Practical Loss-Resilient Codes

Michael G. Luby[*] Michael Mitzenmacher[†] M. Amin Shokrollahi[‡]

Daniel A. Spielman[§] Volker Stemann[¶]

Abstract

We present randomized constructions of linear-time encodable and decodable codes that can transmit over lossy channels at rates extremely close to capacity. The encoding and decoding algorithms for these codes have fast and simple software implementations. Partial implementations of our algorithms are faster by orders of magnitude than the best software implementations of any previous algorithm for this problem. We expect these codes will be extremely useful for applications such as real-time audio and video transmission over the Internet, where lossy channels are common and fast decoding is a requirement.

Despite the simplicity of the algorithms, their design and analysis are mathematically intricate. The design requires the careful choice of a random irregular bipartite graph, where the structure of the irregular graph is extremely important. We model the progress of the decoding algorithm by a set of differential equations. The solution to these equations can then be expressed as polynomials in one variable with coefficients determined by the graph structure. Based on these polynomials, we design a graph structure that guarantees successful decoding with high probability.

1 Introduction

Studies show that the Internet exhibits packet loss, and the measurements in [10] show that the situation has become worse over the past few years. A standard solution to this problem is to request retransmission of data that is not received. When some of this retransmission is lost, another request is made, and so on. In some applications, this introduces technical difficulties. For real-time transmission this solution can lead to unacceptable delays caused by several rounds of communication between sender and receiver. For a multicast protocol with one sender and many receivers, different sets of receivers can lose different sets of packets, and this solution can add significant overhead to the protocol.

An alternative solution, often called *forward error-correction* in the networking literature, is sometimes desirable. Consider an application that sends a real-time stream of data symbols that is partitioned and transmitted in logical units of blocks.[1] Suppose the network experiences transient and unpredictable losses of at most a p fraction of symbols out of each block. The following insurance policy can be used to tradeoff the effects of such uncontrollable losses on the receiver for controllable degradation in quality. Let n be the block length. Instead of sending blocks of n data symbols each, place $(1-p)n$ data symbols in each block, by either selecting the most important parts from the original data stream and omitting the remainder, or by generating a slightly lower quality stream at a $(1-p)$ fraction of the original rate. Fill out the block to its original length of n with pn redundant (check) symbols. This scheme provides optimal loss protection if the $(1-p)n$ symbols in the message can all be recovered from any set of $(1-p)n$ received sym-

[*]Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, and Computer Science Division, University of California at Berkeley. A substantial portion of this research done while at the International Computer Science Institute. Research supported in part by National Science Foundation operating grant NCR-9416101, and United States-Israel Binational Science Foundation grant No. 92-00226.

[†]Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. A substantial portion of this research done while at the Computer Science Department, UC Berkeley, under the National Science Foundation grant No. CCR-9505448.

[‡]International Computer Science Institute Berkeley, and Institut für Informatik der Universität Bonn, Germany. Research supported by a Habilitationsstipendium of the Deutsche Forschungsgemeinschaft, Grant Sh 57/1–1.

[§]Department of Mathematics, M.I.T. Supported by an NSF mathematical sciences postdoc. A substantial portion of this research done while visiting U.C. Berkeley.

[¶]Research done while at the International Computer Science Institute.

[1]An example of this is an MPEG stream, where a *group of pictures* can constitute such a block, and where each symbol corresponds to the contents of one packet in the block. The latency incurred by the application is proportional to the time it takes between when the first and last packet of the block is sent, plus the one-way travel time through the network.

bols from the block. Such a scheme can be used as the basic building block for the more robust and general protection scheme described in [1].

To demonstrate the benefits of forward error-correction in a simplified setting, consider the Internet loss measurements performed by [14] involving a multicast by one sender to a number of geographically distributed receivers. In one typical transmission to eleven receivers for a period of an hour, the average packet loss rate per receiver was 9.3%, yet 46.5% of the packets were lost by at least one of the receivers. To simplify the example, suppose that the maximum loss rate per block of 1000 packets for any receiver is 200.[2] Then, the sender could use forward error-correction by adding 200 redundant packets to each 800 data packets to form blocks consisting of 100 packets each. This approach sends 25% more packets total than are in the original data stream, whereas a naive scheme where the sender retransmits lost packets would send about 50% more.

It is a challenge to design fast enough encoding and decoding algorithms to make forward error-correction feasible in real-time for high bandwidth applications. In this paper, we present codes that can be encoded and decoded in linear time while providing near optimal loss protection. Moreover, these linear time algorithms can be implemented to run very quickly in software.

Our results hold whether each symbol is a single bit or a *packet* of many bits. We assume that the receiver knows the position of each received symbol within the stream of all encoding symbols. This is appropriate for the Internet, where packets are indexed. We adopt as our model of losses the *erasure channel*, introduced by Elias [6], in which each encoding symbol is lost with a fixed constant probability p in transit independent of all the other symbols. This assumption is not appropriate for the Internet, where losses can be highly correlated and bursty. However, losses on the Internet in general are not sensitive to the actual contents of each packet, and thus if we place the encoding into the packets in a random order then the independent loss assumption is valid.

Elias [6] showed that the capacity of the erasure channel is $1 - p$ and that a random linear code can be used to transmit over the erasure channel at any rate $R < 1 - p$. Furthermore, a standard linear MDS code can be used to convert a message of length Rn into a transmission of length n from which the message can be recovered from any portion of length greater than Rn. Moreover, general linear codes have quadratic time encoding algorithms and cubic time decoding algorithms. One cannot hope for better information recovery, but faster encoding and decoding times are desirable, especially for real-time applications.

Reed-Solomon codes can be used to transmit at the capacity of the erasure channel with $n \log n$ encoding time and quadratic decoding time. These codes have recently been customized to compensate for Internet packet loss in real-time transmission of moderate-quality video [1]. Even this optimized implementation required the use of dedicated workstations. Transmission of significantly higher quality video requires faster coding algorithms.

In theory, it is possible to decode Reed-Solomon codes in time $\mathcal{O}(n \log^2 n \log \log n)$ (see, [4, Chapter 11.7] and [9, p. 369]). However, for small values of n, quadratic time algorithms are faster than the fast algorithms for the Reed-Solomon based codes, and for larger values of n the $\mathcal{O}(\log^2 n \log \log n)$ multiplicative overhead in the running time of the fast algorithms (with a moderate sized constant hidden by the big-Oh notation) is large, i.e., in the hundreds or larger.

