Design and Analysis of 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 this paper we propose a new design of LT codes, which decreases the average amount of redundancy in comparison to existing designs. The design focuses on a parameter of the LT decoding process called the ripple size. This parameter was also a key element in the design proposed in the original work by Luby. Specifically, Luby argued that an LT code should provide a constant ripple size during decoding. In this work we argue that the ripple size should decrease during decoding, in order to reduce the redundancy. Initially we motivate this claim by analytical results related to the redundancy within an LT code. We then propose a new degree distribution, which provides the desired decreasing ripple size. The new degree distribution is evaluated and compared to the current state of the art through simulations. This reveals a noticeable increase in performance with respect to the average amount of redundancy.**

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. The generation of encoded symbols can be done on the fly during transmission, which means the rate of the code decreases as the transmission proceeds, as opposed to fixed rate codes, hence the name. Rateless codes are attractive due to their flexible nature. Regardless of the channel conditions, a rateless code will approach the channel capacity without the need for feedback. Moreover, practical implementations of rateless codes can be made with very low encoder and decoder complexity. The most successful examples are LT codes [1] and Raptor codes [2]. Originally rateless codes were intended for reliable file downloading in broadcast channels [3]. However, lately rateless codes have drawn significant interest in the area of mobile multimedia broadcast [4] [5].

LT codes were developed by Luby and were the first practical capacity achieving rateless code. A key part of Luby's design was the degree distribution, which is essential to a well-performing LT code. Initially Luby presented the Ideal Soliton distribution (ISD), which was shown to be optimal in terms of overhead, when all random processes follow expected behavior. By this we mean that when modeling the encoding and decoding processes for analysis, all random variables are assigned their expected value. Optimal behavior is achieved with the ISD, by keeping a parameter called the ripple size constantly equal to one throughout the decoding process. This parameter is described in details in section II. A ripple size above one introduces overhead, while decoding fails if the ripple size hits 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. In order to counter this problem, Luby developed the Robust Soliton distribution (RSD), which aims at ensuring a ripple size larger than one, yet still constant. The performance of the RSD is significantly better than that of the ISD, and it is the de facto standard for LT codes. In [6] the authors address the problem of finding a degree distribution, which provides a ripple of a given predefined constant size R. Initially, they show that such a degree distribution does not exist for high k, and then describe an approximate solution. In [7] the variance of the ripple size is derived with the purpose of designing a robust degree distribution. The analysis is based on an assumption, which makes it valid for only "most of the decoding process". The authors state that their next step is to work around this assumption, in order to solve the design problem.

In this work we investigate the trade-off between robustness against variance in the ripple size and 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 showing that a new degree distribution, proposed in this paper, outperforms both the RSD and the distribution developed in [6].

The remainder of this paper is organized as follows. Section II provides a brief overview of LT codes, explaining the encoding and decoding processes and relevant parameters. The analytical work of this paper is presented in section III, while simulation results are given in section IV. Conclusions are drawn in section V, followed by an Appendix explaining the details of the degree distribution design.

II. Background

A. LT Codes

In this section an overview of regular LT codes is given. Assume we wish to transmit a given amount of data, e.g. a file or slice of video from a stream. 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:

Encoder:

1) Randomly choose a degree i by sampling $\pi(i)$.
2) Choose uniformly at random i of the k possible input symbols.
3) Perform bitwise XOR of the i chosen input symbols. The resulting symbol is the output symbol.

This process can be iterated as many times as needed, which results in a rateless code.

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 iterative decoding process can be explained in two steps:

Decoder:

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

Decoding is successful when all input symbols have been recovered. If at any point before this, the ripple size equals zero, decoding has failed. This hints that a well performing LT code should ensure a high ripple size during the decoding process. However, when a symbol is released, there is a risk that it is already contained in the ripple, in which case the symbol is redundant. Hence, to minimize the risk of redundancy, the ripple size should be kept low. This trade-off was the main argument for the design goal in [1], that the ripple size should be kept constant at a reasonable level above one.

III. Analysis

It is clear from the description of LT codes in section II, that the ripple size is a very important parameter. The evolution of the ripple size is determined by the degree distribution. Thus, to obtain high decoding performance, the degree distribution should be chosen carefully, such that a desirable ripple evolution is achieved. The relation between the degree of an encoded symbol and the point of release was derived by Luby in Proposition 7 in [1]. By point of release, we mean the point in the decoding process, where the symbol is reduced to one of the input symbols and potentially added to the ripple. It is parameterized by L, the number of remaining unprocessed symbols. The relationship is given as a probability mass function, pmf, which expresses the release probability as a function of L and the original degree, i. Fig. 1 is a plot of the function for a number of fixed degrees, $i = 2, 4, ..., 20$, and $k = 100$. The figure clearly shows that as the degree increases, the symbol is more likely to be released late in the decoding process, which follows intuition. However, it also shows that already at quite low degrees, there is a significant probability that the symbol is not released until very late in the decoding process.

