Raptor Codes

Amin Shokrollahi*

Laboratoire d'algorithmique
Laboratoire de mathematique algorithmique
École Polytechnique Fédérale de Lausanne
1015 Lausanne, Switzerland
amin.shokrollahi@epfl.ch

AND

Digital Fountain, Inc.
39141 Civic Center Drive
Fremont, CA 94538, USA
amin@digitalfountain.com

June 19, 2003

Abstract

LT-Codes are a new class of codes introduced in [1] for the purpose of scalable and fault-tolerant distribution of data over computer networks. In this paper we introduce Raptor Codes, an extension of LT-Codes with linear time encoding and decoding. We will exhibit a class of universal Raptor codes: for a given integer k, and any real $\varepsilon > 0$, Raptor codes in this class produce a potentially infinite stream of symbols such that any subset of symbols of size $k(1+\varepsilon)$ is sufficient to recover the original k symbols with high probability. Each output symbol is generated using $O(\log(1/\varepsilon))$ operations, and the original symbols are recovered from the collected ones with $O(k \log(1/\varepsilon))$ operations.

We will also introduce novel techniques for the analysis of the error probability of the decoder for finite length Raptor codes. Moreover, we will introduce and analyze systematic versions of

*Work on this project was done while the author was a full time employee of Digital Fountain, Inc.

Raptor codes, i.e., versions in which the first output elements of the coding system coincide with the original k elements.

1 Introduction

The binary erasure channel (BEC) of communication was introduced by Elias [2] in 1955, but it was regarded as a rather theoretical channel model until the large-scale deployment of the Internet about 40 years later.

On the Internet data is transmitted in the form of packets. Each packet is equipped with a header that describes the source and the destination of the packet, and often also a sequence number describing the absolute or relative position of the packet within a given stream. These packets are routed on the network from the sender to the receiver. Due to various reasons, for example buffer overflows at the intermediate routers, some packets may get lost and never reach their destination. Other packets may be declared as lost if the internal checksum of the packet does not match. Therefore, the Internet is a very good real-world model of the BEC.

Reliable transmission of data over the Internet has been the subject of much research. For the most part, reliability is guaranteed by use of appropriate protocols. For example, the ubiquitous TCP/IP ensures reliability by essentially re-transmitting packets within a transmission window whose reception has not been acknowledged by the receiver (or packets for which the receiver has explicitly sent a negative acknowledgment). It is well-known that such protocols exhibit poor behavior in many cases, such as transmission of data from one server to multiple receivers, or transmission of data over heavily impaired channels, such as poor wireless or satellite links. Moreover, ack-based protocols such as TCP perform poorly when the distance between the sender and the receiver is long, since large distances lead to idle times during which the sender waits for an acknowledgment and cannot send data.

For these reasons other transmission solutions have been proposed. One class of such solutions is based on coding. The original data is encoded using some linear erasure correcting code. If during the transmission some part of the data is lost, then it is possible to recover the lost data using erasure correcting algorithms. For applications it is crucial that the codes used are capable of correcting as many erasures as possible, and it is also crucial that the encoding and decoding algorithms for these codes are very fast.

Elias showed that the capacity of the BEC with erasure probability p equals $1 - p$. He further proved that random codes of rates arbitrarily close to $1 - p$ can be decoded on this channel with an exponentially small error probability using Maximum Likelihood (ML) decoding. In the case of the erasure channel ML decoding of linear codes is equivalent to solving systems of linear equations. This task can be done in polynomial time using Gaussian elimination. However, Gaussian elimination is not fast enough, especially when the length of the code is long.

Reed-Solomon codes can be used to partially compensate for the inefficiency of random codes. Reed-Solomon codes can be decoded from a block with the maximum possible number of erasures in time quadratic in the dimension. (There are faster algorithms based on fast polynomial arithmetic, but these algorithms are often too complicated in practice.) However, quadratic running times are still too large for many applications.

In [3] the authors construct codes with linear time encoding and decoding algorithms that can come arbitrarily close to the capacity of the BEC. These codes, called Tornado codes, are very similar to Gallager's LDPC-codes [4], but they use a highly irregular weight distribution for the underlying graphs.

The running times of the encoding and decoding algorithms for Tornado codes are proportional to their block-length rather than to their dimension. Therefore, for small rates the encoding and decoding algorithms for these codes are slow. This turns out to be quite limiting in many applications, such as those described in [5], since the codes used there are of extremely low rate. This suggests that the encoding/decoding times of traditional coding technologies may not be adequate for the design of scalable data transmission systems.

There are more disadvantages of traditional block codes when it comes to their use for data transmission. The model of a single erasure channel is not adequate for cases where data is to be sent concurrently from one sender to many receivers. In this case the erasure channels from the sender to each of the receivers have potentially different erasure probabilities. Typically in applications the sender or the receiver may probe their channels so the sender has a reasonable guess of the current erasure probability of the channel and can adjust the coding rate accordingly. But if the number of receivers is large, or in situations such as satellite or wireless transmission where receivers experience sudden abrupt changes in their reception characteristics, it becomes unrealistic to assume and keep track of the loss rates of individual receivers. The sender is then forced to assume a worst case loss rate for all the receivers. This not only puts unnecessary burdens on the network if the actual loss

rate is smaller, but also compromises reliable transmission if the actual loss rate is larger than the one provisioned for.

Therefore, to construct robust and reliable transmission schemes, a new class of codes is needed. Fountain Codes constitute such a class, and they address all the above mentioned issues. They were first mentioned without an explicit construction in [5]. A Fountain Code produces for a given set of k input symbols $(x_1, \ldots, x_k)$ a potentially limitless stream of output symbols $z_1, z_2, \ldots$. The input and output symbols can be bits, or more generally, they can be binary vectors of arbitrary length. The output symbols are produced independently and randomly, according to a given distribution on $\mathbb{F}_2^k$. Each output symbol is the addition of some of the input symbols, and we suppose that the output symbol is equipped with information describing which input symbols it is the addition of. In practice, this information can be either a part of the symbol (e.g., using a header in a packet), or it can be obtained via time-synchronization between the sender and the receiver, or it may be obtained by other application-dependent means. A decoding algorithm for a Fountain Code is an algorithm which can recover the original k input symbols from any set of n output symbols with high probability. For good Fountain Codes the value of n is very close to k, and the decoding time is close to linear in k.

Fountain Codes are ideally suited for transmitting information over computer networks. A server sending data to many recipients can implement a Fountain Code for a given piece of data to generate a potentially infinite stream of packets. As soon as a receiver requests data, the packets are copied and forwarded to the recipient. In a broadcast transmission model there is no need for copying the data since any outgoing packet is received by all the receivers. In other types of networks, the copying can be done actively by the sender, or it can be done by the network, for example if multicast is enabled. The recipient collects the output symbols, and leaves the transmission as soon as it has received n of them. At that time it uses the decoding algorithm to recover the original k symbols. Note that the number n is the same regardless of the channel characteristics between the sender and the receiver. More loss of symbols just translates to a longer waiting time to receive the n packets. If n can be chosen to be arbitrarily close to k, then the corresponding Fountain Code has a universality property in the sense that it operates close to capacity for *any* erasure channel with erasure probability less than 1.

Fountain Codes have also other very desirable properties. For example, since each output symbol is generated independently of any other one, a receiver may collect output symbols generated from

the same set of k input symbols, but by different devices operating a Fountain encoder. This allows for the design of massively scalable and fault-tolerant communication systems over packet based networks. In this paper we will not address these and other applications, but will instead focus on the theory of such codes.

In order to make Fountain Codes work in practice, one needs to ensure that they possess a fast encoder and decoder, and that the decoder is capable of recovering the original symbols from any set of output symbols whose size is close to optimal with high probability. We call such Fountain Codes *universal*. The first class of such universal Fountain Codes was invented by Luby [6, 7, 1]. The codes in this class are called LT-Codes.

The distribution used for generating the output symbols lies at the heart of LT-Codes. Every time an output symbol is generated in an LT-Code, a weight distribution is sampled which returns an integer d between 1 and the number k of input symbols. Then d random distinct input symbols are chosen, and their value is added to yield the value of that output symbol. Decoding of LT-Codes is similar to that of LDPC codes over the erasure channel and is described later in Section 3. Whether or not the decoding algorithm is successful depends solely on the weight distribution.

It can be shown (see Proposition 1) that if an LT-Code has a decoding algorithm with a probability of error that is at most inversely polynomial in the number of input symbols, and if the algorithm needs n output symbols to operate, then the average weight of an output symbol needs to be at least $ck \log(k)/n$ for some constant c. Hence, in the desirable case where n is close to k, the output symbols of the LT-Code need to have an average weight of $\Omega(\log(k))$. It is absolutely remarkable that it is possible to construct a weight distribution that matches this lower bound via a fast decoder. Such distributions were exhibited by Luby [1].

For many applications it is important to construct universal Fountain Codes for which the average weight of an output symbol is a constant and which have fast decoding algorithms. In this paper we introduce such a class of Fountain Codes, called *Raptor Codes*. The basic idea behind Raptor codes is a pre-coding of the input symbols prior to the application of an appropriate LT-Code (see Section 4). In the asymptotic setting, we will design a class of universal Raptor Codes with linear time encoders and decoders for which the probability of decoding failure converges to 1 polynomially fast in the number of input symbols. This will be the topic of Section 6.

In practical applications it is important to bound the error probability of the decoder. The bounds obtained from the asymptotic analysis of Section 6 are rather poor. Therefore, we develop in

Section 7 analytic tools for the design of finite length Raptor Codes which exhibit very low decoding error probabilities, and we will exemplify our methods by designing a specific Raptor Code with guaranteed bounds on its error performance.

One of the disadvantages of LT- or Raptor Codes is that they are not systematic. This means that the input symbols are not necessarily reproduced among the output symbols. The straightforward idea of transmitting the input symbols prior to the output symbols produced by the coding system is easily seen to be flawed, since this does not guarantee a high probability of decodability from any subset of received output symbols. In Section 8 we develop a new set of ideas and design efficient systematic versions of Raptor Codes.

Raptor Codes were discovered in the late 2000, and patented in late 2001 [8]. Independently, Maymounkov [9] later discovered the idea of pre-coding to obtain linear time codes. His results are similar to parts of Section 6.

Raptor codes have been highly optimized and are being used in commercial systems Digital Fountain (http://www.digitalfountain.com), a Silicon Valley based startup specializing in fast and reliable delivery of data over heterogeneous networks. the Raptor implementation of Digital Fountain reaches speeds of several gigabits per second, on a 2.4Ghz Intel Xeon processor, while ensuring very stringent conditions on the error probability of the decoder, even for very short lengths.