We obtain very fast linear-time algorithms by transmitting just below channel capacity. We produce rate $R = 1 - p(1 + \epsilon)$ codes along with decoding algorithms that recover from the random loss of a p fraction of the transmitted symbols in time proportional to $n \ln(1/\epsilon)$ with high probability, where n is the length of the encoding. They can also be encoded in time proportional to $n \ln(1/\epsilon)$. In Section 7, we do this for all $\epsilon > 0$. The fastest previously known encoding and decoding algorithms [2] with such a performance guarantee have run times proportional to $n \ln(1/\epsilon)/\epsilon$. (See also [3] for related work.)

The overall structure of our codes are related to codes introduced in [13] for error-correction. We explain the general construction along with the encoding and decoding algorithms in Section 2.

Our encoding and decoding algorithms are almost symmetrical. Both are extremely simple, computing exactly one exclusive-or operation for each edge in a randomly chosen bipartite graph. As in many similar applications, the graph is chosen to be sparse, which immediately implies that the encoding and decoding algorithms are fast. Unlike many similar applications, the graph is not regular; instead it is quite irregular with a carefully chosen degree sequence. We describe the decoding algorithm as a process on the graph in Section 3. Our main tool is a model that characterizes almost exactly the performance of the decoding algorithm as a function of the degree sequence of the graph. In Section 4, we use this tool to model the progress of the decoding algorithm by a set of differential equations. As shown in Lemma 1, the solution to these equations can then be expressed as polynomials in one variable with coefficients determined by the degree sequence. The positivity of one of these polynomials on the interval $(0, 1]$ with respect to a parameter δ guarantees that, with high probability, the decoding algorithm can recover almost all the message symbols from a loss of up to a δ fraction of the encoding symbols. The complete success of the decoding algorithm can then be demonstrated by combinatorial arguments such as Lemma 3.

Our analytical tools allow us to almost exactly characterize the performance of the decoding algorithm for any

[2] This is twice the average loss rate, but due to the bursty nature of losses in the Internet it is still likely that the maximum loss rate per block exceeds 20%. One can use the results of [1] to simultaneously protect against various levels of loss while still keeping the overall redundancy modest.

given degree sequence. Using these tools, we analyze regular graphs in Section 6, and conclude that they cannot yield codes that are close to optimal. Hence irregular graphs are a necessary component of our design.

Not only do our tools allow us to analyze a given degree sequence, but they also help us to *design* good irregular degree sequences. In Section 7 we describe, given a parameter $\epsilon > 0$, a degree sequence for which the decoding is successful with high probability for a loss fraction δ that is within ϵ of $1 - R$. Although these graphs are irregular, with some nodes of degree $1/\epsilon$, the average node degree is only $\ln(1/\epsilon)$. This is the main result of the paper, i.e., a code with encoding and decoding times proportional to $\ln(1/\epsilon)$ that can recover from a loss fraction that is within ϵ of optimal.

In Section 9, we show how linear programming techniques can be used to find good degree sequences for the nodes on the right given a degree sequence for the left nodes. We demonstrate these techniques by finding the right degree sequences that are close to optimal for a series of example left degree sequences.

1.1 Terminology

The *block length* of a code is the number of symbols in the transmission. In a *systematic* code, the transmitted symbols can be divided into *message* symbols and *check* symbols. We take the symbols to be bits, and write $a \oplus b$ to denote the exclusive-or of bits a and b. It is easy to extend our constructions to work with symbols that are packets of bits: where we would take the $\oplus$ of two bits, just take the bit-wise $\oplus$ of two packets. The message symbols can be chosen freely, and the check symbols are computed from the message symbols. The *rate* of a code is the ratio of the number of message symbols to the block length. In a code of block length n and rate R, the encoder takes as input Rn message symbols and produces n symbols to be transmitted. In all of our constructions, we assume that the symbols are bits.

2 The Codes

In this section, we explain the overall construction, as well as the encoding and decoding algorithms. We begin by defining a code $\mathcal{C}(B)$ with n message bits and βn check bits, by associating these bits with a bipartite graph B. The graph B has n left nodes and βn right nodes, corresponding to the message bits and the check bits, respectively. $\mathcal{C}(B)$ is encoded by setting each check bit to be the $\oplus$ of its neighboring message bits in B (see Figure $1(a)$). Thus, the encoding time is proportional to the number of edges in B.

The main contribution of our work is the design and analysis of the bipartite graph B so that the repetition of the following simplistic decoding operation recovers all the missing message bits.

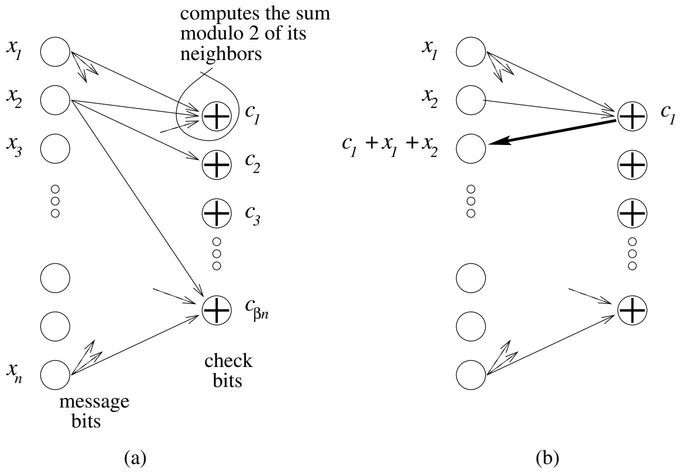

Figure 1: (a) A graph defines a mapping from message bits to check bits.
(b) Bits x_1, x_2, and c_1 are used to solve for x_3.

Decoding Operation

Given the value of a check bit and all but one of the message bits on which it depends, set the missing message bit to be the $\oplus$ of the check bit and its known message bits.

See Figure $1(b)$ for an example of this operation. The advantage of relying solely on this recovery operation is that the total decoding time is (at most) proportional to the number of edges in the graph. Our main technical innovation is in the design of sparse random graphs where repetition of this operation is guaranteed to usually recover all the message bits if at most $(1-\epsilon)\beta n$ of the message bits have been lost from $\mathcal{C}(B)$.

