LT Codes

Michael Luby
Digital Fountain, Inc.
luby@digitalfountain.com

Abstract

We introduce LT codes, the first rateless erasure codes that are very efficient as the data length grows.

Keywords: erasure codes, rateless codes, universal codes, reliable transport, balls and bins, randomized algorithms

1. Introduction

LT codes are the first realization of a class of erasure codes that we call universal erasure codes. The symbol length for the codes can be arbitrary, from one-bit binary symbols to general ℓ-bit symbols. We analyze the run time of the encoder and decoder in terms of symbol operations, where a symbol operation is either an exclusive-or of one symbol into another or a copy of one symbol to another. If the original data consists of k input symbols then each encoding symbol can be generated, independently of all other encoding symbols, on average by $O(\ln(k/\delta))$ symbol operations, and the k original input symbols can be recovered from any $k+O(\sqrt{k}\ln^2(k/\delta))$ of the encoding symbols with probability $1-\delta$ by on average $O(k \cdot \ln(k/\delta))$ symbol operations.

LT codes are rateless, i.e., the number of encoding symbols that can be generated from the data is potentially limitless. Furthermore, encoding symbols can be generated on the fly, as few or as many as needed. Also, the decoder can recover an exact copy of the data from any set of the generated encoding symbols that in aggregate are only slightly longer in length than the data. Thus, no matter what the loss model is on the erasure channel, encoding symbols can be generated as needed and sent over the erasure channel until a sufficient number have arrived at the decoder in order to recover the data. Since the decoder can recover the data from nearly the minimal number of encoding symbols possible, this implies that LT codes are near optimal with respect to any erasure channel. Furthermore, the encoding and decoding times are asymptotically very efficient as a function of the data length. Thus, LT codes are universal in the sense that they are simultaneously near optimal for every erasure channel and they are very efficient as the data length grows.

The analysis of LT codes is quite different than the analysis of Tornado codes [16] [17] [15]. In particular, the Tornado codes analysis is only applicable to graphs with constant maximum degree, and LT codes use graphs of logarithmic density, and thus the Tornado codes analysis does not apply. Furthermore, the Tornado codes analysis relies on techniques that lead to a reception overhead that is inherently at least a constant fraction of the data length, whereas the analysis of LT codes shows that the reception overhead is an asymptotically vanishing fraction of the data length, albeit at the cost of slightly higher asymptotic encoding and decoding times.

The key to the design and analysis of the LT codes is the introduction and analysis of the LT process. This process and its analysis is a novel generalization of the classical process and analysis of throwing balls randomly into bins and it may be of independent interest. We provide a full analysis of the behavior of the LT process using first principles of probability theory, which precisely captures the behavior of the data recovery process.

The "digital fountain approach" concept introduced in [7] [6] is similar to that of a universal erasure code, and LT codes are the first full realization of this concept. This paper describes some of the theoretical underpinnings of some portions of the work described in [12] [13].

1.1 Some encoding details

The process of generating an encoding symbol is conceptually very easy to describe:

- Randomly choose the degree d of the encoding symbol from a degree distribution. The design and analysis of a good degree distribution is a primary focus of the remainder of this paper.
- Choose uniformly at random d distinct input symbols as neighbors of the encoding symbol.
- The value of the encoding symbol is the exclusive-or of the d neighbors.

When using the encoding symbols to recover the original input symbols of the data, the decoder needs to know the degree and set of neighbors of each encoding symbol. There are many different ways of communicating this information to the decoder, depending on the application. For example, the degree and a list of neighbor indices may be given explicitly to the decoder for each encoding symbol. As another example, the degree and neighboring indices of each encoding symbol may be computed by the decoder implicitly based for example on the timing of the reception of the encoding symbol or the position of the encoding symbol relative to the positions of other encoding symbols. As another example, a key may be associated with each encoding symbol and then both the encoder and decoder apply the same function to the key to produce the degree and set of neighbors of the encoding symbol. In this case, the encoder may randomly choose each key it uses to generate an encoding symbol and keys may be passed to the decoder along with the encoding symbols. Each key may instead be produced for example by a deterministic process, e.g., each key may be one larger than the previous key. The encoder and decoder may have access to the same set of random bits, and the key may be used as the seed to a pseudo-random generator that uses these random bits to produce the degree and the neighbors of the encoding symbol. There are a variety of ways to associate a degree and a set of neighbors with an encoding symbol depending on the application, and these implementation details are beyond the scope of this paper.

1.2 Comparison to traditional erasure codes

Traditional erasure codes are typically block codes with a fixed rate, i.e., k input symbols are used to generate $n - k$ redundant symbols for a total of n encoding symbols, and the rate of the code is k/n. For example, research in networking has suggested using implementations of both Reed-Solomon and Tornado codes for reliable data distribution applications, and in these cases both k and n either are limited to fairly small values due to practical considerations or are fixed before the encoding process begins, see e.g. [21] [26] [2] [7] [6]. Ideally, as is the case with a Reed-Solomon code [25], any k of the n encoding symbols is sufficient to recover the original k input symbols (see for example [18]). Sometimes, as is the case with a Tornado code [16] [17], slightly more than k of the n encoding symbols are needed to recover the original k input symbols.

Reed-Solomon codes in practice are only efficient for relative small settings of k and n. The reason is that for a standard implementation of Reed-Solomon codes [26] [2], $k(n - k)A/2$ symbol operations are needed to produce the n encoding symbols, where A is the size of the finite field used. Decoding times are similar. Although there are theoretical algorithms for implementing Reed-Solomon codes that are asymptotically faster for both encoding and decoding, i.e., k times polylogarithmic in n, the quadratic time implementations are faster in practice for values of k and n of interest.

