Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 Introduction

Multicast or broadcast transmission is an immediate solution for software companies that intend to spread new software over the Internet directed towards million of users. This is one of the several applications that this kind of transmissions can have nowadays, and this is motivated by the fact that currently there is an incresing proliferation of applications that must reliably distribute huge amount of data. For the nature of the problem, these transmissions have to be fully reliable, support a vast number of receivers and have a low network overhead. All these requirements together imply that retransmission of lost or corrupted packets is not feasible, as the clients requests for retransmission can rapidly overwhelm the server, with consequent *feedback implosion*. [1] These problems have led researchers to focus more on the application of erasure codes in Forward Error Correction (FEC) techniques for reliable multicast. Erasure codes are centered around the simple principle that the sender can transmit the k packets, which we could represent the original source data with, together with additional redundant packets that can subsequently be used to recover lost source data at the receiver. This receiver can reconstruct the original

source data once it has received a sufficient number of packets and a big benefit in this is that different receivers can recover different lost packets using the same redundant data.

A good application of this concept can be seen in the figure of *digital fountain codes*, a better implementation than the one involving Reed-Solomon codes.[1]

2 The digital fountain

The digital fountain approach takes its name from the analogy with spraying water drops that are collected into a bucket. When the bucket is being filled by a fountain, we are not interested in knowing which droplets are falling into the bucket, as long as it is being filled. Once the filling part is done, the collection process ends and the content decoding will start. [2]

In tha same way, this approach injects a flow of distinct encoding packets into the network in such a way that the source data can be reconstructed entirely from any subset of them having the same length, in total, as the source data. A straightforward implementation of a digital fountain directly uses an erasure code that takes k-packets source data and produces as many encoded packets as the user demand needs. Thus, using for example the Reed-Solomon erasure code in such a way that the client side decoder can recontstruct the original source data whenerver k transmitted packets are received, the file is *stretched* into n encoding packets, with $\frac{n}{k}$ being the *stretch factor* or erasure codes. This technique comes with some side effects, one of which is the decreased efficiency of encoding and decoding operations: with the standard Reed-Solomon codes, for example, the behaviour is prohibitive even for small values of k and n. Hence, an alternative to these codes is proposed in [1], by using the so-called *Tornado codes*. In this codes, the equations have the form $y_3 = x_2 \oplus x_7 \oplus x_9$ where $\oplus$ represents the *bitwise* exclusive or and is used in place of the normal exclusive or $+$. On the other side, the decoding process of the Tornado codes uses two basic operations that are:

Table 1: Properties of Tornado and Reed-Solomon codes

	Tornado	Reed-Solomon
Decoding inefficiency	$1+\epsilon$ required	klP
Encoding Times	$(k+l)ln(1/\epsilon)P$	klP
Decoding Times	$(k+l)ln(1/\epsilon)P$	klP
Basic Operation	XOR	Field operation

- replacing the received variables with theyr values in the equations they appear in;
- a simple *subsitution rule*;

Through these operations, Tornado codes avoid both the filed operations and the matrix inversion inherent in decoding Reed-Solomon codes. Furthermore, while the latter are built using a dense system of linear equations, the former are built using random *sparse* equations, and this brings to a significant reduction in decoding time. The price we pay for faster coding is that k packets are no longer enough to reconstruct the source data, translating into higher decoding inefficiency. In practice, however, the number of possible substitution rule applications stay minimal as far as slightly more than k packets are received, and often a single arrival generates a whirlwind of subsections that permit to recover all the remaining source data packets. Hence the name of Tornado codes. We can conclude that the advantage of these codes is that they are a tradeoff between a small increase in decoding inefficiency and a significant decrease of encoding and decoding times. [1]. 1 highlights the main propoerties of the Tornado codes, compared to the ones of Reed-Solomon codes.

3 The LT-codes

A further improvement in the field of erasure codes is given by the important calss of LT codes, ratless codes introduced and described by Luby in [3]. LT codes are the first realization of universal erasure codes, meaning that the codes symbol length can be arbitrary. Moreover, these codes are rateless, which means that the number of encoding symbols that can be generated from the data is potentially limitless.

Comparing LT codes to the preceding Tornado codes, it can be seen that their analysis is slightly different: in particular the Tornado codes analysis can only be applied to graphs with constant maximum degree, while LT codes graphs are of logarithmic density. Moreover, while for the Tornado codes the reception overhead is at least a constant fraction of the data length, for the LT codes it is an asymptotically fading fraction of the data length, even though with the tradeoff of lightly higher asymptotic encoding and decoding times. Another considerable advantage of these codes is that, once k and n have been chosen, the Tornado encoder generates n symbols and cannot produce others on the fly like LT codes do. Finally, there is a notable difference in the degree distribution of input and encoding symbols: for Tornado codes the degree distribution in input symbols is similar to the so-called *Soliton distribution*, while the degree distribution of encoding symbols can be approxhimated by the Poisson distribution. This thing, though, cannot be applied to the LT codes, since the Soliton distribution cannot be the resulting degree distribution on input symbols when encoding symbols are generated independently, regardless of the distribution of neighbors of encoding symbols. [3]