To produce codes that can recover from losses regardless of their location, we cascade codes of the form $\mathcal{C}(B)$: we first use $\mathcal{C}(B)$ to produce βn check bits for the original n message bits, we then use a similar code to produce $\beta^2 n$ check bits for the βn check bits of $\mathcal{C}(B)$, and so on (see Figure 2). At the last level, we use a more conventional loss-resilient code. Formally, we construct a sequence of codes $\mathcal{C}(B_1), \ldots, \mathcal{C}(B_m)$ from a sequence of graphs $B_0, \ldots, B_m$, where B_i has $\beta^i n$ left nodes and $\beta^{i+1} n$ right nodes. We select m so that $\beta^{m+1} n$ is roughly $\sqrt{n}$ and we end the cascade with a loss-resilient code C of rate $1 - \beta$ with $\beta^{m+1} n$ message bits for which we know how to recover from the random loss of β fraction of its bits with high probability. We then define the code $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ to be a code with n message bits and

$$\sum_{i=1}^{m+1} \beta^i n + \beta^{m+2} n/(1-\beta) = n\beta/(1-\beta)$$

check bits formed by using $\mathcal{C}(B_0)$ to produce βn check bits for the n message bits, using $\mathcal{C}(B_i)$ to form $\beta^{i+1} n$ check bits for the $\beta^i n$ bits produced by $\mathcal{C}(B_{i-1})$, and finally using C to produce an additional $n\beta^{m+2}/(1-\beta)$ check bits for the $\beta^{m+1}n$ bits output by $\mathcal{C}(B_m)$. As $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ has n message bits and $n\beta/(1-\beta)$ check bits, it is a code of rate $1-\beta$.

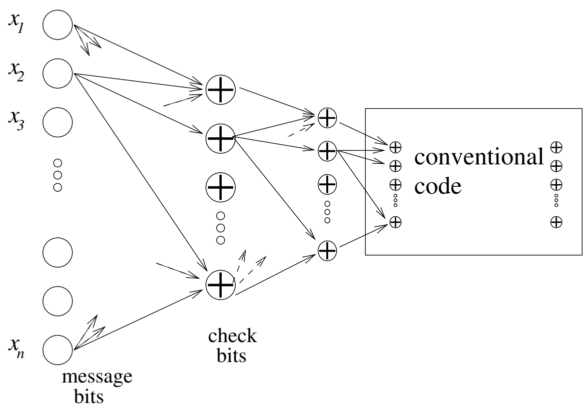

Figure 2: The code levels.

Assuming that the code C can be encoded and decoded in quadratic time[3], the code $\mathcal{C}(B_0, \ldots, B_m, C)$ can be encoded and decoded in time linear in n. We begin by using the decoding algorithm for C to recover losses that occur within its corresponding bits. If C recovers all the losses, then the algorithm now knows all the check bits produced by $\mathcal{C}(B_m)$, which it can then use to recover losses in the inputs to $\mathcal{C}(B_m)$. As the inputs to each $\mathcal{C}(B_i)$ were the check bits of $\mathcal{C}(B_{i-1})$, we can work our way back up the recursion until we use the check bits produced by $\mathcal{C}(B_0)$ to recover losses in the original n message bits. If we can show that C can recover from the random loss of a $\beta(1-\epsilon)$ fraction of its bits with high probability, and that each $\mathcal{C}(B_i)$ can recover from the random loss of a $\beta(1-\epsilon)$ fraction of its message bits with high probability, then we have shown that $\mathcal{C}(B_0, B_1, \ldots, B_m, C)$ is a rate $1-\beta$ code that can recover from the random loss of a $\beta(1-\epsilon)$ fraction of its bits with high probability. Thus, for the remainder of the paper, we only concern ourselves with finding graphs B so that the decoding algorithm can recover from a $\beta(1-\epsilon)$ fraction of losses in the message bits of $\mathcal{C}(B)$, given all of its check bits.

3 The Graph Process and Degree Sequences

We now relate the decoding process of $\mathcal{C}(B)$ to a process on a subgraph of B, so that hereafter we can use this simpler terminology when describing the process. This subgraph consists of all nodes on the left that were lost but have not been decoded thus far, all the nodes on the right, and all the edges between these nodes. Recall that the decoding process requires finding a check bit on the right such that only one adjacent message bit is missing; this adjacent bit can then be recovered. In terms of the subgraph, this is equivalent to finding a node of degree one on the right, and removing it, its neighbor, and all edges adjacent to its neighbor from the subgraph. We refer to this entire sequence of events hereafter as one step of the decoding process. We repeat this step until there are no nodes of degree one available on the right. The entire process is successful if it does not halt until all nodes on the left are removed, or equivalently, until all edges are removed.

The graphs that we use are chosen at random from carefully chosen degree sequence on sparse bipartite graphs. In contrast with many applications of random graphs in computer science, our graphs are not regular. Indeed, the analysis in Section 6 shows that it is not possible to approach channel capacity with regular graphs.

We refer to edges that are adjacent to a node of degree i on the left (right) as *edges of degree* i on the left (right). Each of our degree sequences is specified by a pair of vectors $(\lambda_1, \ldots, \lambda_m)$ and $(\rho_1, \ldots, \rho_m)$ where λ_i is the initial fraction of edges on the left of degree i and ρ_j is the initial fraction of edges on the right of degree j. Note that our graphs are specified in terms of fractions of *edges*, and not *nodes*, of each degree; this form is more convenient for much of our work. To choose an appropriate random bipartite graph B with E edges, n nodes on the left, and βn nodes on the right, we begin with a bipartite graph B' with E nodes on both the left and right hand sides, with each node of B' representing an edge slot. Each node on the left hand side of B' is associated with a node on the left side of B, so that the distribution of degrees is given by $(\lambda_1, \ldots, \lambda_m)$, and similarly for the right. We then choose a random matching (i.e., a random permutation) between the two sets of E nodes on B'. This induces a random bipartite graph on B (perhaps with multi-edges) in the obvious manner with the desired degree structure.