2 Distributions on $\mathbb{F}_2^k$

Let k be a positive integer. The dual space of $\mathbb{F}_2^k$ is the space of linear forms in k variables with coefficients in $\mathbb{F}_2$. This space is non-canonically isomorphic to $\mathbb{F}_2^k$ via the isomorphism mapping the vector $(a_1, \ldots, a_k)$ with respect to the standard basis to the linear form $a_1X_1 + \cdots + a_kX_k$. A probability distribution on $\mathbb{F}_2^k$ induces a probability distribution on the dual space of $\mathbb{F}_2^k$ with respect to this isomorphism. For the rest of this paper we will use this isomorphism and will freely and implicitly interchange distributions on $\mathbb{F}_2^k$ and its dual.

Let $\Omega_1, \ldots, \Omega_k$ be a distribution on $\{1, \ldots, k\}$ so that Ω_i denotes the probability that the value i is chosen. Often we will denote this distribution by its generator polynomial $\Omega(x) = \sum_{i=1}^{k} \Omega_i x^i$. For example, using this notation, the expectation of this distribution is succinctly given by $\Omega'(1)$, where $\Omega'(x)$ is the derivative of $\Omega(x)$ with respect to x.

The distribution $\Omega(x)$ induces a distribution on $\mathbb{F}_2^k$ (and hence on its dual) in the following way:

For any vector $v \in \mathbb{F}_2^k$ the probability of v is $\Omega_w / \binom{k}{w}$, where w is the weight of v. A simple sampling algorithm for this distribution would be to sample first from the distribution $\Omega(x)$ to obtain a weight w, and then the sample a vector of weight w in $\mathbb{F}_2^k$ uniformly at random. By abuse of notation, we will in the following denote the distribution induced by $\Omega(x)$ on $\mathbb{F}_2^k$ by $\Omega(x)$ as well.

As an example we mention that the uniform distribution on $\mathbb{F}_2^k$ is given by the generating polynomial $\Omega(x) = \frac{1}{2^k}(1+x)^k$.

3 Fountain Codes and LT-Codes

The theoretical idea of Fountain Codes was introduced in [5] and the first practical realizations of Fountain Codes were invented by Luby [6, 7, 1]. They represent a new class of linear error-correcting codes. Let k be a positive integer, and let $\mathcal{D}$ be a degree distribution on $\mathbb{F}_2^k$. A Fountain Code with parameters $(k, \mathcal{D})$ has as its domain the space $\mathbb{F}_2^k$ of binary strings of length k, and as its target space the set of all sequences over $\mathbb{F}_2$, denoted by $\mathbb{F}_2^{\mathbb{N}}$. Formally, a Fountain Code with parameters $(k, \mathcal{D})$ is a linear map $\mathbb{F}_2^k \to \mathbb{F}_2^{\mathbb{N}}$ in which the coordinates are independent random variables with distribution $\mathcal{D}$ over $\mathbb{F}_2^k$. The block-length of a Fountain Code is potentially infinite, but in applications we will solely consider truncated Fountain Codes, i.e., Fountain Codes with finitely many coordinates, and make frequent and implicit use of the fact that unlike traditional codes the length of a Fountain Code is not fixed a-priori.

The symbols produced by a Fountain Code are called *output symbols*, and the k symbols from which these output symbols are calculated are called *input symbols*. The input and output symbols could be elements of $\mathbb{F}_2$, or more generally the elements of any finite dimensional vector space over $\mathbb{F}_2$ (or more generally, over any field $\mathbb{F}$).

Encoding of a Fountain Code is rather straightforward: for a given vector $(x_1, \ldots, x_k)$ of input symbols, each output symbol is generated independently and randomly by first sampling from the distribution $\mathcal{D}$ to obtain a weight w between 1 and k. Next, a vector $(v_1, \ldots, v_k)$ of weight w is chosen uniformly at random from $\mathbb{F}_2^k$ and the value of the output symbol is calculated as $\sum_i v_i x_i$. We will not be concerned with the cost of sampling from the distribution $\mathcal{D}$ over $\mathbb{F}_2^k$, as this will be trivial in our applications. The *encoding cost* of a Fountain Code is the expected number of operations sufficient to calculate an output symbol. This is easily seen to be at most $\omega - 1$, where ω is the expected Hamming weight of the random variable with distribution $\mathcal{D}$ over $\mathbb{F}_2^k$.

In addition to conceptual differences between Fountain Codes and block codes there is also an important operational difference between these classes of codes. For a traditional block code the structure of the code is determined prior to its use for transmission of information. This is also true for randomized block codes, such as random LDPC codes. On the other hand, in practice, Fountain Codes are generated "online." Each set of input symbols may have its own associated Fountain Code. There are various advantages to this mode of operation of Fountain Codes such as those described in Luby's paper [1], in Luby's patents on this subject [6, 7], or in [5].

In this paper we will consider Fountain Codes over a memoryless BEC with erasure probability p. Even though all our results also hold for more general and not necessarily memoryless erasure channels, we will only consider the memoryless case for sake of simplicity.

A *reliable decoding algorithm of length* n for a Fountain Code is an algorithm which can recover the k input symbols from any set of n output symbols and errs with a probability that is at most inversely polynomial in k (i.e., the error probability is at most $1/k^c$ for some positive constant c). Often, we will skip the term reliable and only talk about an algorithm of length n. The *cost* of such a decoding algorithm is the (expected) number of its arithmetic operations divided by k. This is equal to the average cost of recovering each input symbol.

When transmitting information using a traditional code, both the sender and the receiver are in possession of a description of the coding method used. For Fountain Codes this is not necessarily the case, since the code is being generated concurrently with the transmission. Therefore, in order to be able to recover the original information from the output symbols, it is necessary to transmit a description of the code together with the output symbols. In a setting where the symbols correspond to packets in a computer network, one can augment each transmission packet with a header information that describes the set of input symbols from which this output symbol was generated. We refer the reader to Luby [1, 6, 7] for a description of different methods for accomplishing this. In this paper, we will implicitly assume that the structure of the Fountain Code is transmitted together with the code using one of the many existing methods.

A special class of Fountain Codes is furnished by LT-Codes. In this class the distribution $\mathcal{D}$ has the form $\Omega(x)$ described in Section 2. It is relatively easy to prove an information theoretic lower bound on the encoding/decoding cost of any LT-Code which has a decoding algorithm of length approximately equal to k. We will prove the lower bound in terms of the number of edges in the *decoding graph*. The decoding graph of an algorithm of length n is a bipartite graph with k nodes

on the one side (called the input nodes or the input symbols) and n nodes on the other (called the output nodes or the output symbols). There is an edge between an input symbol and an output symbol if the input symbol contributes to the value of the output symbol.

The following proposition shows that the decoding graph of a reliable decoding algorithm has at least of the order of $k \log(k)$ edges. Therefore, if the number n of collected output symbols is close to k, then the encoding cost of the code is at least of the order of $\log(k)$.

Proposition 1. *If an LT-Code with k input symbols possesses a reliable decoding algorithm, then there is a constant c such that the graph associated to the decoder has at least $ck \log(k)$ edges.*

Proof. Suppose that the Fountain Code has parameters $(k, \Omega(x))$. In the decoding graph we call an input node covered if it is the neighbor of at least one output node. Otherwise, we call the node uncovered. The error probability of the decoder is lower bounded by the probability that there is an uncovered input node. We will establish a relationship between this probability and the average degree of an output node.

Let $\mathcal{G}$ denote the decoding graph of the algorithm. $\mathcal{G}$ is a random bipartite graph between k input and n output nodes such that each output node is of degree d with probability Ω_d, and such that the neighbors of an output node are randomly chosen. Let ν be an input node in $\mathcal{G}$. If an output node is of degree d, then the probability that ν is not a neighbor of that output node is $1 - d/k$. Since the output node is of degree d with probability Ω_d, the probability that ν is not a neighbor of an output node is $\sum_d \Omega_d \cdot (1 - d/k) = 1 - a/k$, where $a = \Omega'(1)$ is the average degree of an output node. Since output nodes are constructed independently, the probability that ν is not a neighbor of any of the output nodes is $(1 - a/k)^n$. We may assume that $a < k$. Then the Taylor expansion of $-\ln(1 - x)$ shows that $\ln(1 - a/k) \geq (a/k)/(1 - a/k)$ and hence $(1 - a/k)^n \geq \mathrm{e}^{-\alpha/(1-\alpha/n)}$, where $\alpha = an/k$ is the average degree of an input node. Since the decoder is assumed to be reliable, it errs with probability at most $1/k^u$ for some constant u. This shows that $\mathrm{e}^{-\alpha/(1-\alpha/n)} \leq 1/k^u$, i.e.,

$$\begin{aligned}
\alpha &\geq \ln(k) \frac{u}{1 + u \ln(k)/n} \\
&\geq \ln(k) \frac{u}{1 + u \ln(k)/k} \\
&\geq \ln(k) \frac{u}{1 + u \ln(3)/3} \\
&= \log(k) \frac{u}{\log(2)(1 + u \ln(3)/3)} \\
&=: c \log(k).
\end{aligned}$$

This completes the proof. □

In the following we will give some examples of Fountain Codes, and study different decoding algorithms. A *random* LT-Code is an LT-Code with parameters $(k, \Omega(x))$ where $\Omega(x) = \frac{1}{2^k}(1+x)^k$. As discussed earlier, this choice for $\Omega(x)$ amounts to the uniform distribution on $\mathbb{F}_2^k$, which explains the name.

Proposition 2. *A random LT-Code with k input symbols has encoding cost $k/2$, and ML decoding is a reliable decoding algorithm for this code of overhead $1 + O(\log(k)/k)$.*

Proof. Since the expected weight of a vector in $\mathbb{F}_2^k$ under uniform distribution is $k/2$, the encoding cost of the random LT-Code is $k/2$.

In the case of the erasure channel the ML decoding algorithm amounts to Gaussian elimination: we collect n output symbols (the value of n will be determined shortly). Each received output symbol represents a linear equation (with coefficients in $\mathbb{F}_2$) in the unknown input values $x_1, \ldots, x_k$, and thus the decoding process can be viewed as solving a (consistent) system of n linear equations in k unknowns. The decoding cost of this algorithm is $O(nk)$, since Gaussian elimination can be performed using $O(nk^2)$ operations.

It is well-known that a necessary and sufficient condition for the solvability of this system is that the rank of the corresponding matrix is equal to k. The entries of this matrix are independent binary random variables with equal probability of being one or zero. We will now prove that the probability that this matrix is not of full rank is at most 2^{k-n}. This is shown by using a union bound. For each hyperplane in $\mathbb{F}_2^k$ the probability that all the rows of the matrix belong to the hyperplane is 2^{-n}. There are $2^k - 1$ hyperplanes. Therefore, the probability that the matrix is not of full rank is at most $(2^k - 1)/2^n \leq 2^{k-n}$. Choosing $n = 1 + c\log(k)/k$, we see that the error probability of ML decoding becomes $1/k^c$, which proves the claim. □

Gaussian elimination is computationally expensive for dense codes like random LT-Codes. For properly designed LT-Codes, the Belief-Propagation (BP) decoder [3, 1] provides a much more efficient decoder. The BP decoder can be best described in terms of the graph associated to the decoder. It performs the following steps until either no output symbols of degree one are present in the graph, or until all the input symbols have been recovered. At each step of the algorithm the decoder identifies an output symbol of degree one. If none exists, and not all the input symbols have

been recovered, the algorithm reports a decoding failure. Otherwise, the value of the output symbol of degree one recovers the value of its unique neighbor among the input symbols. Once this input symbol value is recovered, its value is added to the values of all the neighboring output symbols, and the input symbols and all edges emanating from it are removed from the graph.