Tornado codes are block erasure codes that have linear in n encoding and decoding times [16] [17] [7] [6]. LT codes are somewhat similar to Tornado codes. For example, they use a similar rule to recover the data, and the degree distributions used for Tornado codes is superficially similar to the degree distributions used for LT codes. However, the actual degree distribution used for Tornado codes turns out to be inapplicable to LT codes. For Tornado codes the degree distribution on input symbols is similar to the Soliton distribution described later and the degree distribution on the first layer of redundant symbols is close to the Poisson distribution. The Soliton distribution on input symbols is inapplicable for LT codes, as this distribution cannot be the resulting degree distribution on input symbols when encoding symbols are generated independently, no matter what the distribution on neighbors of encoding symbols is chosen to be. One could imagine flipping the distributions on Tornado codes so that, like the LT codes, the Poisson distribution is on the input symbols and the Soliton distribution is on the redundant symbols. However, the distribution induced on the redundant symbols by the missing input symbols in the Tornado code graph is far from the Soliton distribution, and applying the recovery rule to the induced distributions clearly won't lead to a reasonable reception overhead.

Beyond the differences between LT codes and Tornado codes mentioned previously, LT codes have significant application advantages over Tornado codes. Let $c = n/k$ be the constant stretch factor of the Tornado code design. Once k and n have been fixed the Tornado encoder produces n encoding symbols, and cannot produce further encoding symbols on the fly if the demand arises. In contrast, the encoder can generate as few or as many encoding symbols as needed on demand.

Tornado codes use a cascading sequence of bipartite graphs between several layers of symbols, where the input symbols are at the first layer and redundant symbols are at each subsequent layer. In practice, this requires either prior construction of the exact same graph structure at both the encoder and decoder, or prior construction of the graph structure at the encoder that is then communicated to the decoder. This preprocessing is quite cumbersome in either case, and in particular the graph structure size is proportional to $n = ck$. Although the amortized overhead is lessened if the same length data is to be encoded and decoded repeatedly, a different graph structure is required for each data length and thus the preprocessing is quite a large drawback if each data is of a different length. In contrast, the degree distributions used by LT codes are easily computed based on the data length, and this is the only preprocessing needed prior to invoking either the encoder or the decoder.

For the Tornado codes the decoding depends on having a sufficient number of distinct encoding symbols from the n encoding symbols. In some transport applications, the total number of encoding symbols transmitted from a sender is several times the number of encoding symbols than can be reasonably generated by the Tornado encoder. For these applications, in some cases a good policy is to randomly select an encoding symbol each time a symbol is to be sent. In this case, it is not hard to see that the average number of encoding symbols that a receiver needs to receive to obtain k distinct encoding symbols among the $n = ck$ encoding symbols is $ck \ln(c/(c-1))$. (Recall that at least k distinct encoding symbols are required to recover the data). Thus, for example, if $c = 2$ then the client needs to receive $1.386k$ encoding symbols on average in order to receive k distinct encoding symbols. To make this more acceptable, e.g., reception of $1.05k$ encoding symbols, requires $c > 10$. These values for c makes the Tornado codes unattractive for these types of applications because the encoder and decoder memory usage and the decoding time is proportional to c times the data length. In contrast, for LT codes the encoder and decoder memory usage is proportional to the data length and the decoding time depends only on the data length, independent of how many encoding symbols are generated and sent by the encoder.

Both Reed-Solomon and Tornado codes are systematic codes, i.e. all the input symbols that constitute the data are directly included among the encoding symbols, whereas LT codes are not systematic.

1.3 Applications

Normally when data is to be transmitted on an IP based network, it is partitioned into equal length pieces and placed into packets. Any transmission protocol that uses this type of protocol must be able to ensure the reception of all packets by receivers. This is why for example TCP receiver acknowledges each received packet and a TCP sender retransmits unacknowledged packets. Because of these properties, TCP/IP does not scale well to transmission over networks with high latencies, to transmission over networks with high loss rates, or to highly concurrent transmission of the same content to multiple clients.

Erasure codes have been proposed to provide reliability for a variety of data delivery applications. For one-to-one delivery, there are advantages in designing the reliability mechanism independently from the flow and congestion control mechanism, as outlined in [1]. The advantage of using erasure codes in one-to-one data delivery is that it makes it easier to design the flow and congestion control mechanisms independently of reliability.

For the one-to-many data delivery problem, where scalability of the sender and the network is of paramount importance, feedback to the sender needs to be limited and sending packets that are redundant for some receivers is wasteful, reliability may be able to be provided using erasure codes as described in [21], [28], [26], [27], [22], [23], [9] [7] [6]. The attractiveness of this approach is that if the erasure codes are powerful enough then a single sender can potentially be used to reliably deliver data efficiently to a large number of concurrent receivers without feedback.

The one-to-many problem is even more difficult when different receivers have different bandwidth connections to the sender and it is desired that each receiver obtain the data at the fastest possible speed independent of other receivers. Receiver-driven congestion control protocols have been designed independently of a reliability protocol that scalably delivers a different fraction of the packets generated to each receiver depending on current network conditions and the receivers bandwidth connection to the sender [19] [30] [4] [14]. Erasure codes can be used in conjunction with these congestion control protocols to help provide a complete reliable delivery transport.