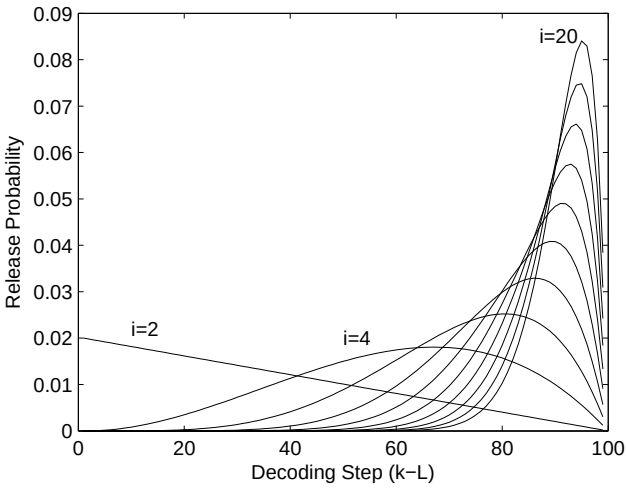

Fig. 1. The release probability as a function of the decoding step for fixed degrees (i).

The pmf in Lubys Proposition 7 expresses the release probability only and therefore does not take into account the probability of a redundant symbol, i.e. when the achieved input symbol is already in the ripple. This has been taken into account in Lemma 1.

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*

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

Proof: As in the proof of Proposition 7 in [1], this is the probability that $i-2$ of the neighbors are among the first $k-(L+1)$ processed symbols, one neighbor is the symbol processed at step $k-L$, and the last neighbor is among the $L-R+1$ unprocessed symbols which are not already in the ripple. Hence,

$$
\begin{aligned}
q(i,L,R) &= \frac{\binom{k-(L+1)}{i-2}\binom{1}{1}\binom{L-R+1}{1}}{\binom{k}{i}} \\
&= \frac{(L-R+1)\frac{(k-(L+1))!}{(i-2)!(k-(L+1)-(i-2))!}}{\frac{k!}{i!(k-i)!}} \\
&= \frac{(L-R+1)i!(k-i)!(k-(L+1))!}{(i-2)!k!(k-(L+1)-(i-2))!} \\
&= \frac{i(i-1)(L-R+1)\prod_{j=0}^{i-3}(k-(L+1)-j)}{\prod_{j=0}^{i-1}(k-j)}.
\end{aligned}
$$

■

Lemma 2. *(Redundancy Probability): Assuming a constant ripple size R, the probability that a symbol of original degree i is redundant is*

$$
\begin{aligned}
r(i,R) &= 1 - \sum_{L=R}^{k-i+1} q(i,L,R) \\
&\text{for } \; i = 2,...,k-R+1, \\
&R = 1,...,k-1.
\end{aligned}
$$

Proof: When summing $q(i,L,R)$ for all L, we get the probability that the symbol, at some point, will be released and be useful to us. The remaining probability mass accounts for the events where the symbol is released, but provides an input symbol which is already in the ripple. When this happens the symbol is redundant. ■

Lemma 2 is quite important, since it tells us much about when redundancy occur in an LT code. Fig. 2 shows a plot of $r(i,R)$ for $k = 100$. Note that $r(i,1) = 0$, $\forall\ i$, which was expected, since a ripple size of one means that a released symbol has zero probability of already being in the ripple. That is why the Ideal Soliton distribution is optimal for expected behavior. However, we must have a more robust ripple size, and even at R only slightly larger than one, high degree symbols are very likely to be redundant. In general, as i increases, $r(i,R)$ becomes a faster and faster increasing function of R. The following fact can be deduced from Figs. 1 and 2:

Fact. Early in the decoding process, when mostly low degree symbols are released, a ripple size larger than one induces a relatively low probability of redundancy. Conversely, late in the decoding process, when high degree symbols are released, a ripple size larger than one induces a relatively high probability of redundancy.

As mentioned in section II, Luby sets forth a design goal of having a constant ripple size at a reasonable level above one. This was motivated by the trade-off between overhead and robustness against variance in the ripple size. The design goal is formalized in a strict and approximate version in Definitions 1 and 2.

Definition 1. *(Constant Ripple Size): An LT code has a constant ripple size iff*

$$
\Pi^L(1) = \Pi^k(1) \;\text{ for }\; \Pi^k(1) \leq L \leq k,
$$

where $\Pi^L(1)$ is the expected number of unprocessed degree one symbols, i.e. the ripple, when the total number of unprocessed input symbols is L.

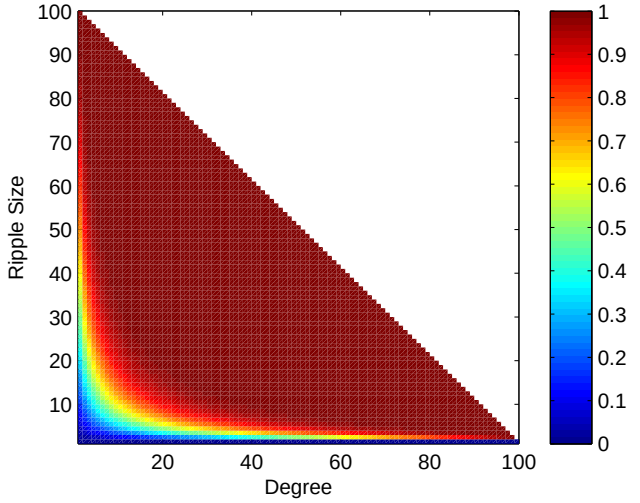

Fig. 2. The probability that an encoded symbol is redundant as a function of its degree and the ripple size at the point of release.

