of original degree i is released and added to the ripple, when L out of k input symbols remain unprocessed, given that the ripple size is R at the point of release, is:*

$$q(i,L,R) = \frac{i(i-1)(L-R+1)\prod_{j=0}^{i-3}(k-(L+1)-j)}{\prod_{j=0}^{i-1}(k-j)}$$
$$\text{for } i = 2,...,k-R+1, \quad L = R,...,k-i+1, \quad R = 1,...,k-1.$$

Lemma 2. *(Redundancy Probability): Assuming a constant ripple size R, the probability $r(i,R)$ that a symbol of original degree i is redundant is (under similar assumptions as in Lemma 1):*

$$r(i,R) = 1 - \sum_{L=R}^{k-i+1} q(i,L,R)$$
$$\text{for } i = 2,...,k-R+1, \quad R = 1,...,k-1.$$

Lemma 3. *(Ripple Evolution): The evolution of the ripple size, given expected behavior in the encoding and decoding processes, can be evaluated with the following set of equations:*

$$\Pi^k(i) = (1+\alpha)k\pi(i), \quad \text{for } i = 1,2,...,k,$$
$$\Pi^{L-1}(1) = \Pi^L(1) - 1 + \frac{2(L-\Pi^L(1))}{L(L-1)}\Pi^L(2),$$
$$\Pi^{L-1}(i) = \Pi^L(i) - \frac{i}{L}\Pi^L(i) + \frac{i+1}{L}\Pi^L(i+1), \quad \text{for } i = 2,3,...,L-1,$$
$$\Pi^{L-1}(L) = 0,$$

where $\Pi^L(i)$ is the amount of degree i symbols left in the decoding process, for an LT code with any degree distribution, $\pi(i)$, when L input symbols remain unprocessed.

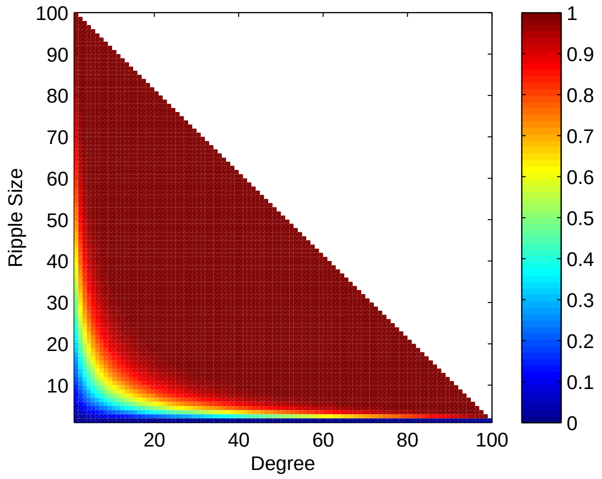

Fig. 1. The probability that an encoded symbol is redundant as a function of its degree and the ripple size at the point of release.

The redundancy $r(i,R)$ within an LT code with a fixed ripple size of R, as given by Lemma 2, is shown in Fig. 1, for the case of $k = 100$ input symbols. Note that $r(i,1) = 0$, $\forall i$, i.e., a released symbol has zero probability of already being in the ripple. This implies that a unit ripple size is optimal and the ISD should therefore be optimal. However, we must have a more robust ripple size, to avoid that the decoding fails, and even at R slightly greater than one, high degree symbols are very likely to be redundant.

A. Decreasing Ripple Size

At this point, we present a new degree distribution with decreasing ripple size. The details regarding the construction of this distribution can be found in [3]. The main argument for a decreasing ripple size is the fact that $r(i,R)$ is increasing much faster as a function of R at high i compared to at low i. Moreover, high degree symbols are released late in the decoding process. Thus, the price, in terms of redundancy, of having a robust ripple size increases significantly during the decoding process. For this reason, a decreasing ripple size will provide a better trade-off between robustness and overhead.

Definition 1. *(Decreasing Ripple Degree Distribution):*

$$\begin{aligned}
\theta(1) &= \frac{R}{n}\\
\theta(2) &= \frac{k(k-1)}{2n(k-R)}\\
\theta(i) &= \frac{i-2}{i}\theta(i-1) \quad \text{for} \quad i < \lfloor k/3 \rceil\\
\theta(i) &= \theta(i-1) \quad \text{for} \quad \lfloor k/3 \rceil \le i < \lfloor 2k/3 \rceil\\
\theta(i) &= \frac{k-i+1}{k-i}\theta(i-1) \quad \text{for} \quad \lfloor 2k/3 \rceil \le i \le k-R+1
\end{aligned}$$

where n is chosen such that $\sum_{i=1}^{k}\theta(i) = 1$.