Note that, in the corresponding subgraph of B' remaining after each step, the matching remaining on B' still corresponds to a random permutation. Hence, conditioned on the degree sequence of the remaining subgraph after each step, the subgraph that remains is uniform over all subgraphs with this degree sequence. The evolution of the degree sequence is therefore a Markov process, a fact we make use of below.

In the next two sections, we develop techniques for the analysis of the process for general degree sequences. After using these techniques in Section 6 to analyze regular graphs, we describe degree sequences that result in codes that approach the capacity of the erasure channel in Section 7.

[3] A good candidate for the code C is the low-density parity-check [7, 12] version of these codes: only send the messages that cause all the check bits to be zero. These codes can be decoded in linear time and encoded in quadratic time with miniscule constants. In the final version of this paper, we show how C can be replaced with an even simpler code C' that can be encoded and decoded in linear time but that has a worse decoding guarantee. Using C', we can end the cascade with roughly $\epsilon n/\ln\ln n$ nodes instead of $\sqrt{n}$ for C.

4 Differential Equations Description

To analyze the behavior of the process on the subgraph of B described in the previous section, we begin by establishing a set of differential equations that describes its behavior in the limiting case, as the block length goes to infinity. Alternatively, one may think of these equations as describing the expected behavior of the associated random variables. Although these differential equations provide the key insight into the behavior of the decoding process, additional work is required to justify the relationship between the limiting and finite cases.

We begin with the initial random graph B, with n left nodes and βn right nodes. Consider the two vectors (λ_i) and (ρ_i), where λ_i and ρ_i are the fractions of edges of degree i on the left and right, with respect to the total number E of edges in the original graph. The average node degree on the left a_ℓ initially satisfies $a_\ell^{-1} = \sum_i \lambda_i/i$, and similarly the average node degree on the right a_r initially satisfies $a_r^{-1} = \sum_i \rho_i/i$.

We scale the passage of time so that each time unit of length $\Delta t := \frac{1}{E}$ corresponds to one step of the decoding process. Let δ be the fraction of losses in the message. Initially, just prior to time 0, each node on the left is removed with probability $1-\delta$ (because the corresponding message bit is successfully received), and thus the initial subgraph of B contains δn nodes on the left. If the process terminates successfully, it runs until time $\delta n/E = \delta/a_\ell$. We let $\ell_i(t)$ and $r_i(t)$ represent the fraction of edges (in terms of E) of degree i on the left and right, respectively, at time t. We denote by $e(t)$ the fraction of the edges remaining, that is, $e(t) = \sum_i \ell_i(t) = \sum_i r_i(t)$.

Recall that at each step, a random node of degree one on the right is chosen, and the corresponding node on the left and all of its adjacent edges are deleted. (If there is no such node, the process necessarily stops.) The probability that the edge adjacent to the node of degree one on the right has degree i on the left is $\ell_i(t)/e(t)$, and in this case we lose i edges of degree i. This gives rise to the difference equation

$$\mathcal{L}_i(t+\Delta t) - \mathcal{L}_i(t) = -\frac{i\ell_i(t)}{e(t)}$$

for the expected change of the *number* of edges $\mathcal{L}_i(t)$ of degree i on the left. Noting that $\ell_i(t) = \mathcal{L}_i(t)/E = \mathcal{L}_i(t)\Delta t$, we see that in the limit as $E \to \infty$ the solution of this difference equation is described by that of the differential equation

$$\frac{\mathrm{d}\ell_i(t)}{\mathrm{d}t} = -\frac{i\ell_i(t)}{e(t)}. \tag{1}$$

When we remove a node of degree i on the left, we remove the one edge of degree one from the right, along with the $i-1$ other edges adjacent to this node. Hence the expected number of other edges deleted is $a(t)-1$, where $a(t) = \sum i\ell_i(t)/e(t)$. The right endpoints of these $i-1$ other edges on the right hand side are randomly distributed. If one of these edges is of degree j on the right, we lose j edges of degree j, and gain $j-1$ edges of degree $j-1$. The probability that an edge has degree j on the right is just $r_j(t)/e(t)$. For $i > 1$, then, the difference equation

$$\mathcal{R}_i(t+\Delta t) - \mathcal{R}_i(t) = (r_{i+1}(t) - r_i(t))\frac{i(a(t)-1)}{e(t)}$$

describes the expected change of the *number* of edges $\mathcal{R}_i(t)$ of degree i on the right. The corresponding differential equations for the $r_i(t)$ are therefore

$$\frac{\mathrm{d}r_i(t)}{\mathrm{d}t} = (r_{i+1}(t) - r_i(t))\frac{i(a(t)-1)}{e(t)}. \tag{2}$$

(We assume that $r_i(t)$ is defined for all positive i, and is 0 for sufficiently large i.) The case $i=1$ plays a special role, as we must take into account that at each step an edge of degree one on the right is removed. Hence, the differential equation for $r_1(t)$ is given as

$$\frac{\mathrm{d}r_1(t)}{\mathrm{d}t} = (r_2(t) - r_1(t))\frac{(a(t)-1)}{e(t)} - 1 \tag{3}$$

Our key interest is in the progression of $r_1(t)$ at a function of t. As long as $r_1(t) > 0$, so that we have a node of degree one on the right, the process continues; when $r_1(t) = 0$ the process stops. Hence we would like for $r_1(t) > 0$ until all nodes on the left are deleted and the process terminates successfully.

These differential equations are more easily solved by defining x so that $\mathrm{d}x/x = \mathrm{d}t/e(t)$. The value of x in terms of t is then $x := \exp(-\int_0^t \mathrm{d}\tau/e(\tau))$. By substituting $\mathrm{d}x/x$ for $\mathrm{d}t/e(t)$, equation (1) becomes $\mathrm{d}\ell_i(x)/\mathrm{d}x = -i\ell_i(x)/x$, and integrating yields $\ell_i(x) = c_i x^i$. Note that $x = 1$ for $t = 0$, and $\ell_i(t=0) = \delta\lambda_i$. Hence, $c_i = \delta\lambda_i$ and

$$\ell_i(x) = \delta\lambda_i x^i.$$

Since $\ell_i(x)$ goes to zero as t goes to δ/a_ℓ, x runs over the interval $[1, 0)$.

