Practical Loss-Resilient Codes

Michael Luby* Michael Mitzenmacher† Amin Shokrollahi‡
Daniel Spielman§ Volker Stemann¶

Abstract

We present a randomized construction 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.

---

*International Computer Science Institute, Berkeley, California. Portions of this research done while at Digital Equipment Corporation, Systems Research Center, Palo Alto, CA. 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 Introduction

In many communication situations, data is lost in transit. A standard response 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. Such communication protocols can lead to delays due to the need for several rounds of communication between sender and receiver.

An alternative solution, often called *forward error-correction* in the networking literature, is sometimes desirable: Suppose an application sends a real-time stream of data symbols that is partitioned and transmitted in logical units of blocks.[1] Furthermore, 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. Suppose originally a particular block consists of n data symbols. Instead of sending a message of n data symbols, send a message of only $(1-p)n$ data symbols, 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 symbols 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].

The problem is to design fast enough encoding and decoding algorithms to make this solution feasible. 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 $R < 1-p$. This means that a random linear 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 most portions of length greater than Rn. Moreover, every linear code has quadratic time encoding algorithms and cubic time decoding algorithms. One cannot hope for better information recovery, but

[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.

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 order $n \log n$ 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. 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 [12] 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 XOR 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 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 degree of each nodes 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 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 elements of GF(2), and all arithmetic operations to be over this field; i.e., addition is equivalent to taking the exclusive-or of two elements. 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. For example, 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. It is easy to extend our constructions to work with symbols that are packets of bits: where we would take the sum of two bits, just take the bit-wise sum of two packets.

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. The encoding consists of computing each check bit as the sum of the bits of its neighbors 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.

> **If one knows the value of a check bit and all but one of the message bits on which it depends,**
> **Then the value of the missing message bit can be found by computing the sum 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

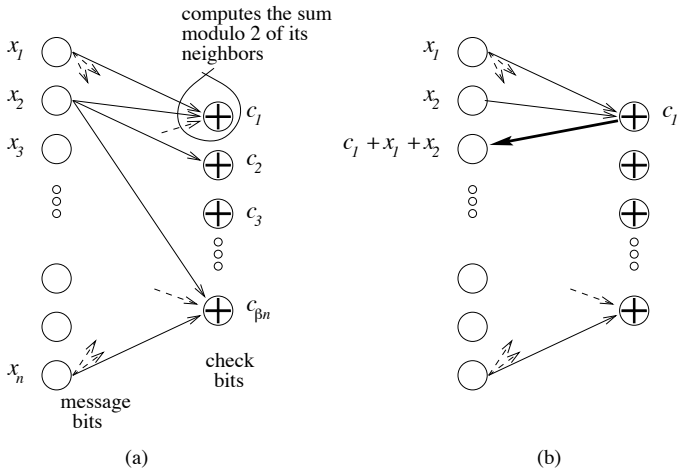

Figure 1: (a) A graph defines a mapping from message bits to check bits. (b) Bits x_1, x_2, and c_1 are used to solve for x_3.

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 begin with 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, to construct a sequence of codes $\mathcal{C}(B_1), \ldots, \mathcal{C}(B_m)$. 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 $1-\beta$.

Assuming that the code C can be encoded and decoded in quadratic time[2], the code

[2] A good candidate for the code C is the low-density parity-check [7, 11] 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 ϵn nodes instead of $\sqrt{n}$ for C.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.