For random LT-Codes the BP decoder fails miserably even when the number of collected output symbols is very large. Thus the design of the degree distribution $\Omega(x)$ must be dramatically different from the random distribution to guarantee the success of the BP decoder.

The analysis of the BP decoding algorithm is more complicated than the analysis of ML decoding. For the sake of completeness, we include a short expectation analysis for the case where every output symbol chooses its neighbors among the input symbols randomly and with replacement. We refer the reader to [1] for the analysis of the original case where the choice is done without replacement.

As described above, the BP decoder proceeds in steps, and recovers one input symbol at each step. Following Luby's notation, we call the set of output symbols of reduced degree one the *output ripple* at step i of the algorithm. We say that an output symbol is *released* at step $i+1$ if its degree is larger than 1 at step i, and it is equal to one at step $i+1$, so that recovery of the input symbol at step $i+1$ reduces the degree of the output symbol to one. The probability that an output symbol of initial degree d releases at step $i+1$ can be easily calculated as follows: This is the probability that the output symbol has exactly one neighbor among the $k-i-1$ input symbols that are not yet recovered, and that not all the remaining $d-1$ neighbors are among the i already recovered input symbols. The probability that the output symbols has exactly one neighbor among the unrecovered input symbols, and that all its other neighbors are within a set of size s contained in the set of remaining input symbols is $d(1-\frac{i+1}{k})\left(\frac{s}{k}\right)^{d-1}$. Therefore,

$$\Pr[\text{output symbol is released at step } i+1 \mid \text{degree is } d] = d\left(1-\frac{i+1}{k}\right)\left(\left(\frac{i+1}{k}\right)^{d-1} - \left(\frac{i}{k}\right)^{d-1}\right).$$

Multiplying the term with the probability Ω_d that the degree of the symbol is d, and summing over all d we obtain

$$\Pr[\text{output symbol is released at step } i+1] = \left(1-\frac{i+1}{k}\right)\left(\Omega'\left(\frac{i+1}{k}\right) - \Omega'\left(\frac{i}{k}\right)\right).$$

Note that

$$\Omega'\left(\frac{i+1}{k}\right) - \Omega'\left(\frac{i}{k}\right) \sim \frac{1}{k}\Omega''\left(\frac{i}{k}\right).$$

The approximation is very good if $0 \le i/k \le 1-\gamma$ for constant γ and large k.

Suppose that the decoder collects n output symbols. Then the expected number of output symbols releasing at step $i+1$ is n times the probability that an output symbol releases at step $i+1$, which, by the above, is approximately equal to

$$\frac{n}{k}\left(1-\frac{i+1}{k}\right)\Omega''\left(\frac{i}{k}\right).$$

In order to construct asymptotically optimal codes, i.e., codes that can recover the k input symbols from any n output symbols for values of n arbitrarily close to k, we require that every decoded input symbol releases exactly one output symbol. Thus, in the limit, we require $n = k$, and we require that the output ripple has expected size one at every step. This means that $(1-x)\Omega''(x) = 1$ for $0 < x < 1$. Solving this differential equation, and keeping in mind that $\Omega(1) = 1$, we obtain the soliton distribution: $\Omega(x) = \sum_{i\geq 2}\frac{x^i}{i(i-1)}$. The distribution is similar to the ideal soliton distribution of Luby [1], except that it assigns a probability of zero to degree one, and has infinitely many terms.

The distribution of the size of the output ripple at each point in time is more difficult to calculate and we refer the reader to the upcoming paper [10] for details.

The reader is referred to Luby's paper for a description of LT-Codes with a distribution $\Omega(x)$ with $\Omega'(1) = O(\log(k))$ and for which the BP decoder is a reliable decoder of overhead $k(1 + O(\log^2(k)/\sqrt{k}))$. These degree distributions are absolutely remarkable, since they lead to an extremely simple decoding algorithm that essentially matches the information theoretic lower bound in Proposition 1.

4 Raptor Codes

The results of the previous section imply that LT-Codes cannot be encoded with constant cost if the number of collected output symbols is close to the number of input symbols. In this section we will present a different class of Fountain Codes. One of the many advantages of the new construction is that it allows for encoding and decoding with constant cost, as we will see below.

The reason behind the lower bound of $\log(k)$ for the cost of LT-Codes is the information theoretic lower bound of Proposition 1. The decoding graph needs to have of the order of $k\log(k)$ edges in order to make sure that all the input nodes are covered with high probability. The idea of Raptor Coding is to relax this condition and require that only a *constant fraction* of the input symbols be recoverable. Then the same information theoretic argument as before shows only a linear lower bound for the number of edges in the decoding graph.

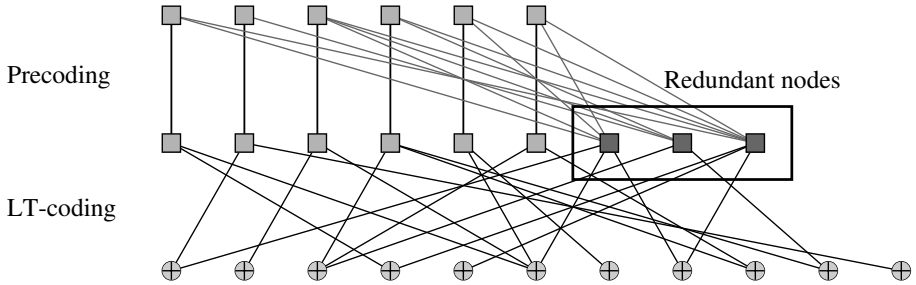


Figure 1: Raptor Codes: the input symbols are appended by redundant symbols (black squares) in the case of a systematic pre-code. An appropriate LT-code is used to generate output symbols from the pre-coded input symbols

There are two potential problems with this approach: (1) The information theoretic lower bound may not be matchable with an algorithm, and (2) We need to recover all the input symbols, not only a constant fraction.

The second issue is addressed easily: we encode the input symbols using a traditional erasure correcting code, and then apply an appropriate LT-Code to the new set of symbols, in a way that the traditional code is capable of recovering all the input symbols even in face of a fixed fraction of erasures. To deal with the first issue, we need to design the traditional code and the LT-Code appropriately.

Let $\mathcal{C}$ be a linear code of block-length n and dimension k, and let $\Omega(x)$ be a degree distribution. A *Raptor Code* with parameters $(k, \mathcal{C}, \Omega(x))$ is an LT-Code with distribution $\Omega(x)$ on n symbols which are the coordinates of codewords in $\mathcal{C}$. The code $\mathcal{C}$ is called the *pre-code* of the Raptor Code. The input symbols of a Raptor Code are the k symbols used to construct the codeword in $\mathcal{C}$ consisting of n *intermediate symbols*. The output symbols are the symbols generated by the LT-Code from the n intermediate symbols. A graphical presentation of a Raptor Code is given in Figure 1. Typically, we assume that $\mathcal{C}$ is equipped with a systematic encoding, though this is not necessary.

A moment's thought reveals that Raptor Codes form a subclass of Fountain Codes: The output distribution $\Omega(x)$ and a fixed pre-code $\mathcal{C}$ induce a distribution $\mathcal{D}$ on $\mathbb{F}_2^k$, where k is the number of input symbols of the Raptor Code. The output symbols of the Raptor Code are sampled independently from the distribution $\mathcal{D}$.

A Raptor Code has an obvious encoding algorithm as follows: given k input symbols, an encoding

algorithm for $\mathcal{C}$ is used to generate a codeword in $\mathcal{C}$ corresponding to the input symbols. Then an encoding algorithm for the LT-Code with distribution $\Omega(x)$ is used to generate the output symbols.

A *reliable decoding algorithm of length m* for a Raptor code is an algorithm which can recover the k input symbols from any set of m output symbols and errs with probability which is at most $1/k^c$ for some positive constant c. As with LT-Codes, we sometimes omit mentioning the attribute "reliable."

The definition of the *encoding cost* of a Raptor Code differs slightly from that of a Fountain Code. This is because the encoding cost of the pre-code has to be taken into account. We define the encoding cost of a Raptor Code as $\mathrm{E}(\mathcal{C})/k+\Omega'(1)$, where $\mathrm{E}(\mathcal{C})$ is the number of arithmetic operations sufficient for generating a codeword in $\mathcal{C}$ from the k input symbols. The encoding cost equals the per-symbol cost of generating k output symbols.

The *decoding cost* of a decoding algorithm for a Raptor Code is the expected number of arithmetic operations sufficient to recover the k input symbols, divided by k. As with the Fountain Codes, this cost counts the expected number of arithmetic operations per input symbol.

We will study Raptor Codes with respect to the following performance parameters:

1. *Space:* Since Raptor Codes require storage for the intermediate symbols, it is important to study their space consumption. We will count the space as a multiple of the number of input symbols. The space requirement of the Raptor Code is $1/R$, where R is the rate of the pre-code.

2. *Overhead:* The overhead is a function of the decoding algorithm used, and is defined as the number of output symbols that the decoder needs to collect in order to recover the input symbols with high probability. We will measure the overhead as a multiple of the number k of input symbols, so an overhead of $1+\varepsilon$, for example, means that $(1+\varepsilon)k$ output symbols need to be collected to ensure successful decoding with high probability.

3. *Cost:* The cost of the encoding and the decoding process.

In the next section we will give several examples of Raptor Codes and study their performance.

5 First Examples of Raptor Codes

The first example of a Raptor Code is an LT-code. An LT-code with k input symbols and output distribution $\Omega(x)$ is a Raptor Code with parameters $(k, \mathbb{F}_2^k, \Omega(x))$. ($\mathbb{F}_2^k$ is the trivial code of dimension

and block-length k.) LT-codes have optimal space consumption (i.e., 1). With an appropriate output distribution $\Omega(x)$ the overhead of an LT-code is $1 + O(\log^2(k)/\sqrt{k})$, and its cost is proportional to $\log(k)$, as was seen in Section 3.

LT-codes have no pre-coding, and compensate for the lack of it by using a very intricate output distribution $\Omega(x)$. At the other end of the spectrum are Raptor Codes that have the simplest possible output distribution, with a sophisticated pre-code, which we call *pre-code-only* (PCO) Raptor Codes. Let $\mathcal{C}$ be a code of dimension k and block-length n. A Raptor Code with parameters $(k, \mathcal{C}, x)$ is called a PCO Raptor Code with pre-code $\mathcal{C}$. In this code the k input symbols are encoded via $\mathcal{C}$ to produce the n intermediate symbols and the output distribution is fixed to the trivial distribution $\Omega(x) = x$. The value of every output symbol equals that of an input symbol chosen uniformly at random.