3.1 The encoding process

In [3] Luby gives a complete explanation of the encoding and the decoding process, and the former can be here briefly described by the steps:

- From a degree distribution, randomly choose the degree d of the encoding symbol;

- Choose uniformly at random d distinct input symbols which can be neighbors of the encoding symbol;
- Perform the exclusive-or of the d neighbors to obtain the value of the encoding symbol;

NOTE: In addition to this, it should be kept in mind that the decoder needs know the set and degree of neighbors of each encoding symbol in order to recover the original input symbol, and for this there are several ways to communicate this to it.**(Cri says: metterlo?)**

There are several degree distributions that can be used for our encoding symbols and the choice of each one comes with different performance in terms of coding times, efficiency and Bit Error Rate (BER), which we will see further on in this work. In general the most important ones are:

- The *All-At-Ones distribution*: $\rho(d) = 1$;
- The *Ideal Soliton Distribution*: $\rho(1) = 1/k$ and $\forall i = 2, \ldots, k$ then $\rho(i) = 1/i(i-1)$;
- The *Robust Soliton Distribution*:
  - $\beta = sum_{i=1}^{k} \rho(i) + \tau(i)$
  - $\forall i = 1, \ldots, k$ then $\mu(i) = (\rho(i) + \tau(i))/\beta$

  with, for an $R = c \cdot ln(k/\delta)\sqrt{k}$ for some suitable constant $c > 0$, the quantity $\tau(i)$ being:

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

3.2 The decoding process

We can look at the quality of the coding scheme by analyzing the performance of the decoding process under different points of view, from the finite lenght analysis of the error probability [4], to the analysis of the success probability and the overhead dimension for a small number of packets N [5], to the opposite analysis with a high N and another decoding technique [2].

3.2.1 Error probability for the Belief Propagation (BP) decoder

In [4], the authors outline a method for calculating the success probabilty of the BP decoder. They start from defining the shape of the distribution $\Omega = (\Omega_1, \dots, \Omega_k)$, chosen on the set $\{1, \dots, k\}$, and which induces a distribution $\mathcal{D}$ on $\mathbb{F}_2^k$ we can choose our output symbols from. An LT code so described has parameters $(k, \Omega(x))$, with x being the input symbol considered and $\Omega(x) = \Omega_1 x + \Omega_2 x^2 + \cdots + \Omega_k x^k$ is the generating function of the distribution. The importance of the success probability of the decoder comes from the fact that the number n of encoding symbols to which we apply the BP strictly depends on it and the output degree distribution.

Later on they present a recursive approach centered on the probability distribution generating function $P_u(x, y)$.

(Cri says: Non mi convince, non abbiamo spiegato cos'è il BP decoder e nonn ha molto senso specificare un paper che già di per sè ha solo 1 facciata. Piuttosto concentrati meglio sugli altri)

3.2.2 Optimal degree distribution for small message length codes

Another point of view is given in [5] where, instead of focusing on the decoder, the authors decided to approach the problem from the perspective of finding the optimal degree distribution defining the number of blocks in each packet. They do so looking at the problem in two different ways: a Markov chain approach and a combinatorial one.

In the case of the Markov chain approach, from the receiver's point of view, the set of received and partially/fully decoded packets represents a state. At the same time, state transition probabilities depend on specific packets arrival probabilities, which in turn depend on the degree distribution used in the encoding process. From this it follows that the absorbing state is the one formed by the original blocks. It is worth notice that the state space needs only include the states that can not be reduced by the decoder (*irreducible*), plus further reductions can be performed playing on isomorphies on block paths as well as other tricks. However, the state space implosion is unavoidable and occurs already at the lowest values of number of blocks in the message N [5]. After these steps the transition probability matrix can be constructed like:

$$\mathbf{P} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \mathbf{0} & \boldsymbol{I} \end{pmatrix} \tag{31}$$

and we have $\boldsymbol{\pi_0} = (1, \ldots, 0)$ for the initial distribution vector and $\boldsymbol{\pi_{abs}} = (0, \ldots, 1)$ for the absorbing state. With the system set in this way, we can find the best degree distribution through two different criteria: either *minimizing the avg number of steps* needed for decoding, $E[T] = \boldsymbol{\pi_0}(\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{e}^T$, or *maximizing the success probability*, $\mathcal{P}_N = \boldsymbol{\pi_0}\boldsymbol{P}^N\boldsymbol{\pi}_{abs}$.

