TOWSON UNIVERSITY

OFFICE OF GRADUATE STUDIES

INFORMATION RECONCILIATION FOR ERASURE CHANNELS

by

Kyle Sherbert

A thesis

Presented to the faculty of

Towson University

in partial fulfillment

of the requirements for the degree

Master of Science

Department of Computer and Information Sciences

Towson University
Towson, Maryland 21252

May 2017

TOWSON UNIVERSITY
OFFICE OF GRADUATE STUDIES

THESIS APPROVAL PAGE

This is to certify that the thesis prepared by

Kyle Sherbert entitled

Information Reconciliation for Erasure Channels

has been approved by the thesis committee as satisfactorily completing the thesis requirements for the degree Master of Science.
(for example, Master of Science)

Marius Zimand	Marius Zimand	4/28/2017
Chairperson, Thesis Committee Signature	Type Name	Date
B Blair Taylor	B. Blair Taylor	4/28/2017
Committee Member Signature	Type Name	Date
Michael P. McGuire	Michael P. McGuire	4/28/2017
Committee Member Signature	Type Name	Date
Committee Member Signature	Type Name	Date
Committee Member Signature	Type Name	Date
Janet V Dehany	Janet Dehany	5-7-17
Dean of Graduate Studies	Type Name	Date

Acknowledgements

I would like to thank Dr. Marius Zimand for taking me on as a student, Dr. Blair Taylor and Dr. Siddharth Kaza for paying for my graduate studies, Dr. David Vanko for paying for my undergraduate studies and convincing me to come to Towson in the first place, and last but not least Dr. Jia-An Yan for teaching me linear algebra and introducing me to information theory.

Abstract

Information Reconciliation for Erasure Channels

Kyle Sherbert

Say Alice and Bob have n-bit strings which are identical except that Bob's contains t "erasures": the corresponding bit may be either zero or one. Alice will send Bob a message he can use to fill in his erased bits. The Slepian-Wolf bound suggests Alice may send as few as t bits, even when Alice does not know which t bits of are erased. We present several protocols acheiving this bound with varying degrees of success. A binary, linear, and deterministic protocol exists only for $t \in \{0, 1, n-1, n\}$. A nonbinary protocol exists for any $t \leq n \leq 2^m - 1$, but requires that erasures occur in m-bit blocks. A nonlinear protocol seems promising, but the theory to prove it in general is difficult to explore. Finally, a probabilistic protocol has no parameter constraints, but has suboptimal communication complexity based on an additional parameter c, and will fail with an error-rate ϵ dependent on c and t.

Table of Contents

List of Protocols

List of Figures

Glossary

Symbols not listed here are only used locally where they are defined. All indices (usually denoted by i, j, or k) begin numbering from 1.

α - the generator of a finite field, or a primitive nth root of a polynomial

c - the number of symbols in a redundancy s, or a constant

γ - an element of a code (ie. a codeword)

d - the minimum Hamming distance of a set of vectors

δ - the number of extra bits sent, $c - t$

Δ - the effective excess communication complexity

e - an error-vector

ϵ - the failure-rate of a protocol

$\mathbf{F_q}$ - a finite field of order q

$\mathbf{F_q^n}$ - a string of n elements of finite field of order q

h - an entropy

$\mathbf{H}$ - a parity-check matrix

$|\mathbf{M}|$ - the determinant of a square matrix $\mathbf{M}$

m - the number of bits in a symbol of x

μ - number of variables of an RM code polynomial

n - the number of symbols in a message x

p - a probability, or the characteristic of a finite field

q - the size of an alphabet, the order of a finite field (generally a power of 2)

ρ - degree of an RM code polynomial

$\mathbf{R}$ - a random number generator

$\mathrm{rk}\,[\mathbf{M}]$ - the rank of a matrix $\mathbf{M}$

s - Alice's redundancy, or the syndrome of a given string x such that $s = \mathbf{H} \cdot x$

t - the number of erasures

x - Alice's original string, or a coefficient vector

y - Bob's received string, with erasures

X, Y - random variables for X and Y

χ - a variable over which polynomials are written - does not always have physical meaning

1 Introduction

In this thesis, we explore the communication problem of erasure reconciliation over a noiseless channel.