The decoding algorithm for a PCO Raptor Code is the trivial one: a predetermined number m of output symbols are collected. These will determine the values of, say, ℓ intermediate symbols. Next the decoding algorithm for the pre-code is applied to these recovered intermediate values to obtain the values of the input symbols.

The performance of a PCO Raptor Code depends on the performance of its pre-code $\mathcal{C}$, as the following result suggests.

Proposition 3. *Let $\mathcal{C}$ be a linear code of dimension k and block-length n with encoding and decoding algorithms that have the following properties:*

1. *An arbitrary input vector of length k can be encoded with $k \cdot \eta$ arithmetic operations for some $\eta > 0$.*

2. *There is an $\epsilon > 0$ such that the decoding algorithm can decode $\mathcal{C}$ over a binary erasure channel with erasure probability $1 - R(1 + \varepsilon)$ with high probability using $k \cdot \gamma$ arithmetic operations for some $\gamma > 0$.*

Then the PCO Raptor Code with pre-code $\mathcal{C}$ has space consumption $1/R$, overhead $-\ln(1 - R(1 + \epsilon))/R$, encoding cost η, and decoding cost γ with respect to the decoding algorithm for $\mathcal{C}$, where $R = k/n$ is the rate of $\mathcal{C}$.

Proof. The space consumption and the costs of encoding/decoding are clear. As for the overhead, suppose that the decoder collects $m = -k\ln(1-R(1+\varepsilon))/R = -n\ln(1-R(1+\varepsilon))$ output symbols. We need to show that the probability that an intermediate symbol is not covered is at most $1-R(1+\varepsilon)$,

since if this condition is satisfied, then the decoder for the pre-code can decode the input symbols. To show the latter, note that the probability that an intermediate symbol is not covered is $(1-1/n)^m$ which is upper bounded by $\mathrm{e}^{-m/n} = 1 - R(1+\varepsilon)$. □

Note that the overhead of the PCO Raptor Code in the previous proposition is at least $1+\epsilon$, since $-\ln(1-R(1+\epsilon))/R \geq 1+\epsilon$ for $0 < R < 1/(1+\epsilon)$. Moreover, the overhead approaches this upper bound only if R approaches zero. Therefore, to obtain PCO Raptor Codes with close to optimal overhead the rate of the pre-code needs to approach zero, which means that the running time of the code cannot be a constant. The same is true for the space consumption of the PCO Raptor Code.

Despite these obvious shortcomings PCO Raptor Codes are quite appealing, since this transforms any block code into a Fountain Code. For example, PCO Raptor Codes could be useful when the intermediate symbols (codeword in $\mathcal{C}$) can be calculated offline via pre-processing, and the space needed to keep these symbols is of no concern. In such a scenario a PCO Raptor Codes is the fastest possible Fountain Code.

The choice of the code $\mathcal{C}$ depends on the specific application in mind, though usually it is best to choose a code with very good encoding and decoding algorithms and little overhead for a given rate. One possible choice would be a Tornado code [3], though other choices are also possible (for example an LT-code with the appropriate number of output symbols, or an irregular Repeat-Accumulate Code [11]).

6 Raptor Codes with Good Asymptotic Performance

In the last section we encountered two types of Raptor Codes. For one of them, the LT-codes, the overhead and the space were close to 1, while the decoding cost grew with k. For PCO Raptor Codes the decoding cost could be chosen to be a constant, but then the overhead and the space were away from 1; moreover, convergence to an overhead equal to 1 amounted to letting the space and the cost to grow with k.

In this section we will design Raptor Codes between these two extremes. These codes have encoding and decoding algorithms of constant cost, and their space consumption and the overhead are arbitrarily close to 1. We will design these codes by choosing an appropriate output distribution $\Omega(x)$ and an appropriate pre-code $\mathcal{C}$.

The output degree distribution we will use is very similar to the soliton distribution in Section 3.

However, this distribution needs to be slightly modified. First, the soliton distribution does not have output nodes of degree one. This means that it is not possible to start the decoding process with this distribution. Second, the soliton distribution has infinitely many terms. Our distribution will modify the soliton distribution by capping it at some maximum degree D, and giving it an appropriate weight for output symbols of degree one.

Let ε be a real number larger than zero, and set $D := \lceil 4(1+\varepsilon)/\varepsilon \rceil$ and define

$$\Omega_D(x) = \frac{1}{\mu+1}\left(\mu x + \frac{x^2}{1\cdot 2} + \frac{x^3}{2\cdot 3} + \cdots + \frac{x^D}{(D-1)\cdot D} + \frac{x^{D+1}}{D}\right),$$

where $\mu = (\varepsilon/2) + (\varepsilon/2)^2$. Then we have the following result.

Lemma 4. *There exists a positive real number c (depending on ε) such that with an error probability of at most e^{-cn} any set of $(1+\varepsilon/2)n+1$ output symbols of the LT-code with parameters $(n, \Omega_D(x))$ are sufficient to recover at least $(1-\delta)n$ input symbols via BP decoding, where $\delta = (\varepsilon/4)/(1+\varepsilon)$.*

Proof. We use the analysis of the decoding process as described in [3] or in [12]. Consider a set of $n(1+\varepsilon/2)+1$ output symbols and set up the graph associated to these output symbols. This graph is a random graph with edge degree distributions $\iota(x)$ and $\omega(x)$ corresponding to the input and the output symbols, respectively. According to the analysis in [3], for any constant δ, if $\iota(1-\omega(1-x)) < x$ for $x \in [\delta, 1]$, then the probability that the decoder cannot recover δn or more of the input nodes is upper bounded by e^{-cn}, where c is a suitable constant (depending on δ, and $\Omega(x)$, but not on n).

In the case of an LT-code with parameters $(n, \Omega(x))$ we have $\omega(x) = \Omega'(x)/\Omega'(1)$. To compute $\iota(x)$ fix an input node. The probability that this node is the neighbor of a given output node is a/n, where a is the average degree of an output node, i.e., $a = \Omega'(1)$. The probability that the input node is the neighbor of exactly ℓ output nodes is therefore $\binom{N}{\ell}(a/n)^\ell(1-a/n)^{N-\ell}$, where $N = n(1+\varepsilon/2)+1$ is the number of output symbols in the graph. Hence, the generating function of the degree distribution of the input nodes equals

$$\sum_\ell \binom{N}{\ell}\left(\frac{a}{n}\right)^\ell x^\ell \left(1-\frac{a}{n}\right)^{N-\ell} = \left(1-\frac{a(1-x)}{n}\right)^N = \left(1-\frac{a(1-x)}{n}\right)^{(1+\varepsilon/2)\cdot n+1}.$$

The edge degree distribution $\iota(x)$ of the input nodes is the derivative of this polynomial with respect to x, normalized so that $\iota(1) = 1$. This shows that

$$\iota(x) = \left(1-\frac{a(1-x)}{n}\right)^{(1+\varepsilon/2)n}.$$

Since $(1-b/m)^m < \mathrm{e}^{-b}$ for $b \le m$, this implies

$$\iota(1-\omega(1-x)) < \mathrm{e}^{-(1+\varepsilon/2)\Omega'(1-x)}.$$

So, we only need to show that the right-hand side of this inequality is less than x on $[\delta,1]$, or, equivalently, that $\mathrm{e}^{-(1+\varepsilon/2)\Omega'(x)} < 1-x$ for $x \in [0,1-\delta]$. Note that

$$\begin{aligned}\Omega'(x) &= \frac{1}{\mu+1}\left(\mu + x + \frac{x^2}{2} + \cdots + \frac{x^{D-1}}{D-1} + \frac{(D+1)\cdot x^D}{D}\right)\\ &= \frac{1}{\mu+1}\left(\mu - \ln(1-x) + x^D - \sum_{d=D+1}^{\infty}\frac{x^d}{d}\right).\end{aligned}$$

We will show that $x^D > \sum_{d=D+1}^{\infty} x^d/d$ for $x \in [0,1-\delta]$, which proves the inequality $\Omega'(x) > (\mu-\ln(1-x))/(\mu+1)$. To see that $\sum_{d=D+1}^{\infty} x^{d-D}/d < 1$, note that the left hand side is monotonically increasing, so we only need to prove this inequality for $x = 1-\delta$. For the choice $\delta = (\varepsilon/4)/(1+\varepsilon)$ in the statement of the lemma we have

$$\begin{aligned}\sum_{d=D+1}^{\infty}\frac{(1-\delta)^{d-D}}{d} &< \frac{1}{D+1}\sum_{d=1}^{\infty}(1-\delta)^d\\ &= \frac{1-\delta}{(D+1)\cdot\delta}\\ &\le \frac{4+3\varepsilon}{\varepsilon}\cdot\frac{\varepsilon}{4+5\varepsilon}\\ &< 1.\end{aligned}$$

So far, we have shown that $\Omega'(x) > (\mu-\ln(1-x))/(\mu+1)$. So, $\mathrm{e}^{-(1+\varepsilon/2)\Omega'(x)} < \mathrm{e}^{-(1+\varepsilon/2)\mu/(1+\mu)}(1-x)^{(1+\varepsilon/2)/(1+\mu)}$. To complete the proof we need to show that

$$\mathrm{e}^{-(1+\varepsilon/2)\mu/(1+\mu)} < (1-x)^{1-(1+\varepsilon/2)/(1+\mu)}$$

for $x \in [0,1-\delta]$. Note that $\mu > \varepsilon/2$. Therefore, the above inequality is valid on the entire interval $[0,1-\delta]$ iff it is valid at $x = 1-\delta$, i.e., iff

$$\frac{-(1+\varepsilon/2)\mu}{1+\mu} < \frac{\mu-\varepsilon/2}{1+\mu}\cdot\ln(\delta).$$

Plugging in the value of μ, it remains to show that $-(1+\varepsilon/2)^2 < (\varepsilon/2)\ln((\varepsilon/4)/(1+\varepsilon))$, which is verified easily. □

Note that the choices in the previous theorem are far from optimal, but they suffice to prove the asymptotic result.

To construct asymptotically good Raptor Codes, we will use LT-Codes described in the previous lemma, and suitable pre-codes. In the following we will assume that ε is a fixed positive real number, and we assume that for every n we have a linear code $\mathcal{C}_n$ of block-length n with the following properties:

1. The rate R of $\mathcal{C}_n$ is $(1+\varepsilon/2)/(1+\varepsilon)$,
2. The BP decoder can decode $\mathcal{C}_n$ on a BEC with erasure probability $\delta = (\varepsilon/4)/(1+\varepsilon) = (1-R)/2$ with $O(n\log(1/\varepsilon))$ arithmetic operations.

Examples of such codes are Tornado codes [3], right-regular codes [13], and certain types of repeat-accumulate codes. The reader can consult [14] for other types of such capacity-achieving examples. We remark, however, that it is not necessary for $\mathcal{C}_n$ to be capacity-achieving, since we only require that the decoder be able to decode up to $(1-R)/2$ fraction of errors rather than $1-R$. For example, we mention without proof that the right-regular LDPC-code with message edge degree distribution $(2x+3x^2)/5$ can be used as the pre-code $\mathcal{C}_n$.