For many-to-one delivery it is useful for receivers to be able to concurrently receive packets for the same data from multiple senders potentially at different locations. In this case receiving redundant packets is a concern, and erasure codes can be used to minimize delivery of redundant packets. This application is considered in [5], and a partial solution is given based on Tornado codes [16] [17].

LT codes offers compelling advantages for all these different types of data delivery applications that no previous erasure codes provides. Using LT codes, close to the minimal number of encoding symbols can be generated and sent in packets to receivers, and the number of packets each receiver needs before it is able to recover the original data is asymptotically close to the minimal possible. When traditional erasure codes are used in these solutions, the number of packets to generate is fixed based on a best guess of network conditions prior to producing any packets, and inevitably this guess is either too high or too low.

There are a variety of other applications of LT codes, including robust distributed storage, delivery of streaming content, delivery of content to mobile clients in wireless networks, peer-to-peer applications [3], delivery of content along multiple paths to ensure resiliency to network disruptions [24], and a number of other applications too numerous to detail in this paper.

2. LT Codes Design

The length ℓ of the encoding symbols can be chosen as desired. The overall encoding and decoding is more efficient in practice for larger values of ℓ because of overheads with bookkeeping operations, but the value of ℓ has no bearing on the theory. In transport applications ℓ is sometimes chosen to be close to the length of the packet payload.

The encoder works as follows. The data of length N is partitioned into $k = N/\ell$ input symbols, i.e., each input symbol is of length ℓ. Each encoding symbol is generated as described in Subsection 1.1.

Given an ensemble of encoding symbols and some representation of their associated degrees and sets of neighbors, the decoder repeatedly recovers input symbols using the following rule as long as it applies.

Definition 1 *(decoder recovery rule): If there is at least one encoding symbol that has exactly one neighbor then the neighbor can be recovered immediately since it is a copy of the encoding symbol. The value of the recovered input symbol is exclusive-ored into any remaining encoding symbols that also have that input symbol as a neighbor, the recovered input symbol is removed as a neighbor from each of these encoding symbols and the degree of each such encoding symbol is decreased by one to reflect this removal.*

2.1 The LT process

We introduce the LT process to help describe the design and analysis of a good degree distribution for LT codes. Let K be the total number of encoding symbols to be considered in the analysis (typically K is slightly larger than k). The LT process is a novel generalization of the classical process of throwing balls randomly into bins. A well known analysis of the classical process shows that $K = k \cdot \ln(k/\delta)$ balls are necessary on average to ensure that each of the k bins is covered by at least one ball with probability at least $1 - \delta$. In the analysis of the LT process, encoding symbols are analogous to balls and input symbols are analogous to bins, and the process succeeds if at the end all input symbols are covered.

Definition 2 *(LT process): All input symbols are intially uncovered. At the first step all encoding symbols with one neighbor are released to cover their unique neighbor. The set of covered input symbols that have not yet been processed is called the ripple, and thus at this point all covered input symbols are in the ripple. At each subsequent step one input symbol in the ripple is processed: it is removed as a neighbor from all encoding symbols which have it as a neighbor and all such encoding symbols that subsequently have exactly one remaining neighbor are released to cover their remaining neighbor. Some of these neighbors may have been previously uncovered, causing the ripple to grow, while others of these neighbors may have already been in the ripple, causing no growth in the ripple. The process ends when the ripple is empty at the end of some step. The process fails if there is at least one uncovered input symbol at the end. The process succeeds if all input symbols are covered by the end.*

In the classical balls and bins process, all the balls are thrown at once. Because of the high probability of collisions, i.e., multiple balls covering the same bin, many more balls must be thrown to cover all bins than there are bins. In contrast, the proper design of the degree distribution ensures that the LT process releases encoding symbols incrementally to cover the input symbols. Initially only a small fraction of the encoding symbols are of degree one and thus they cover only a small fraction of the input symbols. Covered input symbols are processed one at a time, and each one processed releases potentially other encoding symbols to randomly cover unprocessed input symbols. The goal of the degree distribution design is to slowly release encoding symbols as the process evolves to keep the ripple small so as to prevent redundant coverage of input symbols in the ripple by multiple encoding symbols, but at the same time to release the encoding symbols fast enough so that the ripple does not disappear before the process ends.

Because the neighbors of an encoding symbol are chosen at random independently of all other encoding symbols, it is easy to verify that the step at which a particular encoding symbol is released is independent of all the other encoding symbols. Furthermore, once an encoding symbol is released it covers a uniformly chosen input symbol among the unprocessed input symbols, independent of all other encoding symbols. These properties make the LT process especially amenable to the design of good degree distributions and analysis of the process with respect to these distributions.

It is not hard to verify that there is a one to one correspondence between the LT process and the decoder, i.e., an encoding symbol covers an input symbol if and only if the encoding symbol can recover that input symbol. Thus, the LT process succeeds if and only if the decoder successfully recovers all symbols of the data. The total number of encoding symbols needed to cover all the input symbols corresponds exactly to the total number of encoding symbols needed to recover the data. The sum of the degrees of these encoding symbols corresponds exactly to the total number of symbol operations needed to recover the data.

3. LT Degree Distributions

Recall that each encoding symbol has a degree chosen independently from a degree distribution.

Definition 3 *(degree distribution): For all d, $\rho(d)$ is the probability that an encoding symbol has degree d.*

As we now develop, the random behavior of the LT process is completely determined by the degree distribution $\rho(\cdot)$, the number of encoding symbols K, and the number of input symbols k. Our objective is to design degree distributions that meet the following two design goals.