Solving for the $r_i(x)$ is more involved. The main goal is to show that $r_1(x) > 0$ on $(0, 1]$, so that the process does not halt before completion. Details for solving for the $r_i(x)$, and in particular for $r_1(x)$, are given in Appendix A; the proposition below presents the solution for $r_1(x)$. The result is expressed using the degree sequence functions

$$\lambda(x) = \sum_{i \geq 1} \lambda_i x^{i-1}$$

and

$$\rho(x) = \sum_{i \geq 1} \rho_i x^{i-1}.$$

These degree sequence functions play an important role in the remainder of our presentation.

Lemma 1 *The solution $r_1(x)$ of the differential equation (3) is given by*

$$r_1(x) \quad = \quad \delta\lambda(x)\left[x - 1 + \rho\big(1 - \delta\lambda(x)\big)\right]. \tag{4}$$

From (4), we find that this requires the condition

$$\rho(1 - \delta\lambda(x)) > 1 - x \ , \ x \in (0, 1]. \quad (5)$$

This condition shall play a central role for the remainder of our work.

5 Using the Differential Equations to Build Codes

Recall that the differential equations describe the limiting behavior of the process on B, as the block length goes to infinity. From this one can derive that if (5) is violated, so that $\rho(1 - \delta\lambda(x)) < 1 - x$ somewhere on $(0, 1]$ then the process fails for large block lengths with high probability.

Proving the process progresses almost to completion if (5) is satisfied is possible by relating the differential equations to the underlying random process (see, e.g., [8, 11]). Proving the process runs to completion can be handled with a separate combinatorial argument. In some cases, however, this requires small modifications in the graph construction.

Lemma 2 *Let B be a bipartite graph chosen at random with edge-degrees specified by $\lambda(x)$ and $\rho(x)$. Let δ be fixed so that*

$$\rho(1 - \delta\lambda(x)) > 1 - x, \qquad \textit{for } x \in (0, 1].$$

For all $\eta > 0$, if the message bits of $\mathcal{C}(B)$ are lost independently with probability δ, then the decoding algorithm terminates with more than ηn message bits not recovered with exponentially small probability, as the block length grows large.

The following combinatorial tail argument is useful in showing that the process terminates successfully when there are no nodes on the left of degree one or two. (Such arguments are often required in similar situations; see, for example, [5].)

Lemma 3 *Let B be a bipartite graph chosen at random with edge-degrees specified by $\lambda(x)$ and $\rho(x)$, such that $\lambda(x)$ has $\lambda_1 = \lambda_2 = 0$. Then there is some $\eta > 0$, such that with probability $1 - o(1)$ (over the choice of B) if at most an η fraction of the nodes on the left in B remain, then the process terminates successfully.*

Proof: *[Sketch]* Let S be any set of nodes on the left of size at most ηn. Let a be the average degree of these nodes. If the number of nodes on the right that are neighbors of S is greater than $a|S|/2$, then one of these nodes has only one neighbor in $|S|$, and so the process can continue. Thus, we only need to show that the initial graph is a good expander on small sets. Since the degree of each node on the left is at least three, standard proofs (see [12]) suffice to show that the expansion condition holds with high probability for all sets containing at most an η fraction of the left nodes, for some $\eta > 0$. □

Using Lemmas 2 and 3, we can show that the codes presented in Section 7 work with high probability. As our results for asymptototically good general codes use graphs with degree two nodes on the left, we extend the construction and use more careful arguments to prove that the process completes, as described in Section 7.

6 Analysis of Regular Graphs

We demonstrate the techniques developed in the previous section in a simple setting: we analyze the behavior of the process on random regular graphs. Because random regular graphs have so often been used in similar situations, it is natural to consider if they give rise to codes that are close to optimal. They do not. The following lemma proves a weak form of this behavior. We have explicitly solved for the largest value of δ which satisfies condition (5) for codes based on regular graphs for several rate values. In every case, there is some small degree value that achieves a maximum value of δ far from optimal, and the maximal achievable value of δ decreases as the degree increases thereafter.

Lemma 4 *Let B be a bipartite regular graph chosen at random, with all edges having left degree d and right degree d/β, for $d \geq 3$. Then condition (5) holds only if*

$$\delta \leq 4\left(1 - \left(\frac{1}{d-1}\right)^{\frac{1}{(d/\beta)-1}}\right),$$

and hence as $d \to \infty$, the maximum acceptable loss rate goes to 0.

Proof: We have that $\lambda(x) = x^{d-1}$, and $\rho(x) = x^{(d/\beta)-1}$. Hence our condition (5) on the acceptable loss rate δ becomes

$$\left(1 - \delta x^{d-1}\right)^{(d/\beta)-1} > 1 - x$$

for $x \in (0, 1]$.

We plug in $x = 1 - \frac{1}{d-1}$. Using the fact that $(1 - \frac{1}{d-1})^{d-1} \geq \frac{1}{4}$ for $d \geq 3$ yields the requirement

$$\left(1 - \frac{\delta}{4}\right)^{(d/\beta)-1} \geq \frac{1}{d-1}.$$

This simplifies to the inequality in the statement of the lemma.

It is easy to check that as $d \to \infty$, the right hand side of the above equation goes to 0, which concludes the lemma. □

Hence, for any fixed β, there is some d which achieves the maximum value of δ possible using regular graphs, and this δ is far from optimal. For example, in the case where $\beta = 1/2$, the best solution is to let the left nodes have degree three and the right nodes have degree six. In this case the code can handle any δ fraction of losses on the left with high probability as long as

$$(1-\delta x^2)^5 \geq 1-x$$

for $x \in (0,1]$. The inequality fails for $\delta \geq 0.43$. When this one layer code is cascaded as described in Section 2 to a code with rate $R = 1/2$, simple calculations show that being able to incur a loss fraction of at most $\delta = .43$ in each layer implies that the message cannot be recovered from a portion of the encoding that is smaller than 1.14 times the length of the message, i.e., far from the optimal value of 1.00. Simulation runs of the code turn out to match this theory accurately, demonstrating that these techniques provide a sharp analysis of the actual behavior of the process.

7 Asymptotically Optimal Codes

In this section, we construct codes that transmit at rates arbitrarily close to the capacity of the erasure channel. We do this by finding an infinite sequence of solutions to the differential equations of Section 4 in which δ approaches p.