Theorem 5. *Let ε be a positive real number, k be an integer, $D = \lceil 4(1+\varepsilon)/\varepsilon \rceil$, $R = (1+\varepsilon/2)/(1+\varepsilon)$, $n = \lceil k/(1-R) \rceil$, and let $\mathcal{C}_n$ be a code with the properties described above. Then the Raptor code with parameters $(k, \mathcal{C}_n, \Omega_D(x))$ has space consumption $1/R$, overhead $1+\varepsilon$, and a cost of $O(\log(1/\varepsilon))$ with respect to BP decoding of both the pre-code and the LT-Code.*

Proof. Given $k(1+\varepsilon)$ output symbols, we use the LT-code with parameters $(n, \Omega_D(x))$ to recover at least a $(1-\delta)$-fraction of the input symbols, where $\delta = (\varepsilon/4)/(1+\varepsilon)$. Lemma 4 guarantees that this is possible. Next we use the BP decoder for $\mathcal{C}_n$ to recover the k input symbols in linear time.

It remains to show the assertion on the cost. The average degree of the distribution Ω_D is $\Omega'_D(1) = 1 + H(D)/(1+\mu) = \ln(1/\varepsilon) + \alpha + O(\varepsilon)$, where $H(D)$ is the harmonic sum up to D. (One can show that $1 < \alpha < 1+\gamma+\ln(9)$, where γ is Euler's constant.) The number of operations necessary for generating the redundant symbols of $\mathcal{C}_n$ is proportional to $k\log(1/\varepsilon)/R$ which is proportional to $k\log(1/\varepsilon)$. The same is true for the decoding cost of $\mathcal{C}_n$. This proves the assertion on the cost. □

A careful look at the decoder described above shows that its error probability is only polynomially small in k, rather than exponentially small (in other words, its error exponent is zero). The reason for this is that the error probability of the decoder for $\mathcal{C}_n$ has this property. So, if a different linear

time decoder for $\mathcal{C}_n$ exhibits a subexponentially small error probability, then the same will also be true for the error probability of the Raptor Code which uses $\mathcal{C}_n$ as its pre-code.

We also remark that the construction in the previous theorem is essentially optimal. Using the same techniques as in Proposition 1 it can be shown that the parameters of a Raptor Code with a reliable decoding algorithm of length N and a pre-code of rate $R < 1$ satisfy the inequality

$$\Omega'(1) \geq -c\frac{\ln(1-R)}{R} \cdot \frac{k}{N},$$

for some constant c, where $\Omega(x)$ is the output degree distribution of the corresponding LT-Code. In our construction, we have $\Omega'(1) = \ln(1/\varepsilon) + \alpha + O(\varepsilon)$ for some constant α, k/N is $O(\varepsilon)$, $-\ln(1-R) = \ln(1/\varepsilon) + \beta + O(\varepsilon)$ for some constant β, and $R = 1 - O(\varepsilon)$. (One can show that $\beta = \ln(2)$.) Therefore, the upper and the lower bounds on $\Omega'(1)$ have the same order of magnitude for small ε. In this respect, the codes constructed here are essentially optimal.

7 Finite Length Analysis of Raptor Codes

The analysis in the previous section is satisfactory from an asymptotic but not from a practical point of view. The analysis of the decoding process of the corresponding LT-Codes relies on martingale arguments to enable upper bounds on the error probability of the decoder. The same is true for the pre-code. Such bounds are very far from tight, and are especially bad when the number of input symbols is small.

In this section we will introduce a different type of error analysis for Raptor Codes of finite length with BP decoding. This analysis relies on the exact calculation of the error probability of the LT-decoder, derived in [10], combined with the calculation of the error probability for certain LDPC codes [15].

7.1 Design of the Output Degree Distribution

Following [1], we call an input symbol released at time T if at least one neighbor of that input symbol becomes of reduced degree one after T input symbols are recovered. The input ripple at time T is defined as the set of all input symbols that are released at time T. The aim of the design is to keep the input ripple large during as large a fraction of the decoding process as possible.

We will give a heuristic analysis of the expected size of the input ripple given that the decoding process has already recovered a fixed fraction of the input symbols. For this, it is advantageous to

rephrase the BP decoding. At every round of this algorithm messages are sent along the edges from output symbols to input symbols, and then from input symbols to output symbols. The messages sent are 0 or 1. An input symbol sends a 0 to an incident output symbol iff its value is not recovered yet. Similarly, an output symbol sends a message 0 to an incident input symbol iff the output symbol is not able to recover the value of the input symbol.

Let p_i be the probability that an edge in the decoding graph carries a value 1 from an output symbol at step i of the decoding process. Then, a standard tree analysis argument [16] shows the recursion

$$p_{i+1} = \omega(1 - \iota(1 - p_i)),$$

where ω and ι are the output and the input edge degree distributions, respectively. Note that this recursion is only valid if we can assume that the messages along the edges are statistically independent.

We have $\omega(x) = \Omega'(x)/\Omega'(1)$, and $\iota(x) = \mathrm{e}^{\alpha(x-1)}$, where α is the average degree of an input symbol, and $\Omega'(x)$ is the derivative of $\Omega(x)$. (The latter is a standard approximation of the binomial distribution by a Poisson distribution, see the proof of Lemma 4.) Moreover, the input node degree distribution also equals $\mathrm{e}^{\alpha(x-1)}$, since this distribution is equal to $\iota'(x)/\iota'(1)$.

Let u_i denote the probability that an input symbol is recovered at round i. An input symbol is recovered iff it is incident to an edge which carries the message 1 from some output symbol. The probability that an input symbol is recovered, conditioned on its degree being d, equals $1-(1-p_i)^d$. Hence, the probability that an input symbol is unrecovered at round i of the algorithm is $1-\iota(1-p_i) = 1 - \mathrm{e}^{-\alpha p_i}$. This shows that $p_i = -\ln(1-u_i)/\alpha$. Phrasing the above recursion for the p_i's in terms of the u_i's, we obtain

$$u_{i+1} = 1 - \mathrm{e}^{-\alpha\omega(u_i)}.$$

This recursion shows that if an expected x-fraction of input symbols has been already recovered at some step of the algorithm, then in the next step that fraction increases to $1 - \mathrm{e}^{-\alpha\omega(x)}$. Therefore, the expected fraction of input symbols in the input ripple will be $1 - x - \mathrm{e}^{-\alpha\omega(x)}$.

Suppose that the decoding algorithm runs on $k(1+\varepsilon)$ output symbols. Then $\alpha\omega(x) = (1+\varepsilon)\Omega'(x)$, and we see that the expected fraction of symbols in the input ripple is

$$1 - x - \mathrm{e}^{-\Omega'(x)(1+\varepsilon)}.$$

The above derivation is a heuristic. But for two reasons this does not matter for the design of the Raptor codes:

1. The heuristic is only a means for obtaining good degree distribution candidates. Once we have found a candidate, we will exactly calculate the error probability of the LT-decoder on that candidate as discussed in Section 7.2.

2. It can be shown by other means that the above formula is, in fact, the exact expectation of the size of the input ripple [10].

Let us assume that the pre-code of the Raptor Code to be designed has block-length n. We need to design the output degree distribution in such a way as to ensure that a large fraction of the k input symbols are recovered. To solve this design problem, we use an idea communicated to us by Luby [17]: We try to keep the expected ripple size larger than or equal to $c\sqrt{(1-x)k}$, for some positive constant c. The rationale behind this choice is that if deletion and insertion of elements into the input ripple were to happen independently with probability $1/2$ every time an input symbol is recovered, then the input ripple size needed to be larger by a factor of c than the square root of the number of input symbols yet to be recovered, which is $\sqrt{(1-x)k}$. Though only a heuristic, this condition turns out to be very useful for the design of the degree distributions.

Using this condition, the design problem becomes the following: given ε and δ, and given the number k of input symbols, find a degree distribution $\Omega(x)$ such that

$$1 - x - \mathrm{e}^{-\Omega'(x)(1+\varepsilon)} \geq c\sqrt{\frac{1-x}{k}}$$

for $x \in [0, 1-\delta]$. Indeed, if this condition is satisfied, then the expected size of the input ripple which is $k(1 - x - \mathrm{e}^{-\Omega'(x)(1+\varepsilon)})$ is larger than or equal to $c\sqrt{(1-x)k}$.

This design problem can be solved by means of linear programming, in a manner described in [3]. Namely, the inequality can be manipulated to yield

$$\Omega'(x) \geq \frac{-\ln\left(1 - x - c\sqrt{\frac{1-x}{k}}\right)}{1+\varepsilon}$$

for $x \in [0, 1-\delta]$. Note that for this to be solvable, δ needs to be larger than $c/\sqrt{k}$. By discretizing the interval $[0, 1-\delta]$ and requiring the above inequality to hold on the discretization points, we obtain linear inequalities in the unknown coefficients of $\Omega(x)$. Moreover, we can choose to minimize

k	65536	80000	100000	120000
Ω_1	0.007969	0.007544	0.006495	0.004807
Ω_2	0.493570	0.493610	0.495044	0.496472
Ω_3	0.166220	0.166458	0.168010	0.166912
Ω_4	0.072646	0.071243	0.067900	0.073374
Ω_5	0.082558	0.084913	0.089209	0.082206
Ω_8	0.056058	0.049633	0.041731	0.057471
Ω_9	0.037229	0.043365	0.050162	0.035951
Ω_{18}				0.001167
Ω_{19}	0.055590	0.045231	0.038837	0.054305
Ω_{20}		0.010157	0.015537	
Ω_{65}	0.025023			0.018235
Ω_{66}	0.003135	0.010479	0.016298	0.009100
Ω_{67}		0.017365	0.010777	
ε	0.038	0.035	0.028	0.02
a	5.87	5.91	5.85	5.83

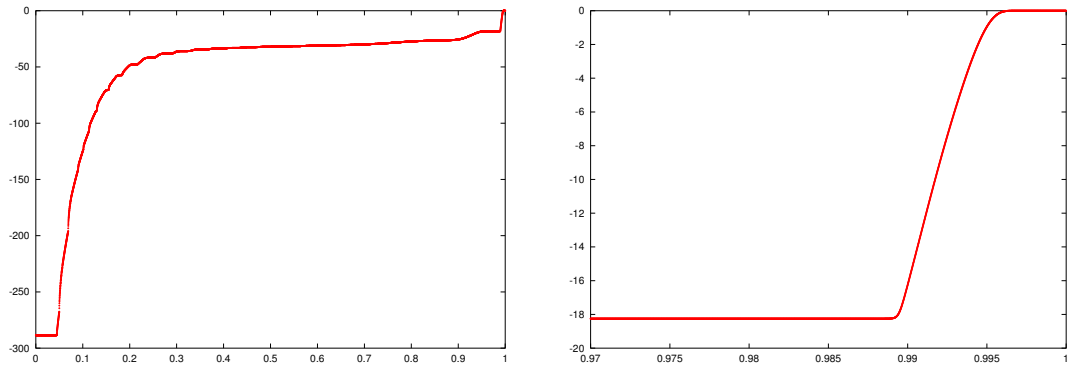
Table 1: Degree distributions for various values of k; $1+\varepsilon$ is the overhead, and a is the average degree of an output symbol

the objective function $\Omega'(1)$ (which is again linear in the unknown coefficients of $\Omega(x)$), in order to obtain a degree distribution with the minimum possible average degree.