In communication theory, we consider the system where an **information source** generates a **message**, which is transformed into a **signal** by a **transmitter**, which then sends the signal over a **channel** to a **receiver**, which converts it back into the original message. In discrete systems, which we are concerned with, the signal is a series of **symbols** drawn from a finite set, most often $\{0, 1\}$. For the sake of clarity, we will name our transmitter "Alice" and our receiver "Bob".

The quintessential problem in communication theory is to develop a protocol which allows Bob to reconstruct Alice's original message, minimizing **communication complexity**, which usually means sending as few bits as possible. If the channel is **noiseless**, the signal Alice sends is identical to the signal Bob receives. Bob can recover the original message by performing the inverse of the transformation Alice used to generate the signal. If the channel is **noisy**, the signal Alice sends experiences random alterations before Bob receives it, and message recovery is more complicated.

For example, in a "symmetric channel", each symbol is exchanged for another at a certain probability. That is, a 0 might be replaced with a 1, and a 1 might be replaced with a 0. Bob must determine which bits were flipped before he can perform his inverse-transformation. In the "erasure channel", each symbol is erased

at a certain probability. That is, both 0 and 1 might be replaced with an X. Bob can see very plainly which bits were erased, but he must determine what they were originally before he can perform his inverse-transformation. The erasure-channel is of interest to us in this thesis.

Perhaps the best known communication scenario is that of **Forward Error Correction (FEC)**, in which Alice sends a signal with redundant bits - that is, more bits than are necessary to represent the message. This way, Bob can theoretically have enough information in his partially erased signal to extract the message.

In this thesis, we instead consider a variant of the **reconciliation** scenario, where Alice sends the message untransformed through the erasure channel, and she is then allowed a second transmission over a noiseless channel, omnisciently aware of how many erasures occurred. We will call this second transmission the **redundancy**. Recall that we would like to minimize communication complexity - Alice could resend the entire message, but Bob should need far fewer bits in order to reconcile his partially-erased version.

We will explore several protocols accomplishing this task, present some relevant theoretical proofs, and discuss the benefits and disadvantages of each. We also provide an interactive application implementing each protocol.

2 Background and Literature Review

2.1 Background

This section describes some of the mathematical and information theoretical concepts necessary to understanding the main work of the thesis.

2.1.1 Finite Fields

In this section, we briefly describe the theory of finite fields, as can be found in a number or coding theory textbook such as [1]. A **field** is a set $\mathbf{F}$ combined with two commutative and associative operations (denoted $+$, or addition, and $\cdot$, or multiplication) that take any two elements of $\mathbf{F}$ to produce a third. In addition, a field has the following properties:

1. $\mathbf{F}$ has an additive identity, called 0: such that $\forall a \in \mathbf{F}, a + 0 = a$
2. $\mathbf{F}$ has a multiplicative identity, called 1, such that $\forall a \in \mathbf{F}, a \cdot 1 = a$
3. All elements $a \in \mathbf{F}$ have an additive inverse, called $-a$, such that $a + -a = 0$
4. All elements $a \in \mathbf{F}$ except 0 have a multiplicative inverse, called a^{-1}, such that $a \cdot a^{-1} = 1$
5. Multiplication is distributive over addition, such that $\forall a, b, c \in \mathbf{F}, a \cdot (b + c) = a \cdot b + a \cdot c$

Finite fields are fields where $\mathbf{F}$ has a finite number of elements. The number of elements q is called the **order** of the finite field, and a finite field is referred to by $\mathbf{F}_q$. We use the notation $\mathbf{F}_q^n$ to represent an n-length string of $\mathbf{F}_q$ elements, which can be thought of more formally as a point in a finite n-dimensional space.

Finite fields are also called Galois fields, named for the mathematician who showed that the finite field GF(q) exists if and only if $q = p^m$, where p is a prime number (generally 2). The prime number p also serves as the **characteristic** of the finite field, which is the number of times an element can be added to itself before the result is guaranteed to be 0.

Galois also showed that all finite fields with q elements are isomorphic, meaning no matter what you happen to call each element, the fundamental relationships are all the same. This allows us to standardize our represention of each finite field $\mathbf{F}_q$. When q is prime (so $m = 1$ and $q = p$), $\mathbf{F}_q$ is arithmetic in $\mathbf{Z}_q$, that is, the set of integers modulo q. Addition and multiplication function exactly as we are accustomed to, save that we must take the result modulo q. Inverses are not as we are accustomed, but they are guaranteed to exist and can be calculated efficiently.