In the case of the combinatorial approach, the central observation is that in such case the orther in which the packets are received is irrelevant. The recursive algorithm that derives from it aims at calculating the probability $\mathcal{P}_N$ that a message consisting of N blocks is succesfully decoded with exactly N received packets. So in general we take this probability $\mathcal{P}_n = P_n(p_1, \ldots, p_n)$, with the trivial cases $P_0 = 1$ and $P_1(p_1) = p_1$ being the seeds for recursion, and condition it on $n - m$ of the n received packets having degree 1. Moreover we need to specify that having no duplicates and all the $n - m$ degree 1 packets distinct happens with probability $(n-1)!/m!n^{n-m-1}$ and this is needed to weight the remaining decoding problem, consituting the recursion, for the m other packets whose degree can be lowered from being ≥ 2 after removing the $n - m$ degree-1

packets. With these specifications, the recursive approach can be expressed as:

$$P_n(p_1,\ldots,p_n) = \sum_{m=0}^{n-1} \binom{n}{m} p_1^{n-m}(1-p_1)^m \tag{32}$$

$$\times \frac{(n-1)!}{m!n^{n-m-1}} P_m(p_1^{(n,m)},\ldots,p_m^{(n,m)}), \tag{33}$$

where

$$p_j^{(n,m)} = \sum_{i=2}^{n} \frac{p_i}{1-p_1} \frac{\binom{m}{j}\binom{n-m}{i-j}}{\binom{n}{i}}, \qquad j=1,\ldots,m \tag{34}$$

with the first fraction in 34 calculates the probability that a packet has degree i conditioned on that it is not of degree 1, while the second fraction is the probability that when "from an urn" containing n blocks, m of which are white, a degree-i packet is constructed, then the subsequent reduced degree is exactly j[5].

The results The performance analysis have been done comparing the obtained optimal results to four main distributions: the uniform distribution, the degree-1 distribution ($p_i = 1 \quad (i=1)$), the binomial distribution and the Soliton distribution. In the case of the Markov approach the authors show how the *MinAvg* and the *MaxPr* lead to similar results, which in general are better than the results from all the other distributions. In particular, Table 2 shows how Binomial and Soliton distribution work still reasonably well, but not the same can be said regarding the other two distributions. In the parallel case of the combinatorial approach, two main aspects are highlighted: first, with the state explosion not being such a problem now, the fact that we can analyze cases for values of N up to 30; second, being the eigenvalues of A (The second derivative matrix of $\mathcal{P}_N$) a rapidly decreasing sequence, the function is extremely insensitive to changes along the last eigenvectors. This bringing to the result that typically three conditions are enough to define a close-to-the-optimum distribution [5].

(Cri says: sistema la nomenclatura! Nel paragrafo di prima il nostro N era il k degli altri paragrafi, forse è meglio se ti attieni al k...)

	MinAvg	MAxPr	Binomial	Soliton	Uniform	Degree-1
p_1	0.524	0.517	3/7	1/3	1/3	1
p_2	0.366	0.397	3/7	1/2	1/3	0
p_3	0.109	0.086	1/7	1/6	1/3	0
$E[T]$	4.046	4.049	4.133	4.459	4.725	5.5
$\mathcal{P}_3$	0.451	0.452	0.437	0.397	0.354	0.222

Table 2: Optimal performance and weights in the case N = 3

3.2.3 Optimal decoding algorithm for high values of k

The approach explained in [2] is slightly different and is based also on the assumption, showed in [6], that LT codes perform very well for big values of k. In this way the authors underline the difference of their approach form the work of Hyytä *et al.* presented in the previous paragraph, since those methods already appeared to be not scalable and being able only to handle symbols $k \leq 10$.

In the LT decoding process, at each round a symbol is removed from the ripple and the related packets are updated accordingly. The process terminates succesfully when there is no more symbol left in the ripple and all symbols are recovered. Hence it is important not to let ripple size become 0 before all symbols are recovered. In this process the degree distribution plays a big role, since the arrival of new symbols in the ripple also depends in it.The algorithm proposed by Lu *et al.* in [2], called *full rank decoding*, is intended to extend de decodability of LT codes, preventing them from terminating prematurely. To accomplish that, whenever the ripple is empty and would cause an early termination, the method *borrows* a particular symbol, decodes it through some other method (in particular $Wiedeman$ algorithm) and places it in the ripple for the process to continue.

The criteria the authors use to choose the right symbol to $borrow$ is a maximization

problem described by:

$$b_j = argmax_j(V(j)|\mathbf{V} := \sum_{i=1}^{n} \mathbf{v_i}) \tag{35}$$

with the used quantities representing:

- j the index of the borrowed symbol b_j;
- $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}$ the coefficients vectors in the linear equations representing the encoding symbols;
- $V(j)$ the value at index j of the vector $\mathbf{V}$;

The results obtained in [2] show improved performance under the analysis of the expected number of packets needed for decoding, as well as the fact that the performance of the method is close to the ideal one. All of this resulting in in an overall lower number of packets needed to recover all the symbols and a lower runtime.

3.2.4 Relay techniques

(Cri says: add something related to the relay techniques, but in a very quick mention)

References

[1] J. W. Byers, M. Luby, and M. Mitzenmacher, "A Digital Fountain Approach to Reliable Distribution of Bulk Data,"

[2] F. Lu, "LT Codes Decoding : Design and Analysis," pp. 1–5.

[3] M. Luby, "LT Codes,"

[4] R. Karp and M. Luby, "Finite Length Analysis of LT Codes," p. 2004, 2004.

[5] E. Hyytiä, “Optimal Degree Distribution for LT Codes with Small Message Length,” pp. 2576–2580, 2007.

[6] A. Shokrollahi and S. Member, “Raptor Codes,” vol. 52, no. 6, pp. 2551–2567, 2006.