Table 1 shows several optimized degree distributions we have found using this method for various values of k. All the δ-values used are equal to 0.01. It is interesting to note that for small values of $d > 1$, Ω_d is approximately equal to $1/(d(d-1))$, which is the same as for the soliton distribution given in Section 3.

7.2 Error Analysis of LT-codes

The upcoming paper [10] describes a dynamic programming approach to calculate the error probability of the LT-decoder for a given degree distribution. More precisely, given k and the degree



(a) (b)

Figure 2: Decimal logarithm of the cumulative probability of error of the LT-decoder (vertical axis) versus the fraction of decoded input symbols (horizontal axis) for the sequence given in Table 1 for $k = 100000$. (a) full range, and (b) towards the end of the decoding process (less than 3% input symbols left to decode).

distribution $\Omega(x)$, the procedure computes for every ℓ the probability $P_\ell = P_\ell^{k,\Omega(x)}$ that the decoding process fails with exactly ℓ input symbols recovered.

Figure 2 shows a plot of the cumulative probability of decoding error (vertical axis in log-scale) versus ℓ/k (horizontal axis), for the sequence in Table 1 corresponding to the value $k = 100,000$. Note that for all the degree distributions given in Table 1 and all large enough number of input symbols the error probability of the LT-decoder jumps to 1 before all input symbols are recovered. This is because the average degree of the output symbols in the LT-decoder is too small to guarantee coverage of all input symbols.

7.3 Design and Error Analysis of the Pre-Code

Even though the choice of a Tornado code or a right-regular code as the pre-code of a Raptor Code is sufficient for proving theoretical results about the linear time encodability and decodability of Raptor codes with suitable distributions, such choices turn out to be rather poor in practical situations. In such cases, one is interested in a robust pre-code with provable guarantees on the decoding capability, even for short lengths.

In this section we discuss a special class of LDPC codes that are well suited as a pre-code. First, let us recall the definition of an LDPC code. Let $\mathcal{G}$ be a bipartite graph with n left and r right nodes.

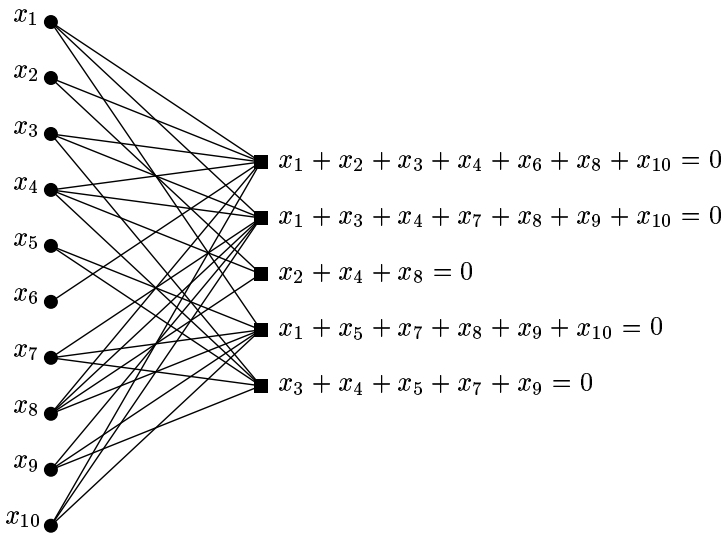


Figure 3: An LDPC code

In the context of LDPC codes the left nodes are often referred to as the message or variable nodes while the right nodes are referred to as the check nodes. The linear code associated with the graph is of block-length n. The coordinate positions of a codeword are identified with the n message nodes. The codewords are those vectors of length n over the base field such that for every check node the sum of its neighbors among the message nodes is zero. (See Figure 3.)

BP decoding of LDPC codes over an erasure channel is very similar to the BP decoding of LT-Codes [3]. It has been shown in [18] that this decoding algorithm is successful if and only if the graph induced by the erased message positions does not contain a *stopping set*. A stopping set is a set of message nodes such that their induced graph has the property that all the check nodes have degree greater than one. For example, in Figure 3 the message nodes $1, 2, 4, 5$ generate a stopping set of size 4.

Since the union of two stopping sets is again a stopping set, a bipartite graph contains a unique maximal stopping set (which may be the empty set). The analysis of erasure decoding for LDPC codes boils down to computing for each value of s the probability that the graph generated by the erased positions has a maximal stopping set of size s.

The LDPC codes we will study in this section are constructed from a node degree distribution $\Lambda(x) = \sum_d \Lambda_d x^d$. For each of the n message nodes the neighboring check nodes are constructed as follows: a degree d is chosen independently at random from the distribution given by $\Lambda(x)$. Then d

random check nodes are chosen which constitute the neighbors of the message node. The ensemble of graphs defined this way will be denoted by $\mathbf{P}(\Lambda(x), n, r)$ in the following.

For a graph in the ensemble $\mathbf{P}(\Lambda(x), n, r)$ we can calculate an upper bound for the probability that the graph has a maximal stopping set of size s. The following theorem has been proved in [15].

Theorem 6. *Let r be a positive integer. For $n \geq 1$, $z, o \in \mathbb{Z}$, and $d \geq 1$ let $A_n(z, o)$ be recursively defined by*

$$\begin{aligned} A_0(r, 0) &:= 1, \\ A_0(z, o) &:= 0 \text{ for } (z, o) \neq (r, 0), \\ A_{n+1}(z, o) &:= \sum_{\ell,k} A_n(\ell, k) \sum_d \Lambda_d \frac{\binom{\ell}{\ell-z}\binom{k}{k+\ell-z-o}\binom{r-\ell-k}{d-k-2\ell+2z+o}}{\binom{r}{d}} \text{ for } n \geq 0. \end{aligned}$$

Let $\mathcal{G}$ be a random graph in the ensemble $\mathbf{P}(\Lambda(x), n, r)$. Then the probability that $\mathcal{G}$ has a maximal stopping set of size s is at most

$$\binom{n}{s} \sum_{z=0}^{r} A_s(z, 0) \left(1 - \sum_d \Lambda_d \frac{\binom{r-z}{d}}{\binom{r}{d}}\right)^{n-s}.$$

In the above theorem $A_n(z, o)$ is the probability that a random bipartite graph in the ensemble ($\mathbf{P}(\Lambda(x), n, r)$ has z check nodes of degree 0 and o check nodes of degree one. This implies the second statement of the theorem.

A standard dynamic programming algorithm can compute the upper bound in the above theorem with $O(D^3 n^2 r^4 \log(r))$ bit operations, where D is the maximum degree in the distribution $\Lambda(x)$.

7.4 Combined Error Probability

The decoding error probability of a Raptor Code with parameters $(k, \mathcal{C}, \Omega(x))$ can be estimated using the finite length analysis of the corresponding LT-Code and of the pre-code $\mathcal{C}$. This can be done for any code $\mathcal{C}$ with a decoder for which the decoding error probability is completely known. For example, $\mathcal{C}$ can be chosen from the ensemble $\mathbf{P}(\Lambda(x), n, r)$.

We will assume throughout that the k input symbols of the Raptor Code need to be recovered from $k(1 + \varepsilon)$ output symbols. Suppose that $\mathcal{C}$ has block-length n. For any ℓ with $0 \leq \ell \leq n$ let p_ℓ denote the probability that the LT-decoder fails after recovering ℓ of the n intermediate symbols. Further, let q_ℓ denote the probability that the code $\mathcal{C}$ cannot decode ℓ erasures at random positions.

Since the LT-decoding process is independent of the choice of $\mathcal{C}$, the set of unrecovered intermediate symbols at the point of failure of the decoder is random. Therefore, if z denotes the probability that the k input symbols cannot be recovered from the $k(1+\varepsilon)$ output symbols, then we have

$$z = \sum_{\ell=0}^{n} p_\ell q_{n-\ell}.$$

Using the results of the previous two subsections, it is possible to obtain good upper bounds on the overall error probability of Raptor Codes.

7.5 Finite Length Design

As an example of the foregoing discussions we will give one possible design for Raptor codes for which the number of input symbols k is larger than or equal to 64536. In this case, we first encode the k input symbols using an extended Hamming code. This increases the number of symbols to a number $\tilde{k}$ which is roughly $k + \lceil \log_2(k) \rceil$.

The reason to choose the extended Hamming code as a first stage of the pre-coding is to reduce the effect of stopping sets of very small size, since an extended Hamming code has minimum distance 4 and thus takes care of stopping sets of sizes 2 and 3. Moreover, stopping sets of larger sizes can also be resolved, with a good probability, using an extended Hamming code.

Next, we use a random code from the ensemble $\mathbf{P}(x^4, \tilde{k} + 1000, 580)$ to pre-code the $\tilde{k}$ symbols and produce $\tilde{k} + 1000$ intermediate symbols. Then we use an LT-Code with the degree distribution for length 65536 given in Table 1. The overall Raptor Code in this case is shown in Figure 4.

Using the results of the previous sections, we can calculate an upper bound on the probability of error for this Raptor code. For any s, let $p(s)$ be the probability that the Raptor code fails to recover a subset of size s within the $\tilde{k} + 1000$ symbols on which the LT-encoding is performed. Figure 5 shows an upper bound on $p(s)$ as s grows. The addition of the Hamming code at the beginning results in $p(s) = 0$ for $s = 1, 2, 3$, and reduces the upper bound on the overall block error probability significantly to 1.71×10^{-14}.

8 Systematic Raptor Codes

One of the disadvantages of Raptor Codes is that they are not systematic. This means that the input symbols are not necessarily reproduced by the encoder. As many applications require systematic

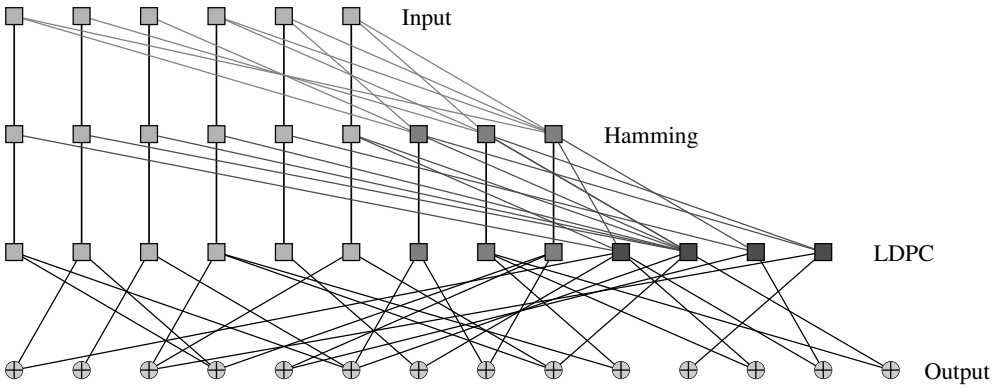


Figure 4: One version of Raptor codes: the pre-coding is done in multiple stages. The first stage is Hamming coding, and the second stage is a version of LDPC coding.