Definition 2. *(App. Constant Ripple Size): An LT code is said to have an (ϵ,δ)-approximately constant ripple size iff*

$$
|\Pi^L(1) - \Pi^k(1)| \leq \epsilon\Pi^k(1) \;\text{ for }\; \delta k \leq L \leq k,
$$

The parameter ϵ represents the chosen tolerance level, i.e. the deviation from the initial ripple size we will accept. The δ parameter indicates for how long we require the ripple size to stay within the tolerance levels. We cannot expect the ripple size to stay constant throughout the entire decoding process, since it must approach zero in the end where the number of unprocessed symbols approach zero.

In order to determine whether a certain degree distribution provides either a constant or approximately constant ripple size, we must be able to calculate how the ripple size evolves during the decoding process. For this purpose we present a set of equations, which have been derived using the same assumption as for the derivation of the ISD in [1]. This assumption is that the encoding and decoding processes follow expected behavior, i.e. that all realizations of random variables result in the expected value.

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:*

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

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.

Proof: The probability that a symbol of degree i has the processed symbol as neighbor is $\frac{i}{L}$. When a symbol of degree two is released, it is added to the ripple with probability $\frac{L-\Pi^L(1)}{L-1}$. ■

Example. If $\Pi^L(1) = 5$, $\Pi^L(2) = 10$ and $L = 90$, the expected number of released degree two symbols is $\frac{i}{L}\Pi^L(2) = \frac{2\cdot 10}{90}$. Out of these, an expected fraction of $\frac{L-\Pi^L(1)}{L-1} = \frac{90-5}{90-1}$ is added to the ripple. Moreover, processing a symbol will result in a decrease by one in the ripple size. Thus, $\Pi^{89}(1) = 5 - 1 + \frac{20}{90}\frac{85}{89} = 4.21$. Similarly, $\Pi^{89}(i)$ for $i = 2, 3, ..., L-1$ can be calculated using Lemma 3.

It was shown in [1] that the ISD satisfies the condition in Definition 1. However, it remains to be shown whether the RSD satisfies any of the conditions in Definitions 1 and 2. Using Lemma 3 the expected ripple evolution of the RSD has been evaluated at different α values. The RSD parameters are $c = 0.1$ and $\delta = 1$, since they have been found to provide the lowest average overhead in [8]. Fig. 3 shows the results. It is seen that the condition of an approximately constant ripple size is satisfied at roughly $\alpha = 0.16$, when $\epsilon = 0.15$ and $\delta = 0.2$ are chosen. The tolerance levels are indicated with dashed lines. In this case $k = 1000$ has been chosen, but the evaluation has been performed for a wide range of k values. This has revealed that the overhead required to satisfy the condition in Definition 2 is a decreasing function of k.

Although the RSD provides an approximately constant ripple size, it can not do so at arbitrary target ripple size, R. This lack of flexibility is inconvenient when designing a degree distribution which aims at minimizing the overhead. We will now present a degree distribution, $\gamma(i)$, with approximately constant arbitrary ripple size, R. The details on how we have derived this distribution are found in the Appendix.

Definition 3. *(App. Constant R-Ripple Degree Distribution):*

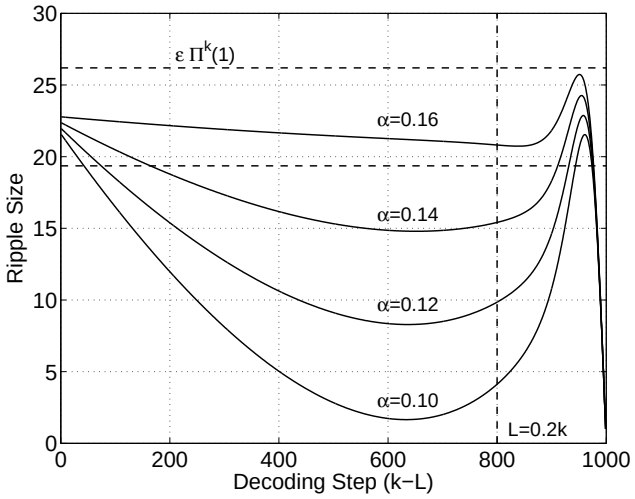

Fig. 3. The ripple evolution of the RSD at $k = 1000$ and $\alpha = \{0.10, 0.12, 0.14, 0.16\}$. The dashed lines indicate the tolerance levels.