When q is not prime, $\mathbf{F}_q$ is arithmetic in $\mathbf{Z}_p[\chi]$ mod $P(\chi)$, that is, the extension of integers modulo p over χ, modulo $P(\chi)$. The **extension** of $\mathbf{Z}_p$ over χ can be thought of as a polynomial of χ, where each coefficient is in the field. $P(\chi)$ can be any irreducible polynomial of degree m. Addition and multiplication function exactly as we are accustomed to with polynomial arithmetic, save that each term's coefficient reduces to its residue modulo q, and we must take the final result modulo $P(\chi)$. A polynomial's additive inverse is simply the same polynomial with each coefficient replaced by its additive inverse in $\mathbf{Z}_p$. Multiplicative inverses are guarateed to exist and can be calculated efficiently.

All finite fields have at least one **generator** α, which is an element with the

following properties:

$$\begin{aligned} \alpha^i = 1 \quad &\iff \quad i \bmod (q-1) = 0 \\ \alpha^i = \alpha^j \quad &\iff \quad i \equiv j \mod q-1 \end{aligned} \tag{1}$$

It is called a generator because every element of the finite field except 0 can be generated by raising α to some power. The set $\mathbf{F} = \{0, \alpha^i\}$ for each $i \in \{1..q-1\}$.

The simplest finite field is $\mathbf{F}_2$, which consists only of the elements 0 and 1. Addition is equivalent to the XOR operation, and multiplication is equivalent to the AND operation. We will call a protocol *binary* if it operates entirely within $\mathbf{F}_2$.

2.1.2 Linear Algebra

In this section, we briefly describe some key tools of linear algebra, as may be found in mathematics or in this case physics textbooks on the subject (see [2] and [3]). Linear algebra is the algebra of **matrices** and **vectors**. An $(a \times b)$ matrix $\mathbf{M}$ over finite field $\mathbf{F}$ is a two-dimensional array of elements of $\mathbf{F}$, with a rows and b columns. We will use the notation $\mathbf{M}_{ij}$ to represent the (scalar) element in the ith row and jth column of $\mathbf{M}$.[1]

An n-vector is a one-dimensional array of length n. It may be arranged as a row (a $1 \times n$ matrix) or a column (an $n \times 1$ matrix). We will use the notation v_i to represent the (scalar) ith element of v. When we refer to n-length strings throughout this thesis, we will assume they are in the form of a column vector.

The transpose of a matrix $\mathbf{M}$, represented by $\mathbf{M}^T$, is the same matrix with the columns and rows swapped. Thus, if $\mathbf{M}$ is an $(a \times b)$ matrix, then $\mathbf{M}^T$ is a $(b \times a)$ matrix. Each element of $\mathbf{M}^T$ is given by $\mathbf{M}^T_{ij} = \mathbf{M}_{ji}$. The transpose of a column

[1] Unless otherwise noted, we will use indices starting from 1.

vector is a row vector and vice versa.

The product of a scalar c and a matrix $\mathbf{M}$, denoted $\mathbf{\Omega} = c\mathbf{M}$, always exists and has the same dimensions as $\mathbf{M}$. Each element of $\mathbf{\Omega}$ is given by $\mathbf{\Omega}_{ij} = c\mathbf{M}_{ij}$.

The dot product of two vectors $z = v \cdot w$ exists if both vectors have the same length n, the first vector v is a row vector and the second vector w is a column vector. The result z is a scalar equal to the following sum:

$$z = \sum_{i=1}^{n} v_i \cdot w_i \tag{2}$$

The sum of two matrices $\mathbf{\Omega} = \mathbf{M} + \mathbf{K}$ exists if both matrices have the same dimensions, ie. if $a_M = a_K$ and $b_M = b_K$. $\mathbf{\Omega}$ also has the same dimensions. Each element of $\mathbf{\Omega}$ is given by $\mathbf{\Omega}_{ij} = \mathbf{M}_{ij} + \mathbf{K}_{ij}$.