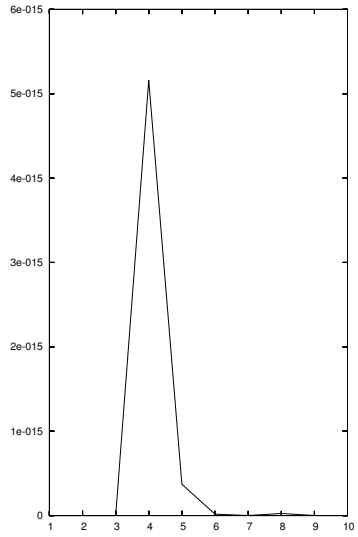


Figure 5: Upper bound on the probability $p(s)$ that the Raptor Code of Section 7 cannot recover a subset of size s for small values of s

codes for better performance, we will design in this section systematic versions of Raptor Codes.

Throughout this section we will assume that we have a Raptor Code with parameters $(k, \mathcal{C}, \Omega(x))$ which has a reliable decoding algorithm of overhead $1+\varepsilon$. We denote by n the block-length of the pre-code $\mathcal{C}$.

We will design an encoding algorithm which accepts k input symbols $x_1, \ldots, x_k$ and produces a set $\{i_1, \ldots, i_k\}$ of k distinct indices between 1 and $k(1+\varepsilon)$ and an unbounded string $z_1, z_2, \ldots$ of output symbols such that $z_{i_1} = x_1, \ldots, z_{i_k} = x_k$, and such that the output symbols can be computed efficiently. Moreover, we will also design a reliable decoding algorithm of overhead $1+\varepsilon$ for this code.

In the following we will refer to the indices $i_1, \ldots, i_k$ as the *systematic positions*, we will call the output symbols $z_{i_1}, \ldots, z_{i_k}$ the *systematic* output symbols, and we will refer to the other output symbols as the *non-systematic* output symbols.

8.1 Summary of the Approach

The overall structure of our approach is as follows. We will first compute the systematic positions $i_1, \ldots, i_k$. This process also yields an invertible binary $k \times k$-matrix R. These data are computed by sampling $k(1+\varepsilon)$ times from the distribution $\Omega(x)$ independently to obtain vectors $v_1, \ldots, v_{k(1+\varepsilon)}$ and applying a modification of the decoding algorithm to these vectors. The matrix R will be the product of the matrix A consisting of the rows $v_{i_1}, \ldots, v_{i_k}$ and a generator matrix of the pre-code. These sampled vectors also determine the first $k(1+\varepsilon)$ output symbols of the systematic encoder.

To encode the input symbols $x_1, \ldots, x_k$ we first use the inverse of the matrix R to transform these into intermediate symbols $y_1, \ldots, y_k$. We then apply the Raptor Code with parameters $(k, \mathcal{C}, \Omega(x))$ to the intermediate symbols, whereby the first $k(1+\varepsilon)$ symbols are obtained using the previously sampled vectors $v_1, \ldots, v_{k(1+\varepsilon)}$. All this will be done in such a way that the output symbols corresponding to the systematic positions coincide with the input symbols.

The decoding process for the systematic Raptor Code will consist of a decoding step for the original Raptor Code to obtain the intermediate symbols $y_1, \ldots, y_k$. The matrix R is then used to transform these intermediate symbols back to the input symbols $x_1, \ldots, x_k$.

In the next section we will introduce a matrix interpretation of the encoding and decoding procedures for Raptor Codes and use this point of view to describe our encoding and decoding algorithms for systematic Raptor Codes.

8.2 A Matrix Interpretation of Raptor Codes

The encoding procedure for a Raptor Code amounts to performing multiplications of matrices with vectors, and to solving systems of equations. The matrices involved are binary, i.e., their coefficients are either zero or one. The vectors on the other hand will be vectors of symbols, where each symbol is a binary digit, or itself a binary vector. We will always view vectors as row vectors.

For the rest of this paper we will fix a generator matrix G of the pre-code. G is an $n \times k$ binary matrix. Let x denote the vector $(x_1, \ldots, x_k)$ consisting of the input vectors. The pre-coding step of the Raptor Code corresponds to the multiplication $u^\top := G \cdot x^\top$.

Each output symbol of the Raptor Code is obtained by sampling independently from the distribution $\Omega(x)$ to obtain a row vector v in $\mathbb{F}_2^n$. The value of the output symbol is calculated as the scalar product $v \cdot u^\top$. We call the vector v the *vector corresponding to the output symbol.*

To any given set of N output symbols of the Raptor Code there corresponds a binary $N \times n$-matrix S in which the rows are the vectors corresponding to the output symbols. In other words, the rows of S are sampled independently from the distribution $\Omega(x)$, and we have

$$S \cdot G \cdot x^\top = z^\top, \tag{1}$$

where $z = (z_1, \ldots, z_N)$ is the column vector consisting of the output symbols. Decoding the Raptor Code corresponds to solving the system of equations given in (1) for the vector x.

8.3 The systematic positions $i_1, \ldots, i_k$ and the Matrix R

In this section we will discuss the problem of calculating the systematic positions, and the matrix R. Moreover, we will study the cost of multiplication of R with a generic vector of length k, and the cost of solving a system of equations $R \cdot x^\top = b$ for x, where b is a given vector of length k. In order to make assertions on the cost, it is advantageous to introduce a piece of notation. For a matrix M we will denote by $L(M)$ the number of arithmetic operations sufficient to calculate the product $M \cdot x^\top$ of the matrix M with a generic vector x, divided by the number of rows of M. This is the number of arithmetic operations per entry of the product $M \cdot x^\top$. In this sense $L(R)$ is the cost of multiplying R with a generic column vector, and $L(R^{-1})$ is the cost of solving the system of equations $R \cdot x^\top = b$ for a generic vector b.

The system (1) is solvable if and only if the rank of $S \cdot G$ is k. Gaussian elimination identifies k rows with indices $i_1, \ldots, i_k$ such that the submatrix A of S consisting of these rows has the property

that $R := A \cdot G$ is an invertible $k \times k$-matrix. This gives us the following algorithm for calculating R and the systematic indices.

Algorithm 7. Input: *Raptor Code with parameters $(k, \mathcal{C}, \Omega(x))$, and positive real number ε.*

Output: *If successful, vectors $v_1, \ldots, v_{k(1+\varepsilon)} \in \mathbb{F}_2^n$, indices $i_1, \ldots, i_k$ between 1 and $k(1+\varepsilon)$, and invertible matrix $R = A \cdot G$ such that A is the matrix formed by rows $v_{i_1}, \ldots, v_{i_k}$.*

(1) *Sample $k(1+\varepsilon)$ times independently from the distribution $\Omega(x)$ on $\mathbb{F}_2^n$ to obtain $v_1, \ldots, v_{k(1+\varepsilon)}$.*

(2) *Calculate the matrix S consisting of rows $v_1, \ldots, v_{k(1+\varepsilon)}$ and the product $S \cdot G$.*

(3) *Using Gaussian elimination, calculate rows $i_1, \ldots, i_k$ such that the submatrix R of $S \cdot G$ consisting of these rows is invertible, and calculate R^{-1}. If the rank of $S \cdot G$ is less than k, output an error flag.*

Theorem 8. (1) *If the decoding algorithm for the Raptor Code errs with probability p, then the probability that Algorithm 7 fails is at most p.*

(2) *The algorithm computes the matrix R and its inverse with $O(k^3 + n^2 k)$ binary arithmetic operations.*

(3) $L(R^{-1}) = O(k^2)$.

(4) *With high probability (over the choice of the v_j) $L(R)$ is upper bounded by $(1+\varepsilon)\Omega'(1)+\gamma+o(1)$, where γ is the encoding cost of $\mathcal{C}$, and $o(1)$ is a function approaching 0 as k approaches infinity.*

Proof. Let S be the matrix whose rows are the vectors $v_1, \ldots, v_{k(1+\varepsilon)}$. The system of equations (1) is solvable if and only if the rank of $S \cdot G$ is k. The probability of solving the system using the decoding algorithm for the Raptor Code is p, hence the probability that the rank of $S \cdot G$ is smaller than k is at most p. This proves (1).

The matrix $S \cdot G$ can be calculated with $O(n^2 k)$ operations. The matrix R and its inverse, and the systematic indices $i_1, \ldots, i_k$ can be obtained using a standard application of Gaussian elimination to the matrix $S \cdot G$. This step uses $O(k^3)$ operations, and proves (2).

It is easily seen that $L(M) = O(k^2)$ for any $k \times k$-matrix, so (3) follows.

The multiplication $R \cdot x^\top$ can be performed by first multiplying G with $x^\top$ to obtain an n-dimensional vector u, and then multiplying $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ with $u^\top$. The cost of the second step

is the average weight of the v_{i_ℓ}. To obtain an upper bound on this average weight, note that a standard application of the Chernoff bound shows that the sum of the weights of $v_1, \ldots, v_{k(1+\varepsilon)}$ is $k(1+\varepsilon)\Omega'(1) + o(k)$, with high probability. The average weight of the vectors $v_{i_1}, \ldots, v_{i_k}$ is therefore at most $k(1+\varepsilon)\Omega'(1)/k + o(1) = (1+\varepsilon)\Omega'(1) + o(1)$. This shows (4). □

The above algorithm can be simplified considerably for the case of LT-Codes by a slight adaptation and modification of BP decoding.