- As few encoding symbols as possible on average are required to ensure success of the LT process. Recall that the number of encoding symbols that ensure success of the LT process corresponds to the number of encoding symbols that ensure complete recovery of the data.
- The average degree of the encoding symbols is as low as possible. The average degree is the number of symbol operations on average it takes to generate an encoding symbol. The average degree times K is the number of symbol operations on average it takes to recover the entire data.

The classical process of throwing balls into bins can be viewed as a special case of the LT process where all encoding symbols have degree one and thus are all released and thrown initially, i.e. the following distribution.

Definition 4 *(All-At-Once distribution):* $\rho(1) = 1$.

In terms of LT codes, this corresponds to generating each encoding symbol by selecting a random input symbol and copying its value to the encoding symbol. The analysis of the classical balls and bins process implies that $k \cdot \ln(k/\delta)$ encoding symbols are needed to cover all k input symbols with probability at least $1 - \delta$ with respect to the All-At-Once distribution. This result can be easily modified to show that for any distribution the sum of the degrees of the encoding symbols must be at least $k \cdot \ln(k/\delta)$ to cover all input symbols at least once. Thus, although the total number of symbol operations with respect to the All-At-Once distribution is minimal, the number of encoding symbols required to cover the k input symbols is an unacceptable $\ln(k/\delta)$ times larger than the minimum possible. Below we develop the Soliton distribution that ensures that just over k encoding symbols with the sum of their degrees being $O(k \cdot \ln(k/\delta))$ suffice to cover all k input symbols. Thus, both the number of encoding symbols and the total number of symbol operations are near minimal with respect to the Soliton distribution.

3.1 Some preliminary probabilistic analysis

We first analyze the probability that a particular encoding symbol of degree i is released when L input symbols remain unprocessed. Since the input symbols neighbors of an encoding symbol are chosen independently of all other encoding symbols, it follows that the probability that this encoding symbol is released when L input symbols remain unprocessed is independent of all other encoding symbols.

Definition 5 *(encoding symbol release): Let us say that an encoding symbol is released when L input symbols remain unprocessed if it is released by the processing of the $k - L^{th}$ input symbol, at which point the encoding symbol randomly covers one of the L unprocessed input symbols.*

Definition 6 *(degree release probability): Let $q(i, L)$ be the probability that an encoding symbol of degree i is released when L input symbols remain unprocessed.*

Proposition 7 *(degree release probability formula)*

- $q(1, k) = 1$.
- *For $i = 2, \ldots, k$, for all $L = k - i + 1, \ldots, 1$,*

$$q(i, L) = \frac{i(i-1) \cdot L \cdot \prod_{j=0}^{i-3} k - (L+1) - j}{\prod_{j=0}^{i-1} k - j}$$

- *For all other i and L, $q(i, L) = 0$.*

Proof: This is the probability that $i-2$ of the neighbors of the encoding symbol are among the first $k-(L+1)$ symbols processed, one neighbor is processed at step $k - L$, and the remaining neighbor is among the L unprocessed input symbols. $\square$

Definition 8 *(overall release probability): Let $r(i, L)$ be the probability that an encoding symbol is chosen to be of degree i and is released when L input symbols remain unprocessed, i.e. $r(i, L) = \rho(i) \cdot q(i, L)$. Let $r(L)$ be the overall probability that an encoding symbol is released when L input symbols remain unprocessed, i.e., $r(L) = \sum_i r(i, L)$.*

3.2 The Ideal Soliton distribution

The basic property required of a good degree distribution is that input symbols are added to the ripple at the same rate as they are processed. This property is the inspiration for the name Soliton distribution, as a soliton wave is one where dispersion balances refraction perfectly [29].

A desired effect is that as few released encoding symbols as possible cover a input symbol that is already in the ripple. This is because released encoding symbols that cover input symbols already in the ripple are redundant. Since released encoding symbols cover a random input symbol from among the unprocessed input symbols, this implies that the ripple size should be kept small at all times. On the other hand, if the ripple vanishes before all k input symbols are covered then the overall process ends in failure. The ripple size should be kept large enough to ensure that the ripple does not disappear prematurely. The desired behavior is that the ripple size never gets too small or too large.

The Ideal Soliton distribution displays ideal behavior in terms of the expected number of encoding symbols needed to recover the data. Unfortunately, like most ideal things, this distribution is quite fragile, in fact so much so that it is useless in practice. However, its description and analysis captures many of the crucial elements of the robust distributions described later.

Definition 9 *(Ideal Soliton distribution): The Ideal Soliton distribution is $\rho(1), \ldots, \rho(k)$, where*

- $\rho(1) = 1/k$
- *For all $i = 2, \ldots, k$, $\rho(i) = 1/i(i-1)$.*

Note that $\sum_i \rho(i) = 1$ as required of a probability distribution. One way to choose a sample from this distribution is to first choose a random real value $v \in (0, 1]$, and then for $i = 2, \ldots, k$, let the sample value be i if $1/i < v \leq 1/i - 1$, and let the sample value be 1 if $0 < v \leq 1/k$. The expected degree of an encoding symbol is $\sum_{i=1}^{k} i/i(i-1) = \mathrm{H}(k)$, where $\mathrm{H}(k) \approx \ln(k)$ is the harmonic sum up to k.