The product of two matrices $\mathbf{\Omega} = \mathbf{MK}$ exists if the inner dimensions of the matrices match, ie. if $b_M = a_K$. $\mathbf{\Omega}$ inherits the outer dimensions, so that it is an $(a_M \times b_K)$ matrix. Each element of $\mathbf{\Omega}$ is given by the dot product $\mathbf{\Omega}_{ij} = \mathbf{M}_{i*} \cdot \mathbf{K}_{*j}$ where $\mathbf{M}_{i*}$ is the ith row of $\mathbf{M}$ and $\mathbf{K}_{*j}$ is the jth column of $\mathbf{K}$.

The **span** of a collection of n-vectors is the set of all vectors which can be produced by a linear combination of each vector in the collection. That is, w is in the span of $\{v_1, v_2, \dots\}$ if there exists a set of constants $\{c_1, c_2 \dots\}$ such that:

$$w = \sum c_i v_i \tag{3}$$

A set of vectors is **linearly independent** if no vector in the set is in the span of the collection of remaining vectors. Note that when $c_1 = c_2 = \ldots = 0$, the vector w is the zero vector, $[0\ 0\ \cdots\ 0]$. Thus, the zero vector is in every span, and any collection including the zero-vector is *not* linearly independent.

The rank of a matrix rk[$\mathbf{M}$] is the number of dimensions needed to represent the span of the rows or columns of $\mathbf{M}$. Alternatively, the rank is the size of the largest set of linearly independent rows or columns in the matrix $\mathbf{M}$. The span of rows and the span of columns of any matrix always have the same dimension, so either can be used to calculate rank. Note that rank can never be greater than the number of rows or columns in $\mathbf{M}$.

The equation $\mathbf{M}x = z$, where x is an unknown b-length column vector and z is a known a-length column vector, is equivalent to the following linear system of equations, with b variables and a equations:

$$\begin{aligned} M_{11}x_1 + M_{12}x_2 + \ldots &= z_1 \\ M_{21}x_1 + M_{22}x_2 + \ldots &= z_2 \\ &\vdots \end{aligned}$$

Let $\mathbf{A} = [\mathbf{M}\ z]$, the "augmented" matrix formed by concatenating z to the right of $\mathbf{M}$. Whether or not this system can be solved for x depends on the relation between its length a and the rank of $\mathbf{A}$):

1. rk[$\mathbf{M}$] $<$ rk[$\mathbf{A}$] - there are no solutions for x (equations are inherently contradictory)

2. rk[$\mathbf{M}$] = rk[$\mathbf{A}$] = a - there is a unique solution for x

3. rk[$\mathbf{M}$] = rk[$\mathbf{A}$] $> a$ - there are several solutions for x

In case 1, when there are no solutions for x, the system is said to be **singular**. In case 2, where there is a unique solution for x, linear algebra theory provides several algorithms for efficiently solving the system. We will not describe them here, although we will make use of them throughout the thesis. In case 3, where there are multiple solutions for x, we cannot solve the system without imposing additional constraints.

The **determinant** of a square $(n \times n)$ matrix, represented by $|\mathbf{M}|$, is another way of seeing if a system is solvable. We will not describe here how to calculate it, but it can be done efficiently, on the order of $\Theta(n^3)$. If $|\mathbf{M}| \neq 0$, its rank is exactly n, and a system of equations with $\mathbf{M}$ as its coefficient matrix must have a unique solution. If $|\mathbf{M}| = 0$, the matrix itself is said to be singular.

2.1.3 Information Entropy

In this section, we formalize the amount of information generated by some information source, as defined by [4] and described in an information theory textbook such as [5]. In general, this is accomplished with information **entropy**. Consider the set of all possible messages $\mathcal{M}$. Some elements of $\mathcal{M}$ may be produced by our information source more frequently: let message $x_i \in \mathcal{M}$ appear with probability p_i, so that p is the probability distribution describing how likely each message appears. Intuitively, more common messages are less informative: if, for instance, p is a unary distribution around p_1, we should be completely unsurprised to see m_1, no matter how many times it appears. We define the entropy h of the information source as follows:

$$h = -\sum_{i=0}^{|\mathcal{M}|} p_i \log p_i \tag{4}$$

When $\mathcal{M}$ is continous, we instead use differential entropy, which replaces the sum with an integral. The units for entropy are determined by the base of the logarithm.