Algorithm 9. Input: *LT-Code with parameters $(k, \Omega(x))$, and positive real number ε.*

Output: *If successful, vectors $v_1, \ldots, v_{k(1+\varepsilon)}$, indices $i_1, \ldots, i_k$ between 1 and $k(1+\varepsilon)$, and invertible matrix R formed by rows $v_{i_1}, \ldots, v_{i_k}$.*

(1) *Sample $k(1+\varepsilon)$ times independently from the distribution $\Omega(x)$ on $\mathbb{F}_2^n$, where n is the block-length of $\mathcal{C}$, to obtain $v_1, \ldots, v_{k(1+\varepsilon)}$, and let S denote the matrix formed by these vectors as its rows.*

(2) *Set counter $\mathsf{c} = 0$, and matrix $M := S$, and loop through the following steps:*

(2.1) *If $\mathsf{c} < k$, identify a row of weight 1 of M; flag an error and stop if it does not exist; otherwise, set i_{c} to be equal to the index of the row in S.*

(2.2) *Identify the unique nonzero position of the row, and delete the column corresponding to that position from M.*

(3) *Set R equal to the rows $i_1, \ldots, i_k$ of S.*

Theorem 10. (1) *Suppose that BP decoding is an algorithm of overhead $1+\varepsilon$ for the above LT-Code, and suppose that it errs with probability p. Then the probability that Algorithm 9 is at most p.*

(2) *The matrix R can be calculated with at most $k\Omega'(1)(1+\varepsilon)$ arithmetic operations.*

(3) *With high probability (over the choice of the v_j) $L(R^{-1})$ is upper bounded by $(1+\varepsilon)\Omega'(1)+o(1)$, where $o(1)$ is a function approaching 0 as k approaches infinity.*

(4) *With high probability (over the choice of the v_j) $L(R)$ is upper bounded by $(1+\varepsilon)\Omega'(1)+o(1)$.*

Proof. The assertions follow from the fact that the algorithm provided is essentially the BP decoding algorithm (except that it does not perform any operations on symbols, and it keeps track of the rows of S which participate in the decoding process). The assertion on the cost of calculating R, and the cost $L(R)$ follow from the upper bound $(1+\varepsilon)\Omega'(1) + o(1)$ on the average weights of the vectors $v_{i_1}, \ldots, v_{i_k}$. (See the proof of the previous proposition.)

To calculate $L(R^{-1})$, note that the success of the algorithm shows that with respect to a suitable column and row permutation R is lower triangular with 1's on the main diagonal. Hence, the cost $L(R^{-1})$ is equal to the average weight of R and the assertion follows. □

In what follows we assume that the matrix R, the vectors $v_1, \ldots, v_{k(1+\varepsilon)}$, and the systematic positions $i_1, \ldots, i_k$ have already been calculated, and that this data is shared by the encoder and the decoder. The systematic encoding algorithm flags an error if Algorithm 7 (or its LT-analogue) fails to calculate this data.

8.4 Encoding Systematic Raptor Codes

The following algorithm describes how to generate the output symbols for a systematic Raptor Code.

Algorithm 11. Input: *Input symbols* $x_1, \ldots, x_k$.

Output: *Output symbols* $z_1, z_2, \ldots$, *where for* $1 \leq i \leq k(1+\varepsilon)$ *the symbol* z_i *corresponds to the vectors* v_i, *and where* $z_{i_j} = x_j$ *for* $1 \leq j \leq k$.

1. *Calculate* $y = (y_1, \ldots, y_k)$ *given by* $y^\top = R^{-1} \cdot x^\top$.
2. *Encode* y *using the generator matrix* G *of the pre-code* $\mathcal{C}$ *to obtain* $u = (u_1, \ldots, u_n)$, *where* $u^\top = G \cdot y^\top$.
3. *Calculate* $z_i := v_i \cdot u^\top$ *for* $1 \leq i \leq k(1+\varepsilon)$.
4. *Generate the output symbols* $z_{k(1+\varepsilon)+1}, z_{k(1+\varepsilon)+2}, \ldots$ *by applying the LT-Code with parameters* $(k, \Omega(x))$ *to the vector* u.

We will first show that this encoder is indeed a systematic encoder with systematic positions $i_1, \ldots, i_k$. This also shows that it is not necessary to calculate the output symbols corresponding to these positions.

Proposition 12. *In Algorithm 11 the output symbols z_{i_j} coincide with the input symbols x_{i_j} for $1 \leq j \leq k$.*

Proof. Note that $R = A \cdot G$, where the rows of A are $v_{i_1}, \ldots, v_{i_k}$. We have $R \cdot y^\top = x^\top$, i.e., $A \cdot u^\top = x^\top$, since $u^\top = G \cdot y^\top$. Hence, for all j, $v_{i_j} \cdot u^\top = x_j$, and we are done. □

Next, we focus on the cost of the encoding algorithm.

Theorem 13. *The cost of Algorithm 11 is at most $L(R^{-1}) + \alpha$, where α is the encoding cost of the Raptor Code. In particular, if the Raptor Code is an LT-Code, then the cost of this algorithm is at most $(2 + \varepsilon)\Omega'(1) + o(1)$.*

Proof. Computation of y has cost $L(R^{-1})$. Encoding y has cost γ, where γ is the encoding cost of the pre-code. Calculation of each of the z_i has expected average cost of $\Omega'(1)$. Therefore, the total cost is $L(R^{-1}) + \gamma + \Omega'(1)$. But $\gamma + \Omega'(1)$ is the encoding cost of the Raptor Code, hence the assertion follows.

If the Raptor Code is an LT-Code, then $L(R^{-1})$ is at most $(1 + \varepsilon)\Omega'(1) + o(1)$ by Theorem 10, and $\gamma = 0$. □

8.5 Decoding Systematic Raptor Codes

The decoder for the systematic Raptor Code collects $k(1+\varepsilon)$ output symbols and recovers the input symbols with high probability.

Algorithm 14. Input: *Output symbols $u_1, \ldots, u_m$, where $m = k(1 + \varepsilon)$.*

Output: *The input symbols $x_1, \ldots, x_k$ of the systematic Raptor Code.*

(1) *Decode the output symbols using the decoding algorithm for the original Raptor Code to obtain the intermediate symbols $y_1, \ldots, y_k$. Flag an error if decoding is not successful.*

(2) *Calculate $x^\top = R \cdot y^\top$, where $y = (y_1, \ldots, y_k)$, and $x = (x_1, \ldots, x_k)$.*

As in the case of the encoding algorithm, we will first focus on the correctness of the algorithm.

Proposition 15. *The output of Algorithm 14 is equal to the input symbols of the systematic encoder, and the error probability of this algorithm is equal to the error probability of the decoding algorithm used in Step 1.*

Proof. The output symbols $u_1, \ldots, u_m$ are independently generated output symbols of a Raptor Code with parameters $(k, \mathcal{C}, \Omega(x))$ applied to the vector y. Therefore, the decoding algorithm used in Step 1 is successful with high probability, and it computes the vector y if it succeeds. Since $x^\top = R \cdot y^\top$, the correctness of the algorithm follows. □

Next we focus on the cost of the algorithm.

Theorem 16. *The cost of Algorithm 14 is at most $\alpha(1+\varepsilon) + \beta + \gamma + o(1)$, where α is the encoding cost and β is the decoding cost of the original Raptor Code, and γ is the encoding cost of the pre-code. If the Raptor Code is an LT-Code, then the cost of Algorithm 14 is at most $(2+\varepsilon)\Omega'(1)$.*

Proof. Step 1 of the algorithm has cost β, and $L(R) = \alpha(1+\varepsilon) + \gamma + o(1)$ by Theorem 8. If the Raptor Code is an LT-Code, then $\beta = O'(1)$, and $L(R)$ is upper bounded by $\alpha(1+\varepsilon) + o(1)$ by Theorem 10. □

We would like to remark that the above algorithm can be improved. For example, if all the systematic positions have been received, then there is no need to run the decoding algorithm at all. More generally, if t systematic positions have been received, then only $k-t$ input symbols need to be calculated. We leave it to the reader to show that the cost of calculating the missing input symbols is actually $\alpha \cdot (1 - t/k) + \beta + \gamma + o(1)$ if $t < k$.

8.6 Practical Considerations

Our first remark concerns the systematic positions. One would like these positions to be $1, 2, \ldots, k$. The reason we could not satisfy this condition is hidden in the proof of Proposition 15, since we needed to make sure that the collected output symbols are statistically independent. In practice, however, it is a good idea to permute the vectors $v_1, \ldots, v_{k(1+\varepsilon)}$ so that the systematic positions become the first k positions.

Next we remark that is is possible to reduce the error probability of the encoder considerably by generating many more initial vectors than $k(1+\varepsilon)$ in Algorithm 7 (or its LT-analogue). Depending on how many initial vectors are generated, this makes the error probability of the algorithm very small (for example much smaller than the probability of failure of the decoding algorithm 14).

There are various ways of improving the running time of the systematic decoder. For example, it is not necessary to entirely re-encode the vector y in Step 2 of Algorithm 14. This is because

the decoding process in Step 1 will have recovered a large fraction of the coordinate positions of the vector obtained by applying the pre-code to y. These coordinate positions do not need to be recalculated.

We also comment that in practice the cost of multiplying with R^{-1} in Algorithm 11 is much smaller than $O(k^2)$. This is because the matrix R can be "almost" upper trianangularized, i.e., R will be of the form $\begin{pmatrix} U & A \\ B & C \end{pmatrix}$, where U is an upper triangular matrix of large size, and C is invertible.

9 Acknowledgments

A number of people have helped with the design of Raptor Codes at various stages. First and foremost I would like to thank Michael Luby for sharing with me his insight into LT-Codes and for carefully proofreading previous versions of the paper. I am also grateful to Igal Sason for carefully reading a previous draft of the paper and pointing out a number of corrections.

Soren Lassen implemented the Raptor Codes reported in this paper and has since optimized the design and the implementation to reach the speeds reported in the introduction. I would also like to thank Richard Karp, Avi Wigderson, Vivek Goyal, Michael Mitzenmacher, and John Byers for many helpful discussions during various development phases of this project.

References

[1] M. Luby, "LT-codes," in *Proceedings of the ACM Symposium on Foundations of Computer Science (FOCS)*, 2002.

[2] P. Elias, "Coding for two noisy channels," in *Information Theory, Third London Symposium*, 1955, pp. 61–76.

[3] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 569–584, 2001.

[4] R. G. Gallager, *Low Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

[5] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *proceedings of ACM SIGCOMM '98*, 1998.

[6] M. Luby, "Information additive code generator and decoder for communication systems," U.S. Patent No. 6,307,487, Oct. 23, 2001.

[7] M. Luby, "Information additive code generator and decoder for communication systems," U.S. Patent No. 6,373,406, April 16, 2002.

[8] A. Shokrollahi, S. Lassen, and M. Luby, "Multi-stage code generator and decoder for communication systems," U.S. patent application 20030058958, Serial No. 032156, December 2001.

[9] P. Maymounkov, "Online codes," Submitted for publication, 2002.

[10] R. Karp, M. Luby, and A. Shokrollahi, "Finite length analysis of LT-codes," To appear, 2002.

[11] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," in *Proc. 2nd International Symposium on Turbo Codes*, 2000, pp. 1–8.

[12] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585–598, 2001.

[13] A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," in *Proceedings of the 13th International Symposium on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, M. Fossorier, H. Imai, S. Lin, and A. Poli, Eds., 1999, number 1719 in Lecture Notes in Computer Science, pp. 65–76.

[14] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 3017–3028, 2002.

[15] A. Shokrollahi and R. Urbanke, "Finite length analysis of a certain class of LDPC codes," Unpublished, 2001.

[16] M. Luby, M. Mitzenmacher, and A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 364–373.

[17] M. Luby, "Design of degree distributions," Private Communication, 2001.

[18] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inform. Theory*, vol. 48, pp. 1570–1579, 2002.