Proposition 10 *(uniform release probability): For the Ideal Soliton distribution, $r(L) = 1/k$ for all $L = k, \ldots, 1$.*

Proof: For $L = k$ all encoding symbols of degree one are released, and an encoding symbol is of degree one with probability $1/k$, and thus the statement is true for $L = k$. For all other values of L, from Proposition 7,

$$r(i, L) = \frac{L \cdot \prod_{j=0}^{i-3} k - (L+1) - j}{\prod_{j=0}^{i-1} k - j}$$

It can be verified that $k \cdot r(i, L)$ can be interpreted as the probability that when throwing balls uniformly at random among $k - 1$ bins, eliminating each bin as it is covered by a ball, that it is the $i - 1^{rst}$ ball thrown that lands in one of L designated bins. These events are mutually exclusive for different values of i, and since $i = 2, \ldots, k - L + 1$ covers all the possible outcomes,

$$k \cdot r(L) = k \cdot \sum_{i=1}^{k-L+1} r(i, L) = 1.$$

□

Suppose the expected behavior of the LT process is its actual behavior. Then, the Ideal Soliton distribution works perfectly, i.e. exactly k encoding symbols are sufficient to cover each of the k input symbols exactly once. This is because if there are k encoding symbols then by Proposition 10 exactly one encoding symbol is expected to be released initially and exactly one encoding symbol is expected to be released each time an input symbol is processed. If the expected behavior actually happens, then there is always exactly one input symbol in the ripple, and when this symbol is processed the released encoding symbol covers an input symbol among the unprocessed input symbols to reestablish this condition, etc.

However, this heuristic analysis makes the completely unrealistic assumption that the expected behavior is the actual behavior, and this is far from the truth. In fact, the Ideal Soliton distribution works very poorly in practice because the expected size of the ripple is one, and even the smallest variance causes the ripple to vanish and thus the overall process fails to cover and process all input symbols.

Note that for the Ideal Soliton distribution the sum of the degrees of k encoding symbols is on average $k \cdot \ln(k)$. Recall that for the All-At-Once distribution it takes on average $k \cdot \ln(k)$ encoding symbols to cover all input symbols with constant probability. It is interesting that the number of symbol operations for the Ideal Soliton distribution and for the All-At-Once distribution coincide, although the number of encoding symbols is quite different. The intuition for this is that for any distribution the sum of the degrees of all the encoding symbols needs to be around $k \cdot \ln(k)$ in order to cover all the input symbols, but the Ideal Soliton distribution compresses this into the fewest number of encoding symbols possible. Thus, the total amount of computation is the same in both cases (since it is proportional to the sum of the degrees of all the encoding symbols). However, the total amount of information that needs to be received (which is proportional to the number of encoding symbols) is compressed to the minimum possible with the Ideal Soliton distribution.

3.3 The Robust Soliton distribution

Although the Ideal Soliton distribution works poorly in practice, it does give insight into a robust distribution. The problem with the Ideal Soliton distribution is that the expected ripple size (one) is too small. Any variation in the ripple size is likely to make the ripple disappear and then the overall process fails. The Robust Soliton distribution ensures that the expected size of the ripple is large enough at each point in the process so that it never disappears completely with high probability. On the other hand, in order to minimize the overall number of encoding symbols used, it is important to minimize the expected ripple size so that not too many released encoding symbols redundantly cover input symbols already in the ripple.

Let δ be the allowable failure probability of the decoder to recover the data for a given number K of encoding symbols. The idea here is to design the distribution so that the expected ripple size is about $\ln(k/\delta)\sqrt{k}$ throughout the process. The intuition is that the probability a random walk of length k deviates from its mean by more than $\ln(k/\delta)\sqrt{k}$ is at most δ. As we describe below, it turns out this can be achieved using $K = k + O(\ln^2(k/\delta)\sqrt{k})$ encoding symbols.

Definition 11 *(Robust Soliton distribution): The Robust Soliton distribution is $\mu(\cdot)$ defined as follows. Let $R = c \cdot \ln(k/\delta)\sqrt{k}$ for some suitable constant $c > 0$. Define*

$$\tau(i) = \begin{cases} R/ik & \texttt{for } i = 1,\ldots,k/R-1 \\ R\ln(R/\delta)/k & \texttt{for } i = k/R \\ 0 & \texttt{for } i = k/R+1,\ldots,k \end{cases}$$

Add the Ideal Soliton distribution $\rho(\cdot)$ to $\tau(\cdot)$ and normalize to obtain $\mu(\cdot)$:

- *$\beta = \sum_{i=1}^{k} \rho(i) + \tau(i)$.*
- *For all $i = 1,\ldots,k$, $\mu(i) = (\rho(i) + \tau(i))/\beta$.*

The intuition for the supplement $\tau(\cdot)$ is the following. At the beginning, $\tau(1)$ ensures that the ripple starts off at a reasonable size. Consider the process in the middle. Suppose that an input symbol is processed and L input symbols remain unprocessed. Since the ripple size decreases by one each time an input symbol is processed, on average the ripple should be increased by one to make up for this decrease. If the ripple size is R then the chance that a released encoding symbol adds to the ripple is only $(L-R)/L$. This implies that at this point it requires $L/(L-R)$ released encoding symbols on average to add one to the ripple. From Proposition 7 it is possible to verify that the release rate of encoding symbols of degree i for i within a constant factor of k/L make up a constant portion of the release rate when L input symbols remain unprocessed. Thus, if the ripple size is to be maintained at approximately R, then the density of encoding symbols with degree $i = k/L$ should be proportional to

$$\begin{aligned} \frac{L}{i(i-1)\cdot(L-R)} &= \frac{k}{i(i-1)\cdot(k-iR)} \\ = \frac{1}{i(i-1)} + \frac{R}{(i-1)(k-iR)} &\approx \rho(i) + \tau(i), \end{aligned}$$

for $i = 2,\ldots,k/R-1$. The final spike $\tau(k/R)$ ensures that all the input symbols unprocessed when $L = R$ are all covered. This is similar to simultaneously releasing $R\ln(R/\delta)$ encoding symbols when R input symbols remain unprocessed to cover them all at once. Thus, the wastage caused by releasing enough encoding symbols to cover each of these input symbols at least once is only a small fraction of the total number k of input symbols.