Fig. 2 shows the ripple evolution of $\theta(i)$ at $k = 1000$ and $R = 20$. As desired, the ripple size decreases during the decoding process, and is quite low, yet still larger than one, near the end. It is worth noticing that this ripple evolution is achieved already at $\alpha = 0.05$. In the same figure, the ripple evolution of the RSD is plotted at the same α value. It is clear that the ripple of $\theta(i)$ experiences a significantly more robust evolution than that of the RSD.

In Fig. 3, the performance of $\theta(i)$ is simulated and compared to the RSD and the distribution proposed in [2], denoted $\beta(i)$.

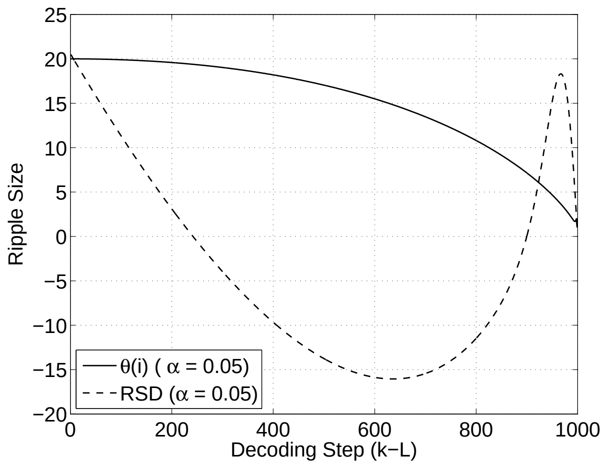

Fig. 2. The ripple evolution of $\theta(i)$ at $k = 1000$ and $R = 20$ compared to the RSD at same α value.

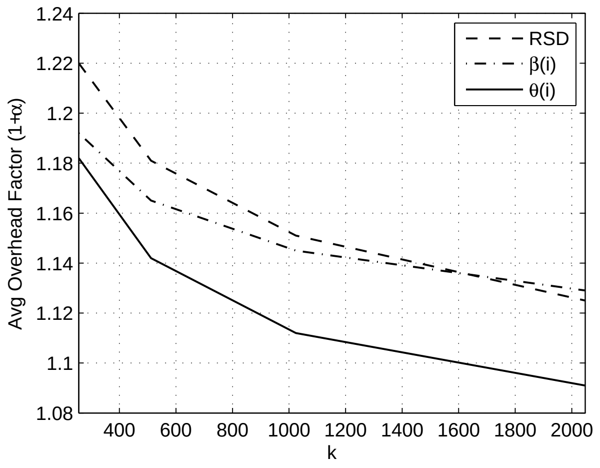

Fig. 3. Simulation results for the RSD, $\theta(i)$ and $\beta(i)$.

The performance metric is average overhead required for successful decoding of all k input symbols. The distributions are simulated at $k = 256, 512, 1024$, and 2048. For $\theta(i)$, we used Monte Carlo simulations to obtain a good value for R at each k, i.e, $R = 15, 17, 21$, and 25. The RSD is simulated with parameters $c = 0.1$ and $\delta = 1$, since these have been found to provide the smallest average overhead in [4]. The parameters for $\beta(i)$ are $\delta = 0.01$ and $R = 2 + \sqrt[4]{k}$, as suggested in [2]. The results are the average of 5000 simulations and reveal that $\theta(i)$ significantly outperforms the other distributions at all simulated k values. The gain compared to the RSD seems constant in absolute values for increasing k, while the gain compared to $\beta(i)$ is increasing. For example at $k = 2048$, $\theta(i)$ decreases α by roughly 0.04 compared to the other distributions, which translates into a decrease of roughly 30% in the average overhead.

References

[1] Michael Luby, "LT Codes," in *Proc. of the 43rd Annual IEEE Symposium on Foundations of Computer Science.*, pp. 271–280, Nov. 2002.

[2] Hongpeng Zhu, Gengxin Zhang and Guangxia Li, "A Novel Degree Distribution Algorithm of LT Codes," in *11th IEEE International Conference on Communication Technology.*, pp. 221–224, 2008.

[3] Jesper H. Sørensen, Petar Popovski and Jan Østergaard, "Design and analysis of LT codes with decreasing ripple size," *IEEE Transactions on Communications.*, 2010. Submitted.

[4] Frank Uyeda, Huaxia Xia and Andrew A. Chien, "Evaluation of a High Performance Erasure Code Implementation," *Technical Report, University of California, San Diego*, 2004.