$$
\begin{aligned}
\gamma(1) &= \frac{R}{n}, \\
\gamma(2) &= \frac{k(k-1)}{2n(k-R)}, \\
\gamma(i) &= \left(\frac{i-2}{i} + \frac{2(i-1)-4}{i(\Delta(k,R)-2)}\right)\gamma(i-1) \;\; \text{for} \;\; i \in \mathfrak{A}, \\
\gamma(i) &= \frac{k-i+1}{k-i+1+\frac{((i-1)-\Delta(k,R))(1-R)}{k-R-\Delta(k,R)}}\gamma(i-1) \;\; \text{for} \;\; i \in \mathfrak{B}, \\
\mathfrak{A} &= \{3, ..., \Delta(k,R)-1\}, \\
\mathfrak{B} &= \{\Delta(k,R), ..., k-R+1\}, \\
\Delta(k,R) &= (0.384R^{-1} + 19.1R^{-2} - 104R^{-3} + 232R^{-4} - 185R^{-5})(k-R-2) + 2,
\end{aligned}
$$

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

Lemma 3 is used to evaluate the ripple evolution of $\gamma(i)$. The result is shown in Fig. 4 for $k = 1000$ and R values of 5, 15 and 30. The ripple size stays within the tolerance levels, indicated with dashed lines, at the chosen R values, when $\epsilon = 0.15$ and $\delta = 0.2$. A wider range of R values has been evaluated. This revealed that the degree distribution is able to provide an approximately constant ripple size at R values between 2 and 40 for $k = 1000$. Higher k makes it possible to increase R even further.

The distribution in Definition 3 provides an approximately constant ripple size at arbitrary R. However, the necessary overhead factor is very high. At $k = 1000$ a ripple size of 5 can only be kept constant if α is in the area of 10^7, which makes the distribution totally inapplicable in practice. The reason for this extreme price in overhead is the fact that $\gamma(i)$ relies much more on high degree symbols than the RSD. We concluded from Figs. 1 and 2 that high degree symbols have a high probability of being redundant, when they are released at a point where the ripple size is larger than one. Thus, since $\gamma(i)$ aims at a robust ripple size, we should expect a high amount

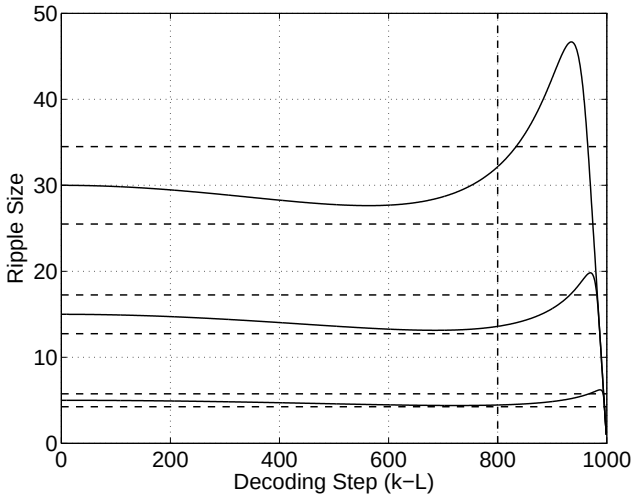

Fig. 4. The ripple evolution of $\gamma(i)$ at $k = 1000$ and $R = \{5, 15, 30\}$.

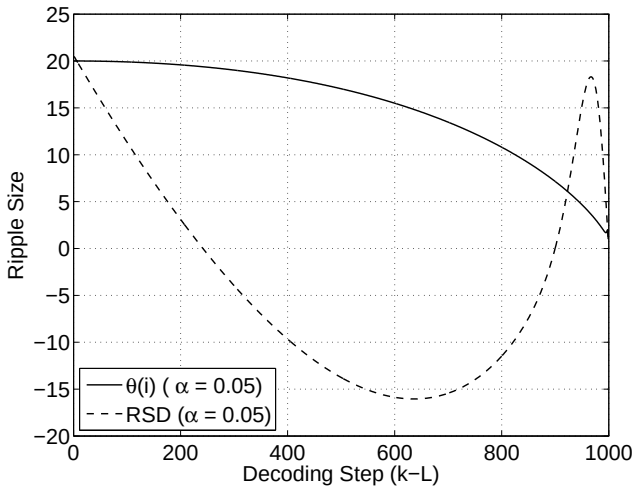

Fig. 5. The ripple evolution of $\theta(i)$ at $k = 1000$ and $R = 20$ compared to the RSD at same α value.

of redundancy.

A. Decreasing Ripple Size

Based on the observations so far, we argue that the degree distribution of an LT code should not aim at a constant ripple size. Instead the ripple size should decrease during decoding, in order to make sure that the ripple size is low near the end of the decoding process, where mainly high degree symbols are released. The main argument is the fact that $r(i, R)$ is increasing much faster as a function of R at high i compared to at low i. 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.