We set the number of encoding symbols to $K = k\beta$. This implies that $k \cdot (\rho(i) + \tau(i))$ is the expected number of encoding symbols of degree i.

3.4 Analysis of Robust Soliton distribution

In this subsection we provide a theoretical analysis of the properties of the Robust Soliton distribution. As is often the case when proving theorems, in several places we make pessimistic estimates that enable a simple, comprehensive, and complete analysis. Heuristic techniques can be used to provide a design and analysis that leads to lower reception overhead and average degree based on computer simulations, but the description of this is beyond the scope of this paper.

Theorem 12 *(number of encoding symbols): The number of encoding symbols is $K = k + O(\sqrt{k} \cdot \ln^2(k/\delta))$.*

Proof:

$$\begin{aligned} K &= k\beta = k \cdot \left(\sum_i \rho(i) + \tau(i)\right) \\ &= k + \sum_{i=1}^{k/R-1} \frac{R}{i} + R\ln(R/\delta) \\ &\leq k + R \cdot \mathrm{H}(k/R) + R \cdot \ln(R/\delta). \end{aligned}$$

□

Theorem 13 *(average degree of an encoding symbol): The average degree of an encoding symbol is $D = O(\ln(k/\delta))$.*

Proof:

$$\begin{aligned} D &= \frac{\sum_i i \cdot (\rho(i) + \tau(i))}{\beta} \\ &\leq \sum_i i \cdot (\rho(i) + \tau(i)) \\ &= \sum_{i=2}^{k+1} \frac{1}{i-1} + \sum_{i=1}^{k/R-1} \frac{R}{k} + \ln(R/\delta) \\ &\leq \mathrm{H}(k) + 1 + \ln(R/\delta). \end{aligned}$$

□

The following propositions are used in the proof of the theorem below that the LT process succeeds with high probability.

Proposition 14 *(robust uniform release probability): For all $L = k-1,\ldots,R$, $K \cdot r(L) \geq L/(L-\theta R)$ for a suitable constant $\theta \geq 0$, excluding the contribution of $\tau(k/R)$.*

Proof: (Sketch) This proof uses the contributions of $\tau(2),\ldots,\tau(k/R-1)$ and the Ideal Soliton distribution $\rho(\cdot)$.

For $L = k/2, \ldots, k-1$, using Proposition 7 and Proposition 10,

$$\begin{aligned} K \cdot r(L) &\geq K \cdot \left(\frac{\left(\sum_i \frac{1}{i(i-1)} \cdot q(i,L)\right) + \tau(2) \cdot q(2,L)}{\beta} \right) \\ &= 1 + k \cdot \tau(2) \cdot q(2,L) \\ &= 1 + \frac{RL}{k(k-1)} \geq \frac{L}{L - R/6}. \end{aligned}$$

More generally, for $L \geq R$,

$$K \cdot r(L) \geq 1 + k \cdot \sum_{d=k/2L}^{k/L} \tau(d) \cdot q(d,L)$$

Then, using Proposition 7 and Proposition 10,

$$\begin{aligned} k \quad &\cdot \sum_{d=k/2L}^{k/L} \tau(d) \cdot q(d,L) \\ &= \sum_{d=k/2L}^{k/L} \frac{RL(d-1)}{k(k-1)} \cdot \prod_{j=0}^{d-3} \left(1 - \frac{L-1}{k-j-2}\right). \end{aligned}$$

For all $v = k/2L, \ldots, k/L$,

$$\prod_{j=0}^{d-3} \left(1 - \frac{L-1}{k-j-2}\right) \geq \left(1 - \frac{L}{k(1 - \frac{1}{L} - \frac{2}{k})}\right)^{\frac{k}{L}-3} \approx 1/\mathsf{e}.$$

Thus,

$$k \cdot \sum_{d=k/2L}^{k/L} \tau(d) \cdot q(d,L) \gtrapprox \frac{R}{8\mathsf{e}L}.$$

Putting this together yields

$$K \cdot r(L) \gtrapprox 1 + \frac{R}{8\mathsf{e}L} \geq \frac{L}{L - \theta R}$$

for $\theta = \frac{1}{16\mathsf{e}}$. □

Proposition 15 *(robust release at end probability): Using only the contribution of $\tau(k/R)$, $K \cdot \sum_{L=R}^{2R} r(L) \geq \gamma \cdot R \cdot \ln(R/\delta)$ for a suitable constant $\gamma > 0$.*

Proof: (Sketch) Fix L between $2R$ and R. It is not hard to show that $K \cdot \tau(k/R)/\beta \cdot q(k/R, L) \geq \gamma \cdot \ln(R/\delta)$ for an appropriate constant $\gamma > 0$. □

Proposition 16 *(pessimistic filtering): Suppose a particular encoding symbol is released when L input symbols remain unprocessed and there are X input symbols in the ripple. If the probability that the encoding symbol lands on an input symbol not currently in the ripple is lowered to any value that is at most $(L-X)/L$ then the success probability of the LT process is at most what it was before the modification.*

Theorem 17 *(high probability recovery): The decoder fails to recover the data with probability at most δ from a set of K encoding symbols.*

Proof: (Sketch) The main point of the proof is that the ripple size variation is very similar to a random walk, and thus the probability that the ripple size deviates from its expected value in k steps by a multiple of $\sqrt{k}$ is small.