As the degree sequences we use have degree two nodes on the left hand side, we cannot appeal to Lemma 3 to show that they work with high probability. Hence our codes require some additional structure.

Let B be a bipartite graph with n left nodes and βn right nodes. We describe our choice for the left and right degree sequences of B that satisfy condition (5). Let d be a positive integer that is used to trade off the average degree with how well the decoding process works, i.e., how close we can make δ to $\beta = 1 - R$ and still have the process finish successfully most of the time.

The left degree sequence is described by the following truncated heavy tail distribution. Let $H(d) = \sum_{i=1}^{d} 1/i$ be the harmonic sum truncated at d, and thus $H(d) \sim \ln(d)$. Then, for all $i = 2, \ldots, d+1$, the fraction of edges of degree i on the left is given by $\lambda_i = 1/(H(d)(i-1))$. The average left degree a_ℓ equals $H(d)(d+1)/d$. Recall that we require the average right degree, a_r, to satisfy $a_r = a_\ell/\beta$. The right degree sequence is defined by the Poisson distribution with mean a_r: for all $i \geq 1$ the fraction of edges of degree i on the right equals

$$\rho_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!},$$

where α is chosen to guarantee that the average degree on the right is a_r. In other words, α satisfies $\alpha e^\alpha/(e^\alpha - 1) = a_r$.

Note that we allow $\rho_i > 0$ for all $i \geq 1$, and hence $\rho(x)$ is not truly a polynomial, but a power series. However, truncating the power series $\rho(x)$ at a sufficiently high term gives a finite distribution of the edge degrees for which the next lemma is still valid.

We show that when $\delta = \beta/(1+1/d)$ in (5), the condition is satisfied, i.e., $\rho(1-\delta\lambda(x)) > 1-x$ on $(0,1]$, where $\lambda(x) = \sum_i \lambda_i x^{i-1}$ and $\rho(x) = \sum_i \rho_i x^{i-1}$. Note that $\lambda(x)$ is the expansion of $-\ln(1-x)$ truncated at the dth term, and scaled so that $\lambda(1) = 1$. Further, $\rho(x) = e^{\alpha(x-1)}$.

Lemma 5 *With the above choices for $\rho(x)$ and $\lambda(x)$ we have $\rho(1-\delta\lambda(x)) > 1-x$ on $(0,1]$ if $\delta \leq \beta/(1+1/d)$.*

Proof: Recall that $\rho(x) = e^{\alpha(x-1)}$, hence $\rho(x)$ increases monotonically in x. As a result, we have

$$\rho(1-\delta\lambda(x)) > \rho(1+\delta\ln(1-x)/H(d)) = (1-x)^{\alpha\delta/H(d)}.$$

Since $a_\ell = H(d)(1+1/d)$ and $a_r = a_\ell/\beta$, we obtain $\alpha\delta/H(d) = (1-e^{-\alpha})(1+1/d)\delta/\beta < \delta(1+1/d)/\beta \leq 1$, which shows that the right hand side of the above inequality is larger than $1-x$ on $(0,1]$. □

A problem is that Lemma 3 does not apply to this system because there are nodes of degree two on the left. Indeed, simulations demonstrate that for these choices of $\lambda(x)$ and $\rho(x)$ a small number of nodes often do remain. To overcome this problem, we make a small change in the structure of the graph B. We split the βn right nodes of B into two distinct sets, the first set consisting of $(\beta-\gamma)n$ nodes and the second set consisting of γn nodes, where γ is a small constant to be determined. The graph B is then formed by taking the union of two graphs, B_1 and B_2. B_1 is formed as described up to this point between the n left nodes and the first set of $(\beta-\gamma)n$ right nodes. B_2 is formed between the n left nodes and the second set of γn right nodes, where each of the n left nodes has degree three and the $3n$ edges are connected randomly to the γn right nodes.

Lemma 6 *Let B be the bipartite graph just described. Then, with high probability, the process terminates successfully when started on a subgraph of B induced by δn of the left nodes and all βn of the right nodes, where $\delta = \beta/(1+1/d)$.*

Proof: *[Sketch]* In the analysis of the process, we may think of B_2 as being held in reserve to handle nodes not already dealt with using B_1. Combining Lemma 5 and Lemma 2, we can show that, for any constant $\eta > 0$, with high probability at most ηn nodes on the left remain after the process runs on B_1. The induced graph in B_2 between these remaining ηn left nodes and the γn right nodes is then a random bipartite graph such that all nodes on the left have degree three. We choose γ to be larger than η by a small constant factor so that with high probability the process terminates successfully on this subgraph by Lemma 3; note that the lemma applies since all nodes on the left have degree three. By choosing η sufficiently small, we may make γ arbitrarily small, and hence the decrease in the value of δ below that stated in Lemma 5 can be made arbitrarily small. □

Note that the degree of each left node in this modified construction of B is at most three bigger than the average degree of each left node in the construction of B described at the beginning of this section. We can use this observation and the lemma above to immediately prove our main theorem.

Theorem 1 *For any R, any positive ϵ, and sufficiently large block length n, there is a loss-resilient code that, with high probability, is able to recover from the random loss of a $(1 - R)(1 - \epsilon)$-fraction of its bits in time proportional to $n \ln(1/\epsilon)$.*

Proof: *[Sketch]* Set $d = 1/\epsilon$ to get a one level code with the properties described in Lemma 6. Cascade versions of these codes as described in Section 2 to get the entire code. The proof follows since each level of the cascade fails to recover all its message bits, given all of its check bits, with small probability. □

8 A Linear-Algebraic Interpretation

The βn check bits of the code $\mathcal{C}(B)$ described in Section 2 can be computed by multiplying the vector of n message bits by the $\beta n \times n$ matrix, $M(B)$, whose (i, j)-th entry is 1 if there is an edge in B between left node i and right node j and is 0 otherwise (the multiplication is over the field of two elements). We choose our graphs B to be sparse, so that the resulting matrix $M(B)$ is sparse and the multiplication can be performed quickly.