In this thesis, we will only discuss messages consisting of a string of n symbols, meaning we are dealing with a discrete $\mathcal{M}$. Furthermore, we will only discuss symbols which can be represented by an integer number of bits m. For example, when we are discussing strings of bits, $m = 1$, and when we are discussing strings of bytes, $m = 8$. Thus, the natural base to use in our logarithm is 2, so our entropy unit is called a

Shannon, in honor of Claude Shannon, the father of information theory.[2] Finally, we will assume that all messages are equally likely, so $p_i = |\mathcal{M}|^{-1}$ and h simplifies to:

$$h = \log_2 |\mathcal{M}| = \log_2 2^{mn} = mn \tag{5}$$

which is simply the number of bits in our message. You may assume $m = 1$ except where otherwise noted.

Also of interest are **conditional** and **joint** entropies, which are derived directly from the analogous concepts in probability. Let X be a random variable representing the signal Alice sends, and Y be a random variable representing the signal Bob receives. Exact quantity depends on the type of noise in the channel, but the following relation is true:

$$h(X, Y) = h(X) + h(Y|X) = h(Y) + h(X|Y) \tag{6}$$

For example, if Alice transmits a binary ($m = 1$) message directly, $h(X) = n$. If the channel introduces exactly 1 erasure at a random location, then Bob may receive n different messages for a given x. Thus, the conditional entropy $h(Y|X) = \log_2 n$. For a given y containing one erasure, the erasure could be replaced by one of two symbols. Thus, the conditional entropy $h(X|Y) = \log_2 2 = 1$. Using the above relation, $h(Y) = n + \log_2 n - 1$.

In general, if Alice sends n m-bit symbols over a channel introducing exactly t

[2] "Shannons" are also simply called "bits", but that may get confusing, since not every bit we meet will contain a "bit" of information.

erasures, we have the following entropies:

$$\begin{aligned} h(X) &= mn \\ h(Y|X) &= \log_2 \tbinom{n}{t} \\ h(X|Y) &= mt \\ h(Y) &= m(n-t) + \log_2 \tbinom{n}{t} \\ h(X,Y) &= mn + \log_2 \tbinom{n}{t} \end{aligned} \tag{7}$$

2.1.4 Channel Rate

In this section, we formalize the amount of information that can be sent along a channel, ie the **channel capacity**, designated $\mathcal{C}$. Normally this is defined as the most amount of information that can be transmitted per second [4, 5], because engineers care about how long things take. As theorists, we shall assume every bit we send over the channel takes the same amount of time, and therefore our channel capacity is only a measure of how many bits must be sent in order to transmit a Shannon of information.

The capacity of a noiseless channel is simple: in order to send one Shannon of information, we must send one bit. Thus, we say $\mathcal{C} = 1$.

To determine how noise affects the capacity of a channel, we must consider how much information is lost when an information-rich ($h(X) = mn$) signal is transmitted through that channel. This turns out to be the conditional entropy $h(X|Y)$, which was discussed in Section 2.1.3. In order to send mn Shannons, we must send an extra $h(X|Y)$ bits to make up for the information lost to noise.

For an erasure channel which erases exactly t symbols, we must send an extra mt

bits (Equation 7). Thus, our capacity is:

$$\mathcal{C} = 1 + \frac{t}{n} \tag{8}$$

The concept of channel capacity is closely related to the concept of **rate**, designated $\mathcal{R}$, which is the number of bits we actually do send, in order to transmit a Shannon of information. A communication protocol is **optimal** if its rate is equal to the capacity of whatever channel it is designed to communicate over, or at least that it does so in the limit where $n \to \infty$. Communication complexity is the rate $\mathcal{R}$ times the number of bits transmitted.

Consider the reconciliation scenario described in Section 1 where Alice sends Bob a message which is partially erased, and then she sends a redundancy s which Bob uses to reconcile his original message. Let c be the length of s, the number of symbols Alice sends in the second part of the protocol (and as usual n is the number of symbols in the message). Then the rate of the protocol is:

$$\mathcal{R} = 1 + \frac{c}{n} \tag{9}$$

which is optimal on an erasure channel when $c = t$. Technically this is not quite accurate, since a part of the protocol takes place over a noiseless channel, but we shall consider it "close enough".