Since the expected number of encoding symbols of degree one is R, a Chernoff bound shows that with probability at least $1 - \delta/3$ the initial size of the ripple due to encoding symbols of degree one is at least $\theta R/2$, and we can filter the original ripple size down to $\theta R/2$.

The argument will be made that the ripple does not disappear for $L = k-1, \ldots, R$ with high probability, and this portion of the argument will not use the $\tau(k/R)$ spike added to the $\mu(\cdot)$ distribution, as the $\tau(k/R)$ spike will be used separately to show that the last R input symbols are covered. Using Proposition 16 and Proposition 14, we can pessimistically filter the distribution for each of the K encoding symbols so that $r(L) = L/((L - \theta R)K)$. Let X_L be the number of encoding symbols that are released due to the processing of the input symbol that leaves L input symbols unprocessed in this filtered distribution, i.e. $\mathrm{E}[X_L] = L/(L - \theta R)$.

Assume for now that the ripple size is always between zero and θR, and later this assumption is removed. Each time an encoding symbol is released when L input symbols remain unprocessed it adds to the ripple with probability at least $(L - \theta R)/L$. Using Proposition 16, each such encoding symbol released can be pessimistically filtered so that it adds to the ripple with probability exactly $(L - \theta R)/L$. Let Y_L be the zero-one valued random variable such that $\Pr[Y_L = 1] = (L - \theta R)/L$. Note that for any $I \subseteq \{R, \ldots, k-1\}$,

$$\theta R/2 + \sum_{L' \in I} X_{L'} \cdot Y_{L'} - (k - L)$$

is the size of the ripple when L input symbols remain unprocessed under the above assumption. Note that

$$\begin{aligned} &\left| \sum_{L' \in I} X_{L'} \cdot Y_{L'} - (k-L) \right| \\ \leq \quad &\left| \sum_{L' \in I} X_{L'} \cdot Y_{L'} - \mathrm{E}[X_{L'}] \cdot Y_{L'} \right| \\ + \quad &\left| \sum_{L' \in I} \mathrm{E}[X_{L'}] \cdot Y_{L'} - \mathrm{E}[X_{L'}] \cdot \mathrm{E}[Y_{L'}] \right| \\ + \quad &\left| \sum_{L' \in I} \mathrm{E}[X_{L'}] \cdot \mathrm{E}[Y_{L'}] - (k-L) \right|. \end{aligned}$$

Using Chernoff Bounds, it can be shown that

$$\Pr\left[\left| \sum_{L' \in I} X_{L'} \cdot Y_{L'} - \mathrm{E}[X_{L'}] \cdot Y_{L'} \right| \geq \theta R/4 \right] \leq \delta/6k$$

and

$$\Pr\left[\left|\sum_{L'\in I}\mathrm{E}[X_{L'}]\cdot Y_{L'} - \mathrm{E}[X_{L'}]\cdot\mathrm{E}[Y_{L'}]\right| \geq \theta R/4\right] \leq \delta/6k.$$

Furthermore,

$$\sum_{L'\in I}\mathrm{E}[X_{L'}]\cdot\mathrm{E}[Y_{L'}] - (k-L) = 0.$$

From this it follows that under the assumption the probability the ripple size varies from its original value of $\theta R/2$ by at least $\theta R/2$ over the interval I is at most $\delta/3k$. This shows that

$$\Pr\left[\left|\sum_{L'\in I}X_{L'}\cdot Y_{L'} - (k-L)\right| \geq \theta R/2\right] \leq \delta/3k,$$

it can be concluded the ripple size is greater than zero and at most θR at the end of interval I with probability at least $1-\delta/3k$. Since there are $k-R$ intervals, this shows that the ripple is strictly above zero and below θR for all intervals. This shows that the assumption made about the maximum ripple size is true and the process is successful until R input symbols remain unprocessed with probability at least $1-\delta/3$.

We now consider the covering and processing of the last R symbols. Proposition 15 can be used to show that enough encoding symbols are released between when $2R$ and R input symbols remain unprocessed to cover with probability at least $1-\delta/3$ the final R unprocessed input symbols.

Overall, the LT process completes successfully with probability at least $1-\delta$. □

One interesting aspect is the comparison between the LT process with respect to the All-At-Once distribution and the Robust Soliton distribution. For both the sum of the degrees of the encoding symbols needed to cover all k of the input symbols is approximately $k\ln(k/\delta)$. However, since each encoding symbol is of degree one with respect to the All-At-Once distribution which corresponds to the classical process, it also takes approximately $k\ln(k/\delta)$ encoding symbols to cover all of the input symbols. With the Robust Soliton distribution, the same amount of degree is distributed essentially among the smallest number of encoding symbols possible, with the net result that it takes close to the minimum number of encoding symbols to recover all k input symbols.

References

[1] M. Adler, Y. Bartal, J.W. Byers, M. Luby, D. Raz. A Modular Analysis of Network Transmission Protocols Proceedings of *Fifth Israeli Symposium on Theory of Computing and Systems*, June 1997.

[2] J. Bloemer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, D. Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. *ICSI TR-95-048*, Technical report at ICSI, August 1995.