The efficiency of the decoding algorithm is related to the ease with which one can perform Gaussian elimination on submatrices of $M(B)$. If one knows all the check bits and all but δn of the message bits, then it is possible to recover the missing message bits if and only if the δn columns of $M(B)$ indexed by the message bits have full rank. These bits can be recovered by Gaussian elimination. In Section 7, we present a distribution on graphs B so that these bits can be recovered by a very simple brute-force Gaussian elimination: for any $\epsilon > 0$, we present a distribution of graphs B so that almost all subsets of $\beta/(1+\epsilon)$ columns of $M(B)$ have full rank and can be placed in lower-triangular form merely by permuting the rows of the matrix. Moreover, the average number of 1s per row in $M(B)$ is $n \ln(1/\epsilon)$; so, the Gaussian elimination can be performed in time $O(n \ln(1/\epsilon))$.

To contrast this with other constructions of loss-resilient codes, observe that most sets of $\beta n - 1$ columns of a random $\beta n \times n$ matrix have full rank. In fact, the probability that some $\beta n - c$ columns fail to have full rank is exponentiall small in c. However, we do not know of an algorithm that solves the resulting elimination problem in less than the time it takes to do a general matrix multiplication.

Ideally, we would use matrices in which every set of βn columns have full rank; but, such matrices do not exist over any constant-size field. However, if we let our field size grow to n, classical matrices solve this problem. For example, all subsets of βn columns of a $\beta n \times n$ Vandermonde matrix are linearly independent.

9 Finding Degree Sequences using Linear Programming

In this section we describe a heuristic approach that has proven effective in practice to find a good right degree sequence given a specific left degree sequence. The method uses linear programming and the differential equation analysis of Section 4.

Recall that, from the differential equations, a necessary condition for the process to complete is that $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$. We first describe a heuristic for determining for a given (finite) vector (λ_i) representing the left degree sequence and a value for δ whether there is an appropriate (finite) vector (ρ_i) representing the right degree sequence satisfying this condition. We begin by choosing a set M of positive integers which we want to contain the degrees on the right hand side. To find appropriate ρ_m, $m \in M$, we use the condition given by Lemma 1 to generate linear constraints that the ρ_i must satisfy by considering different values of x. For example, by examining the condition at $x = 0.5$, we obtain the constraint $\rho(1 - \delta\lambda(0.5)) > 0.5$, which is linear in the ρ_i.

We generate constraints by choosing for x multiples of $1/N$ for some integer N. We also include the constraints $\rho_m \geq 0$ for all $m \in M$. We then use linear programming to determine if suitable ρ_m exist that satisfy our derived constraints. Note that we have a choice for the function we wish to optimize; one choice that works well is to minimize the sum of $\rho(1 - \delta\lambda(x)) + x - 1$ on the values of x chosen to generate the constraints. The best value for δ for given N is found by binary search.

Given the solution from the linear programming problem, we can check whether the ρ_i computed satisfy the condition $\rho(1 - \delta\lambda(x)) > 1 - x$ on $(0, 1]$.

Due to our discretization, there are usually some *conflict subintervals* in which the solution does not satisfy this inequality. Choosing large values for the tradeoff parameter N results in smaller conflict intervals, although it requires more time to solve the linear program. For this reason we use small values of N during the binary search phase. Once a value for δ is found, we use larger values of N for that specific δ to obtain small conflict intervals. In the last step we get rid of the conflict intervals by appropriately decreasing the value of δ. This always works since $\rho(1 - \delta\lambda(x))$ is a decreasing function of δ.

We ran the linear programming approach on left degree sequences of the form $3, 5, 9, \ldots, 2^i + 1$ for codes with rates $1/2, 2/3, 3/4, 4/5, 9/10$ and average left degrees $5.70, 6.82, 8.01$. Figure 3 shows for each code how much of the encoding is sufficient to recover the entire message as a fraction of the message length as the message length goes to infinity. Since these graphs do not have nodes of degree two on the left, Lemma 3 and Lemma 2 imply that with high probability the corresponding codes recover the entire message from the portion of the encoding indicated in the table, provided the message length is large enough.

Average Degree	Rate				
	$1/2$	$2/3$	$3/4$	$4/5$	$9/10$
5.70	1.036	1.023	1.016	1.013	1.006
6.82	1.024	1.013	1.010	1.007	1.004
8.01	1.014	1.008	1.007	1.005	1.002

Figure 3: Close to optimal codes for different rates and average left degrees.

10 Conclusion

We have demonstrated a novel technique for the design and analysis of loss-resilient codes based on random bipartite graphs. Using differential equations derived from the underlying random process, we have obtained approximations of the behavior of this process that very closely match actual simulations. With these tools, we have designed codes with simple and fast linear-time encoding and decoding algorithms that can transmit over lossy channels at rates extremely close to capacity. Specifically, for any constant $\epsilon > 0$ and all sufficiently long block lengths n, we have constructed codes of rate R with encoding and decoding algorithms that run in time proportional to $n \ln(1/\epsilon)$ that can recover from almost all patterns of at most $(1 - R)(1 - \epsilon)n$ losses.

We expect these codes to have many practical applications. Encoding and decoding speeds are several orders of magnitude faster than previous software-based schemes, and are fast enough to be suitable for high bandwidth real-time applications such as high fidelity video. A preliminary implementation running on an Alpha BRET EV-5 330MHz machine can sustain encoding and decoding speeds of more than 100 Mbits/second with packets of length 2Kbit bits each and 100K packets per message and block length 200K (and thus the rate is 1/2).

We have also implemented error-correcting codes that use our novel graph constructions and decode with belief propogation techniques. Our experiments with these constructions yield dramatic improvements in the error recovery rate. We will report these results in a separate paper.

11 Acknowledgements

In the preliminary stages of this research, Johannes Blömer helped to design and test some handcrafted degree sequences that gave the first strong evidence that irregular degree sequences are better than regular degree sequences. Both Johannes and David Zuckerman worked on some of the preliminary combinatorial analysis of the decoding algorithm. We thank them for their contributions.

References

[1] A. Albanese, J. Blömer, J. Edmonds, M. Luby, M. Sudan, "Priority Encoding Transmission", *IEEE Transactions on Information Theory* (special issue devoted to coding theory), Vol. 42, No. 6, November 1996, pp. 1737–1744.

[2] N. Alon, J. Edmonds, M. Luby, "Linear Time Erasure Codes With Nearly Optimal Recovery", *Proc. of the 36^{th} Annual Symp. on Foundations of Computer Science*, 1995, pp. 512-519.