Some protocols are **probabilistic** (as opposed to **deterministic**), meaning Alice fails to transmit any information with failure-rate ϵ. We shall discuss one such protocol, where Alice's redundancy may or may not allow Bob to reconcile his partially-erased string. In order to compare such a protocol fairly, let us assume that Alice omnisciently knows when the protocol has failed, so that she knows to send another

c bits. Thus, the expected number of bits she ultimately sends E_c is equal to:

$$E_c = c + c\epsilon + c\epsilon^2 + c\epsilon^3 + \cdots = c\sum_{i=0}^{\infty} \epsilon^i \tag{10}$$

which reduces by calculus to $E_c = c(1-\epsilon)^{-1}$. Thus, the effective rate $\mathcal{R}_E$ is:

$$\mathcal{R}_E = 1 + \frac{c}{n}(1-\epsilon)^{-1} \tag{11}$$

2.1.5 Random Number Generators

Any probabilistic protocol relies on randomness. This randomness comes from making decisions based on numbers drawn from a **random source**, or **generator**, denoted **R**. A **true** random source is one based on physical processes such as light propagation or thermal vibration. A **pseudorandom** source is one based on deterministic algorithms. A **seed** is the initial parameterization of a pseudorandom source, such that two generators seeded with the same value will generate the same numbers.

If all parties in the protocol (ie. Alice and Bob) have access to the same random source, it is called **shared randomness**. In general, it is difficult for distantly removed parties to reliably share a source of true randomness, so we will assume any protocol calling for shared randomness requires a pseudorandom generator and a common seed. In Section 3.5, we will use a protocol requiring shared randomness. We will discuss several pseudorandom generators then.

A pseudorandom source is called **linear** if each number it produces can be calculated as a function of the previous number it produced. These are not generally considered *good* random generators, without significant post-processing. For one thing, anyone can easily calculate exactly what numbers will follow, if one knows the algorithm to do so. For another, in order to generate the same number twice, one

must necessarily repeat the entire sequence. The number of unique numbers a linear pseudorandom generator produces before repeating is called its **period**. Despite their disadvantages, they are very easy to use and will suit our purposes fine as long as the period is long enough.

2.1.6 Linear Codes

In this section, we briefly describe the theory of linear codes, which will play a major role in our discussion. This information is available in a coding theory textbook such as [6] or [1]. Consider the forward error correction scenario described in Section 1. Alice must transform a message into some longer **codeword** that she transmits to Bob, and Bob must then decode it in spite of any errors it incurred along the way. A **code** is the set of all possible codewords for a given communication protocol. A **linear code** is a code which can be produced from a linear transformation of all possible messages, making for very simple encryption and an easy-to-understand strategy for decryption. We will refer to such protocols as linear also.

Many linear codes have associated with them a $(k \times n)$ **generator matrix G**, which relates a given message ξ to its codeword x such that $\xi^T \cdot \mathbf{G} = x$. In fact, the code is the span of **G**, ie the set of all possible linear combinations of the rows of **G**. The message ξ serves to select which rows to combine to produce the codeword x.

All linear codes also have associated with them a so-called **parity-check matrix H**, with the property that $\mathbf{HG} = \mathbf{0}$. This property also means that, for any n-length string r, $\mathbf{H} \cdot r = \mathbf{0}$ if and only if r is in the code. We define the **syndrome** s of r to be the following vector product:

$$s = \mathbf{H} \cdot r \tag{12}$$

The syndrome serves to identify the **cosets** of r, which is the code offset by an "error"

vector e. A **perfect** code is one in which every n-length string r belongs to exactly one coset, meaning that the syndrome is enough to map r to the "closest" codeword. Closeness is given by the concept of Hamming **distance**, which is the number of bits in which two strings differ.

Error-detection protocols basically amount to Bob finding the syndrome of his received signal r. If it is anything but $\mathbf{0}$, he knows he has an error. Error-correction protocols basically amount to Bob exploiting the syndrome to find the error e, which in combination with r produces the codeword γ. That is, $r = \gamma + e$, where e is given by s.

The smallest distance you can find between two codewords for a given code is called the minimum Hamming distance, d. All linear codes are subject to the **Singleton bound**:

$$d \leq n - k + 1 \tag{13}$$

A code is called **Max Distance Separable (MDS)** if this is an equality, so that $d = n - k + 1$. One can find d by finding the codeword with the least **weight**, ie the least number of non-zero elements. It can be shown that for a linear code, that weight is d.