[3] J.W. Byers, J. Considine, M. Mitzenmacher, S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. Proceedings of *ACM SIGCOMM 2002*, Pittsburgh PA, August 19 – 23, 2002.

[4] J.W. Byers, M. Frumin, G. Horn, M. Luby, M. Mitzenmacher, A. Roetter, W. Shaver. FLID-DL: Congestion control for layered multicast. Proceedings of *2nd Int. Workshop Netw. Group Comm.*, pages 71–81, Stanford, California, November 2000.

[5] J.W. Byers, M. Luby, M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. Proceedings of *IEEE INFOCOM '99*, New York, NY, pp. 275 – 283.

[6] J.W. Byers, M. Luby, M. Mitzenmacher. A Digital Fountain Approach to Asynchronous Reliable Multicast. *IEEE J. on Selected Areas in Communications, Special Issue on Network Support for Multicast Communications*, 2002.

[7] J.W. Byers, M. Luby, M. Mitzenmacher, A. Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. Proceedings of *ACM SIGCOMM '98*, pages 56–67, Vancouver, September 1998.

[8] S. Floyd, V. Jacobson, C.G. Lin, S. McCanne, L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. Proceedings of *ACM SIGCOMM '95*, pp. 343-356, August 1995.

[9] J. Gemmell. ECRSM - Erasure Correcting Scalable Reliable Multicast. *Microsoft Research Technical Report MS-TR-97-20*, June 1997.

[10] T.V. Lakshman, U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products. *IEEE/ACM Trans. Networking*, 5(3):336–350, June 1997.

[11] J.C. Lin, S. Paul. RMTP: A Reliable Multicast Transport Protocol. Proceedings of *IEEE INFOCOM '96*, pp. 1414-1424, March 1996.

[12] M. Luby. Information Additive Code Generator and Decoder for Communication Systems. U.S. Patent No. 6,307,487, Oct. 23, 2001.

[13] M. Luby. Information Additive Code Generator and Decoder for Communication Systems. US Patent No. 6,373,406, April 16, 2002.

[14] M. Luby, V. K Goyal, S. Skaria, G. Horn. Wave and Equation Based Rate Control using Multicast Round-trip Time. Proceedings of *ACM SIGCOMM 2002*, Pittsburgh PA, August 19 – 23, 2002.

[15] M. Luby, M. Mitzenmacher, A. Shokrollahi. Analysis of Random Processes via And-Or Tree Evaluation. Proceedings of *9th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, January 25–27, 1998.

[16] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, V. Stemann. Practical Loss-Resilient Codes. Proceedings of *9th Annual ACM Symposium on Theory of Computing*, 1997.

[17] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman. Efficient Erasure Correction Codes. *IEEE Trans. on Information Theory, Special Issue on Codes and Graphs and Iterative Algorithms*, Vol. 47, No. 2, Feb. 2001.

[18] F. J. MacWilliams, N.J.A. Sloane. The Theory of Error-Correcting Codes. North Holland, Amsterdam, 1977.

[19] S. McCanne, V. Jacobson, M. Vetterli. Receiver-driven Layered Multicast. Proceedings of *ACM SIGCOMM '96*, pp. 117-130, 1996.

[20] C. K. Miller. Reliable Multicast Protocols: A Practical View. Proceedings of *22nd Annual Conference on Local Computer Networks (LCN '97)*, 1997.

[21] J. Nonnenmacher, E.W. Biersack. Reliable Multicast: Where to Use Forward Error Correction. Proceedings of *IFIP 5th Int'l Workshop on Protocols for High Speed Networks*, pp. 135-148, Sophia Antipolis, France, October 1996, Chapman and Hall.

[22] J. Nonnenmacher, E.W. Biersack. Asynchronous Multicast Push: AMP. Proceedings of *International Conference on Computer communications*, Cannes, France, November 1997.

[23] J. Nonnenmacher, E.W. Biersack, D. Towsley. Parity-Based Loss Recovery for Reliable Multicast Transmissions. Proceedings of *ACM SIGCOMM '97*, 1997.

[24] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, Vol. 38, pp. 335-348, 1989.

[25] I.S. Reed, G. Solomon. Polynomial Codes Over Certain Finite Fields. *J. Soc. Indust. Appl. Math*, Vol. 8, pp. 300-304, 1960.

[26] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *Computer Communication Review*, April 1997.

[27] L. Rizzo, L. Vicisano. A Reliable Multicast Data Distribution Protocol Based on Software FEC Techniques. Proceedings of *HPCS '97*, Greece, June 1997.

[28] E. Schooler, J. Gemmell. Using multicast FEC to solve the midnight madness problem. *Microsoft Research Technical Report MS-TR-97-25*, September, 1997.

[29] J. Scott. Report on Waves. *Report of the fourteenth meeting of the British Association for the Advancement of Science*, York, September 1844 (London 1845), pp 311-390, Plates XLVII-LVII)

[30] L. Vicisano, L. Rizzo, J. Crowcroft. TCP-like congestion control for layered multicast data transfer. Proceedings of *IEEE INFOCOM*, volume 3, pages 996–1003, San Francisco, California, March–April 1998.

[31] J. Widmer, M. Handley. Extending equation-based congestion control to multicast applications. Proceedings of *ACM SIGCOMM*, pages 275–286, San Diego, California, August 2001.

[32] R. Yavatkar, J. Griffoen, M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. Proceedings of *ACM Multimedia '95*, San Francisco, California, 1995, pp. 333-344.