Using the same design approach as for $\gamma(i)$, a degree distribution with a decreasing ripple size has been found. A modification in the design has been made, which limits the amount of encoded symbols in the higher third of the degree spectrum. The details are described in the Appendix. We have already established that the flow into the ripple is based on symbols of increasing degrees, as decoding approaches the end. Thus, since we have decreased the amount of high degree symbols compared to the design of $\gamma(i)$, we should expect a ripple size which begins at the target size, R, and gradually decreases during decoding.

Definition 4. *(Decreasing Ripple Degree Distribution):*

$$\theta(1) = \frac{R}{n}$$
$$\theta(2) = \frac{k(k-1)}{2n(k-R)}$$
$$\theta(i) = \frac{i-2}{i}\theta(i-1) \text{ for } i < \lfloor k/3 \rceil$$
$$\theta(i) = \theta(i-1) \text{ for } \lfloor k/3 \rceil \leq i < \lfloor 2k/3 \rceil$$
$$\theta(i) = \frac{k-i+1}{k-i}\theta(i-1) \text{ for } \lfloor 2k/3 \rceil \leq i \leq k-R+1$$

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

Fig. 5 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. Hence, we expect $\theta(i)$ to outperform the RSD with respect to average overhead. Such a comparison is presented in the next section.

IV. Numerical Results

In this section, the performance of $\theta(i)$ is simulated and compared to the RSD and the distribution proposed in [6], denoted $\beta(i)$. The performance metric is average overhead required for successful decoding of all k input symbols. The distributions are simulated at k equal to 256, 512, 1024 and 2048. Well performing R values for $\theta(i)$ has been found for each k value through Monte Carlo simulations. The results were 15, 17, 21 and 25, respectively. The RSD is simulated with parameters $c = 0.1$ and $\delta = 1$, since these have been found to provide the smallest average overhead in [8]. The parameters for β are $\delta = 0.01$ and $R = 2 + \sqrt[4]{k}$, as suggested in [6]. The results are the average of 5000 simulations.

The results in Fig. 6 show 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. This confirms the claims in section III.

V. Conclusions

In this paper LT codes have been analyzed with the purpose of identifying the sources of redundancy. We arrived at the

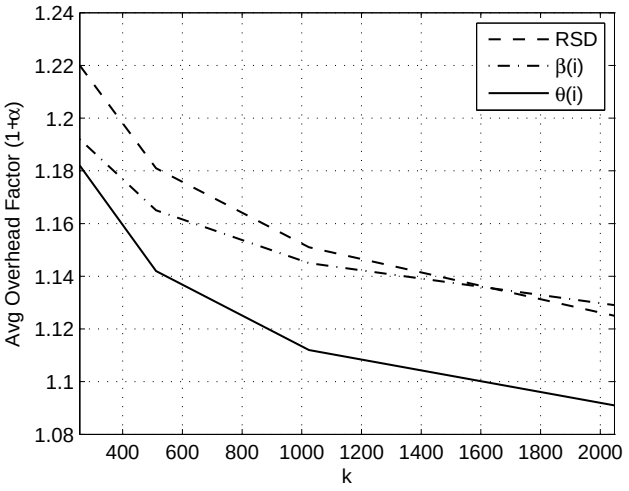

Fig. 6. Simulation results for the RSD, $\theta(i)$ and $\beta(i)$.

conclusion, that the probability of a symbol being redundant is a much faster increasing function of the ripple size when the symbol degree is high compared to when it is low. Since high degree symbols are utilized late in the decoding process, this means that the price of maintaining a high ripple size increases during the decoding process. Motivated by this result, we proposed a novel design in which the aim is to achieve a decreasing ripple size, as opposed to the existing strategy of keeping it constant. A degree distribution which provides a decreasing ripple size has been proposed and evaluated through simulations. The results show a noticeable performance increase compared to state of the art degree distributions. It can thus be concluded, that it is a misconception that a constant ripple size is a desirable feature of an LT code. Instead LT codes should provide a decreasing ripple size, since this gives a better trade-off between robustness and overhead.

APPENDIX
DEGREE DISTRIBUTION ANALYSIS

This appendix describes the analysis related to the derivations of the degree distributions presented in Definitions 3 and 4. With the distribution in Definition 3, we wish to ensure a constant arbitrary ripple size, R, during the decoding process. How we have achieved this is first described. As discussed in section III, the resulting degree distribution is inapplicable in practice. It was argued that instead we should aim at a decreasing ripple size. The design of such a degree distribution is also described.

A. Constant Ripple Degree Distribution

Our design of a constant ripple degree distribution is inspired by an interesting property held by the ISD, which we call the Russian Doll property. For a definition of the ISD see [1]. The Russian Doll property is connected to the expected degree evolution of the entire buffer content, i.e. unreleased symbols, during the decoding process. This buffer content is referred to as *the cloud*. We can show that the degrees of the symbols in the cloud will follow the ISD throughout the entire decoding process, if decoding follows expected behavior. This property is expressed in the following theorem.