[3] N. Alon, M. Luby, "A Linear Time Erasure-Resilient Code With Nearly Optimal Recovery", *IEEE Transactions on Information Theory* (special issue devoted to coding theory), Vol. 42, No. 6, November 1996, pp. 1732–1736.

[4] R. E. Blahut, **Theory and Practice of Error Control Codes**, Addison Wesley, Reading, MA, 1983.

[5] A. Broder, A. Frieze, E. Upfal, 'On the Satisfiability and Maximum Satisfiability of Random 3-CNF Formulas", *Proc. of the 4^{th} ACM-SIAM Symp. on Discrete Algorithms*, 1993, pp. 322-330.

[6] P. Elias, "Coding for Two Noisy Channels" *Information Theory*, Third London Symposium, September 1955, Buttersworth's Scientific Publications, pp. 61-76.

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

[8] T.G. Kurtz, **Approximation of Population Processes**, CBMS-NSF Regional Conf. Series in Applied Math, SIAM, 1981.

[9] F. J. Macwilliams and N. J. A. Sloane, **The Theory of Error-Correcting Codes**, North Holland, Amsterdam, 1977.

[10] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics", *Ph.D. thesis*, UC Berkeley, 1997.

[11] A. Shwartz, A. Weiss, **Large Deviations for Performance Analysis**, Chapman & Hall, 1995.

[12] M. Sipser and D. Spielman, "Expander codes", *IEEE Transactions on Information Theory* (special issue devoted to coding theory), Vol. 42, No. 6, November 1996, pp. 1710–1722.

[13] D. Spielman, "Linear-Time Encodable and Decodable Error-Correcting Codes" *IEEE Transactions on Information Theory* (special issue devoted to coding theory), Vol. 42, No. 6, November 1996, pp. 1723–1731.

[14] M. Yajnik, J. Kurose, D. Towsley, "Packet Loss Correlation in the MBone Multicast Network", *IEEE Global Internet Conference*, London, November, 1996.

A Appendix

In this section we give an explicit solution to the system of differential equations given by (2) and (3). We start with the substitution $\mathrm{d}x/x = \mathrm{d}t/e(t)$, which gives $x := \exp(-\int_0^t \mathrm{d}\tau/e(\tau))$. This transforms for $i > 1$ Equation (2) to

$$r_i'(x) = i(r_i(x) - r_{i+1}(x))\frac{a(x) - 1}{x},$$

where prime stands for derivative with respect to the variable x. Note that the average degree $a(x)$ equals $\sum i\ell_i(x)/e(x)$, which in terms of the function $\lambda(x)$ can be written as $1 + x\lambda'(x)/\lambda(x)$. Hence, we obtain for $i > 1$

$$r_i'(x) = i(r_i(x) - r_{i+1}(x))\frac{\lambda'(x)}{\lambda(x)}.$$

These equations can be solved recursively, starting with the highest nonzero r_i. (In the case where there is no highest nonzero r_i, that is $r_i > 0$ for all i, the following equations are still valid, although stronger analysis tools are required to show that the solution is unique.)

The explicit solution is given by

$$r_i(x) = \lambda(x)^i\Big(-i\int_{y=1}^{x} r_{i+1}(y)\lambda(y)^{-i}\frac{\lambda'(y)}{\lambda(y)}\mathrm{d}x + c_i\Big) \quad (6)$$

for some constants c_i to be determined from the initial conditions for r_i. It is straightforward to check that

$$r_i(x) = \sum_{j\geq i}(-1)^{i+j}\binom{j-1}{i-1}c_j(\lambda(x))^j. \quad (7)$$

Further, one verifies directly that

$$c_i = \sum_{j\geq i}\binom{j-1}{i-1}r_j(1).$$

(Note that $x = 1$ corresponds to the beginning of the process.) We proceed with the determination of the expected value of $r_j(1)$: because each node on the left is deleted randomly just prior to time 0 with probability $1-\delta$, and the graph is a random graph over those with the given degree sequence, to the nodes on the right it is as though each edge is deleted with probability $1-\delta$. Hence, an edge whose right incident node had degree j before the deletion stage remains in the graph and has degree i afterwards with probability $\binom{j-1}{i-1}\delta^i(1-\delta)^{j-i}$. Thus

$$r_j(1) = \sum_{m\geq j}\rho_m\binom{m-1}{j-1}\delta^j(1-\delta)^{m-j}.$$

Plugging in the last formula in that of c_i we see that

$$c_i = \sum_{m\geq i}\binom{m-1}{i-1}\rho_m\delta^i.$$

(Use the identity $\binom{m-1}{j-1}\binom{j-1}{i-1} = \binom{m-1}{i-1}\binom{m-j}{j-i}$.) Hence, we obtain for $i > 1$ from (7)

$$r_i(x) = \sum_{m\geq j\geq i}(-1)^{i+j}\binom{j-1}{i-1}\binom{m-1}{j-1}\rho_m(\delta\lambda(x))^j. \quad (8)$$

To obtain the formula given for $r_1(x)$ in Lemma 1, we note that $r_1(x) = e(x) - \sum_{i>1} r_i(x)$. The sum of the right hand side of (8) over all $i \geq 1$ equals

$$\sum_{m\geq j}(-1)^{j-1}\binom{m-1}{j-1}\rho_m(\delta\lambda(x))^j\sum_{i\leq j}(-1)^{i-1}\binom{j-1}{i-1} = \delta\lambda(x).$$

(The inner sum equals 1 if $j = 1$, and is zero otherwise.) Hence, we have

$$\begin{aligned} r_1(x) &= e(x) - \delta\lambda(x) \\ &\quad + \delta\lambda(x)\sum_m \rho_m \sum_{j\leq m}(-1)^{j-1}\binom{m-1}{j-1}(\delta\lambda(x))^{j-1} \\ &= x\delta\lambda(x) - \delta\lambda(x) + \delta\lambda(x)\sum_m \rho_m(1-\delta\lambda(x))^{m-1} \\ &= \delta\lambda(x)\Big[x - 1 + \rho\big(1-\delta\lambda(x)\big)\Big]. \ \square \end{aligned}$$