A code with codeword-length n, message-length k, minimum Hamming distance d, and symbol alphabet size q is called an $(n, k, d)_q$ code. If the code is linear, use brackets: $[n, k, d]_q$

Consider codes designed for a symmetric channel, meaning any bit has a certain probability of being flipped. A code is guaranteed to be able to detect up to $d - 1$ errors. If there are any more than that, one codeword may be transformed entirely into another and evade detection. The same code is guaranteed to be able to correct up to $\frac{d-1}{2}$ errors. Any more than that and r may be equally close to two codewords,

and Bob has no way to know which was sent.

Now consider the erasure channel: if r contains any X's, then Bob can detect there have been erasures. Given the same code from the previous paragraph, Bob can correct up to $d-1$ erasures. Any more than that and r may be equally close to two codewords. In general, any given code can correct twice as many erasures as it can errors.

2.2 Literature Review

The reconciliation problem is closely related to that of **distributed coding**, first explored by Slepian and Wolf [7]. In distributed coding, the goal is for a single receiver to decode information from two sources X and Y. Each message is encoded and transmitted independently, at which point the receiver can decode both messages together.

Say transmission is over a noiseless channel. If the sources are entirely independent, each message should be transmitted directly, resulting in a total communication complexity of $2n$. Each transmission has rate $\mathcal{R} = 1$. Slepian and Wolf showed that if the two sources are correlated, meaning that their joint entropy $h(X,Y) < h(X) + h(Y)$, the number of bits c for each transmission are bounded as follows:

$$\begin{aligned} c_X &\geq h(X|Y) \\ c_Y &\geq h(Y|X) \\ c_X + c_Y &\geq h(X,Y) \end{aligned} \tag{14}$$

Bob from our reconciliation problem can be thought of as both the receiver *and the second transmitter* of a distributed coding problem. Essentially, the transmitter at Y "sends" its entire message directly, thus having $c_Y = mn$. Our protocol is only interested in c_X, so we'll drop the subscript. Its theoretical bound is

$c \geq \max(h(X|Y), h(X,Y) - mn)$. Using our entropies for the erasure channel from Equation 7, we find the latter argument is always negative, and therefore our communication complexity is:

$$c \geq mt \tag{15}$$

This limit makes intuitive sense: there are precisely mt bits that Bob does not know, so Alice must send him at least mt bits. The surprising bit is that this is theoretically achievable even when Alice does not know *which* bits Bob does not know.

Practical coding schemes acheiving the bounds Slepian and Wolf proved in 1973 were not realized for nearly three decades, but in 1999, Pradhan and Ramachandran [8] proposed the idea of using syndromes to acheive the reconciliation case. [8] goes far beyond what we will explore in this thesis, immediately generalizing to continuous signals and Gaussian channels.

Gehrig and Dragotti in [9] keep to binary n-length messages, but their protocol generalizes to a true distributed coding problem with any number of sources. Here we present a diluted version of their argument, as applied to the reconciliation case. For now, assume X and Y are correlated by a symmetric channel; we will adapt the argument to erasures in Section 3.2.

Let Alice have the n-length string x, and Bob have string $y = x + e$, where e is an error vector with no more than t non-zero elements, indicating exactly where each error occurred. Now consider a binary FEC linear code $[n, k, 2t+1]_q$ for some k, with parity-check matrix $\mathbf{H}$. Neither y nor x are necessarily in the code, but they each have a syndrome s_x, s_y. Let Alice transmit s_x to Bob. Now Bob subtracts s_x from $\mathbf{H}y$ to find the syndrome of e:

$$\mathbf{H}y - s_x = \mathbf{H}y - \mathbf{H}x = \mathbf{H}(y - x) = \mathbf{H}e \tag{16}$$

Bob may then reconstruct e using standard error decoding techniques, and reconstruct x by $x = y - e$. Depending on the code used (and therefore the properties of $\mathbf{H}$) this may not be optimal, and may even be probabilistic, but the reconciliation has been performed.

Note that Bob could have written the equation $\mathbf{H}x = s_x$ and solved using standard linear algebra techniques described in Section 2.1.2, which are usually much simpler. This doesn't work because the rank of $\mathbf{H}$ is too high: the system of equations has several solutions which produce the same syndrome, and Bob can't know which is correct without accounting for y as described above.