Theorem 1. *(The Russian Doll Property): k encoded symbols, whose degrees follow the ISD with parameter k, will reduce to $k-1$ encoded symbols, whose degrees follow the ISD with parameter $k-1$, when a single symbol from the ripple is processed.*

Proof: By $E[I_k^L(i)]$, we denote the expected number of degree i symbols left in the cloud, when L out of k symbols remain unprocessed and the applied degree distribution is the ISD. Hence, the goal is to show that $E[I_k^{k-1}(i)] = E[I_{k-1}^{k-1}(i)]$, $\forall i$. We know from [1] that for every processed symbol, the expected number of released symbols, i.e. degree two symbols which contain the processed symbol, is one. This holds only when k symbols have been received. The expected number of degree two symbols which have the processed symbol as neighbor is denoted $E[I_k^L(2) \rightarrow I_k^L(1)]$. We need to derive $E[I_k^L(i) \rightarrow I_k^L(i-1)]$ for $i = 2, 3, ..., k$. Initially we note that the expected number of received symbols with degree i, when k symbols have been received and decoding has not started, is $E[I_k^k(i)] = \frac{k}{i(i-1)}$. The total number of non unique neighbors of these is $\frac{k}{i-1}$, since each of them have i neighbors. The first processed symbol is expected to constitute a fraction of $\frac{1}{k}$ of these, which means that

$$E[I_k^k(i) \rightarrow I_k^k(i-1)] = \frac{1}{i-1}, \qquad i = 2, 3, ..., k. \tag{1}$$

With this result it is possible to evaluate the expected number of degree i symbols in the cloud, after a single symbol has been processed, $E[I_k^{k-1}(i)]$. Note that the net decrease is $E[I_k^k(i) \rightarrow I_k^k(i-1)] - E[I_k^k(i+1) \rightarrow I_k^k(i)]$, with a special case for $i = k$, where the net decrease is $E[I_k^k(k) \rightarrow I_k^k(k-1)]$, since $E[I_k^k(k+1) \rightarrow I_k^k(k)] = 0$ by definition, hence

$$\begin{aligned} E[I_k^{k-1}(i)] =& E[I_k^k(i)] - E[I_k^k(i) \rightarrow I_k^k(i-1)] \\ &+ E[I_k^k(i+1) \rightarrow I_k^k(i)] \\ =& \frac{k}{i(i-1)} - \frac{1}{i-1} + \frac{1}{i} \\ =& \frac{k}{i(i-1)} - \frac{i}{i(i-1)} + \frac{i-1}{i(i-1)} \\ =& \frac{k-1}{i(i-1)}, \qquad i = 2, 3, ..., k-1, \end{aligned} \tag{2}$$

$$E[I_k^{k-1}(k)] = \frac{k}{k(k-1)} - \frac{1}{k-1} = 0. \tag{3}$$

The expected values expressed in (2) and (3) correspond to the expected initial values for an ISD with parameter $k-1$, $E[I_{k-1}^{k-1}(i)]$, when $k-1$ symbols have been received. ■

The Russian Doll property essentially means that when processing a symbol, the support of the cloud distribution is decreased by one, however the shape of the distribution stays

the same. Since the resulting distribution is also an ISD, it still holds the Russian Doll property. This interpretation is the inspiration for the name. The property is desirable, since it implies a constant ripple size. We therefore use the property as the design goal of our degree distribution, $\gamma_k(i)$, with constant arbitrary ripple size, R.

The ISD is able to achieve the Russian Doll property, and thereby constant ripple, at the reception of only k symbols, i.e. no overhead. Unfortunately, that is not possible for arbitrary R, hence we must consider the case of n collected symbols instead. Initially, we state that in order to have an expected initial ripple size of R, we have that $\gamma_k(1) = \frac{R}{n}$. In order to keep the ripple size constant, we need to ensure that every time we process a symbol, the expected number of released symbols, not already in the ripple, is equal to one. A symbol is released and put in the ripple when a degree two symbol has the processed symbol and one other symbol, which is not already in the ripple, as neighbors. Hence, $\gamma_k(2)$ can be derived as follows:

$$\begin{aligned} E[\Gamma_k^k(2) \to \Gamma_k^k(1)] &= 1 \\ \gamma_k(2) \cdot n \cdot \frac{\binom{1}{1}\binom{k-R}{1}}{\binom{k}{2}} &= 1 \\ \gamma_k(2) &= \frac{k(k-1)}{2n(k-R)}. \end{aligned} \tag{4}$$

When determining $\gamma_k(i)$ for $i > 2$, the property in Theorem 1 is used as the design criteria. Hence, we need to ensure that $E[\Gamma_k^{k-1}(i)] = E[\Gamma_{k-1}^{k-1}(i)]$, which means the following must hold:

$$\begin{aligned} &E[\Gamma_k^k(i)] - E[\Gamma_k^k(i) \to \Gamma_k^k(i-1)] + E[\Gamma_k^k(i+1) \to \Gamma_k^k(i)] = E[\Gamma_{k-1}^{k-1}(i)] \\ &E[\Gamma_k^k(i-1)] - E[\Gamma_k^k(i-1) \to \Gamma_k^k(i-2)] + E[\Gamma_k^k(i) \to \Gamma_k^k(i-1)] = E[\Gamma_{k-1}^{k-1}(i-1)] \\ &\gamma_k(i-1)n - \gamma_k(i-1)n\frac{i-1}{k} + \gamma_k(i)n\frac{i}{k} = \gamma_{k-1}(i-1)n \\ &\gamma_k(i) = \frac{k}{i}\gamma_{k-1}(i-1) - \frac{k-i+1}{i}\gamma_k(i-1). \end{aligned} \tag{5}$$

Through recursion, (5) can be rewritten to express $\gamma_k(i)$ as a linear combination of $\gamma_{k-j}(2)$, $j = 0, 1, ..., d-2$, all of which can be found using (4). In this way the remainder of the distribution can be found. However, at k above approximately 40, this solution gives an invalid distribution with negative entries. Hence, it can be concluded that the property in Theorem 1 can not be achieved in LT codes with more than 40 input symbols and $R > 1$. This conclusion is similar to the conclusion in [6], where an ill-conditioned matrix makes it impossible to find valid distributions, which ensure a constant expected ripple size for high k. As a result, we need to make an approximation in (5).

The aim in this approximation is to eliminate the term including $\gamma_{k-1}(i-1)$, since this is the source of the increasing number of variables in the linear combination found through recursion. We do this by observing the ratios $\frac{\gamma_{k-1}(i-1)}{\gamma_k(i-1)}$ and

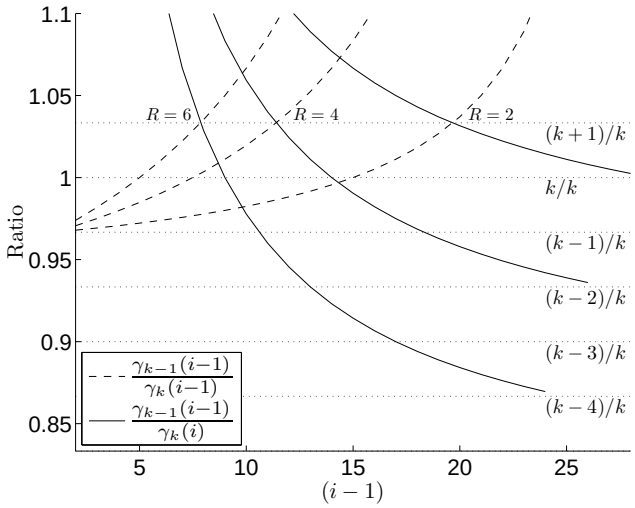

Fig. 7. The observed ratios at $k = 30$ and $R = \{2, 4, 6\}$.

$\frac{\gamma_{k-1}(i-1)}{\gamma_k(i)}$ for increasing k, in the range where the exact solution can be found, $k < 40$. It turns out that a pattern appears, which can be utilized to express $\gamma_{k-1}(i-1)$ as a function of either $\gamma_k(i-1)$ or $\gamma_k(i)$. The observed ratios have been plotted in Fig. 7 along with a number of reference lines. Initially it is observed that the two ratios always intersect at $\frac{k+1}{k}$ regardless of R. Plots at different k show that this holds for any k as well. These plots have been left out due to space considerations. The location of the intersection with respect to i is a function of k and R and is denoted $\Delta(k, R)$. Through polynomial regression an approximation of this function has been found as $\Delta(k, R) = (0.384R^{-1} + 19.1R^{-2} - 104R^{-3} + 232R^{-4} - 185R^{-5})(k-R-2)+2$. Other important observations are that $\frac{\gamma_{k-1}(i-1)}{\gamma_k(i-1)}$ is approximately $\frac{k-1}{k}$ at $i-1 = 2$ and that $\frac{\gamma_{k-1}(i-1)}{\gamma_k(i)}$ is approximately $\frac{k-R+2}{k}$ at $i-1 = k-R$. We now have enough fix points to approximate $\gamma_{k-1}(i-1)$. For $i < \Delta(k, R)$ we use a linear approximation of $\frac{\gamma_{k-1}(i-1)}{\gamma_k(i-1)}$ based on the fix points $\left(2, \frac{k-1}{k}\right)$ and $\left(\Delta(k, R), \frac{k+1}{k}\right)$. Similarly, for $i \geq \Delta(k, R)$ we use a linear approximation of $\frac{\gamma_{k-1}(i-1)}{\gamma_k(i)}$ based on the fix points $\left(\Delta(k, R), \frac{k+1}{k}\right)$ and $\left(k-R, \frac{k-R+2}{k}\right)$. See Fig. 8. Hence,

$$\begin{aligned} \gamma_{k-1}(i-1) \approx \begin{cases} \left(\frac{k-1}{k} + \frac{2(i-1)-4}{k(\Delta(k,R)-2)}\right)\gamma_k(i-1), & \text{for } i \in \mathfrak{A}, \\ \left(\frac{k+1}{k} + \frac{((i-1)-\Delta(k,R))(1-R)}{k(k-R-\Delta(k,R))}\right)\gamma_k(i), & \text{for } i \in \mathfrak{B}, \end{cases} \\ \mathfrak{A} = \{3, ..., \Delta(k, R) - 1\}, \\ \mathfrak{B} = \{\Delta(k, R), ..., k - R + 1\}. \end{aligned} \tag{6}$$

By combining (5) and (6) we arrive at the remaining part of the distribution presented in Definition 3.

B. Decreasing Ripple Degree Distribution

In section III it is concluded that the distribution in Definition 3 is inapplicable in practice due to high overhead. For this reason a degree distribution with a decreasing ripple size

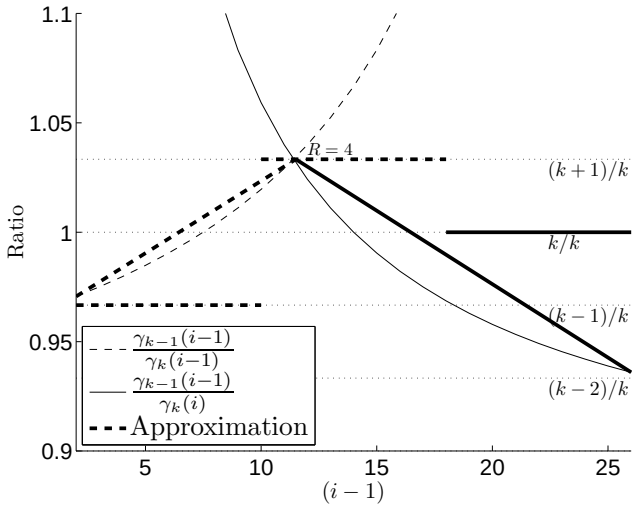

Fig. 8. The observed ratios at $k = 30$ and $R = 4$ compared to the applied approximations.

was presented in Definition 4, which reduces the amount of high degree symbols compared to the distribution in Definition 3. This is achieved by applying a different substitution for $\gamma_{k-1}(i-1)$ than the approximation just explained. We divide the degree spectrum into three sections and use the following substitutions:

$$\gamma_{k-1(i-1)} \approx \begin{cases} \frac{k-1}{k}\gamma_{k(i-1),} & \text{for } i < \lfloor k/3 \rceil, \\ \frac{k+1}{k}\gamma_{k(i-1),} & \text{for } \lfloor k/3 \rceil \leq i < \lfloor 2k/3 \rceil, \\ \gamma_{k(i),} & \text{for } \lfloor 2k/3 \rceil \leq i \leq k-R+1. \end{cases} \tag{7}$$

For the lower two sections, we have used the approximations $\frac{k-1}{k}\gamma_k(i-1)$ and $\frac{k+1}{k}\gamma_k(i-1)$ respectively. For the higher section, containing high degree symbols, we substitute $\gamma_{k-1}(i-1)$ with $\gamma_k(i)$, i.e. ratio one. This ratio is significantly higher than the linearly decreasing ratio applied in (6), which can be seen by comparing in Fig. 8. What this does to our degree distribution can be seen from the following, where we denote the ratio applied in the substitution a.

$$\begin{aligned} \gamma_k(i) &= \frac{k}{i}\gamma_{k-1}(i-1) - \frac{k-i+1}{i}\gamma_k(i-1), \\ &\approx \frac{k}{i}a\gamma_k(i) - \frac{k-i+1}{i}\gamma_k(i-1), \\ &= \frac{k-i+1}{ak-i}\gamma_k(i-1). \end{aligned} \tag{8}$$

Hence, by choosing $a = 1$ instead of the ratio applied in (6), which approaches $\frac{k-R+2}{k}$ for i approaching $k-R+1$, we decrease the amount of high degree symbols, for $R > 2$. Since high degree symbols are released late in the decoding process, this provides the desired decreasing ripple size. Combining (5) and (7) we arrive at the distribution presented in Definition 3.

REFERENCES

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

[2] Amin Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory.*, pp. 2551–2567, 2006.

[3] John W. Byers, Michael Luby, and Michael Mitzenmacher, "A Digital Fountain Approach to Asynchronous Reliable Multicast," *IEEE Journal on Selected Areas in Communications.*, pp. 1528–1540, 2002.

[4] Dino Sejdinović, Dejan Vukobratović, Angela Doufexi, Vojin Šenk, and Robert J. Piechocki, "Expanding Window Fountain Codes for Unequal Error Protection," *IEEE Transactions on Communications.*, pp. 2510–2516, 2009.

[5] Jean-Paul Wagner, Jacob Chakareski and Pascal Frossard, "Streaming of Scalable Video from Multiple Servers using Rateless Codes," in *IEEE International Conference on Multimedia and Expo.*, pp. 1501–1504, 2006.

[6] 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.

[7] Ghid Maatouk and Amin Shokrollahi, "Analysis of the Second Moment of the LT Decoder," in *IEEE International Symposium on Information Theory.*, pp. 2326–2330, 2009.

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