[10] explores the theoretical bounds of the combinatorial variant of distributed coding, when sources may differ by *up to* t bits, rather than *exactly* t bits as we have explored. [11] follows it with a probabilistic protocol which is optimal. It is especially nice because a variant of the protocol exists in which all parties may use their own random number sources (rather than a shared source), albeit at significant computational expense. That said, remember that "optimal" just means as n approaches infinity. This protocol converges very slowly, so as a practical matter, it isn't very practical.

[12] provides a protocol with communication complexity on the order of $t(\log^2 t + \log n)$, which is not altogher impressive except that t is in fact **edit distance**, which accounts for erasures in addition to **synchronization** errors, where the string Bob receives may have bits inserted or deleted. Synchronization errors are studied in [13], [14], and [15].

We have thus far assumed that both Alice and Bob are omnisciently aware of how closely related their strings x and y are. That is, we assume Alice and Bob simply *know* that y incurs t errors during transmission. [16], [17] and [18] discuss incremental distributed coding, where no such assumption is made. These protocols are **variable**

rate, meaning that their communication complexity adjusts to the actual number of errors. However, they depend on back-and-forth communication between transmitter and receiver, whereas the problem we are studying requires the only communication be a single signal from Alice to Bob.

Ideally, we would like to apply the results of [8], [9], and [10] to the case when y incurs erasures, rather than bit-flip errors, during transmission. It seems that this is a simple matter of utilizing a t-erasure FEC code in the same way [9] utilized t-error FEC codes.

There is not an extensive amount of literature dedicated to erasure codes, presumably because, as stated in 2.1.6, any linear code can correct twice as many erasures as errors. In 2001, [19] found that Low-Density Parity Check (LDPC) codes can optimally correct erasures when they occur at a probability p (as opposed to a fixed frequency t). More recently, [20] proved theoretically that Reed-Müller codes also achieve capacity on erasure channels. In fact, the result of [20] is particularly exciting because the proof is easily extended to any code, and to any channel, exhibiting certain properties.

The work presented in this thesis, though completed independently, is not groundbreaking in any way - the problem it studies is much simpler than many of those considered in these works. We shall closely examine linear codes as explored by [9], probabilistic codes as explored by [10], and Reed-Müller codes as explored by [20]. However, we do so not to innovate new and practical methods of network communication, but instead simply to explore communication theory, in as simple and understandable a way as possible.

3 Results

3.1 Overview

Our scenario is as follows: Alice has a string x, consisting of n m-bit symbols. Bob has a string y, which is identical to x except that t symbols have been erased. We must design a protocol for Alice to construct and transmit a redundancy s, which Bob receives and uses to reconcile the erasures in y, producing x'. We consider the protocol successful if $x = x'$.

We consider a protocol to be *ideal* if it is binary, linear, and deterministic. Binary protocols are easier to understand than those existing in higher-order finite fields, and are far easier to implement, since all operations can be treated as logical ones. Linear protocols are those in which the most involved step is solving a system of equations with linear algebraic techniques, which is easy to implement efficiently. Deterministic protocols are preferred to probabilistic ones, simply because, if all else is equal, we'd like to have the right answer.

Section 3.2 presents very simple, specific scenarios to illustrate how reconciliation can be achieved. We then prove that those protocols are the only "ideal" optimal solutions, and that they are only usable for a narrow class of parameters, when $t \leq 1$ or $n - t \leq 1$.

Section 3.3 presents a protocol which utilizes Reed-Solomon (RS) linear codes. This solution foregoes a binary alphabet, but provides a linear and deterministic

solution up to a certain word-size, based on the symbol-size: $n \leq 2^m - 1$.

Section 3.4 presents a protocol which utilizes theory from Reed-Müller (RM) codes, which are based on high-order (ie. nonlinear) polynomials. It provides a binary and deterministic solution, but we cannot use linear algebraic techniques to solve the problem efficiently. It is therefore more difficult to implement, less efficient, and still has some serious constraints on parameters, although less so than RS codes.

Section 3.5 presents a simple and easy-to-implement probabilistic protocol. This solution, being probabilistic, will fail to produce the correct answer every time, but it is binary and linear. The protocol assumes Alice and Bob have a source of shared randomness, and in fact may even work better with certain pseudo-random generators. We prove some theoretical constraints on its rate, and we also present computational simulations verifying the theory.

3.2 MDS Theory: an Ideal Solution

Throughout this section, all discussion is for $\mathbf{F}_2$. Assume $m = 1$.

3.2.1 Parity Bit: Solution for t=1

Before we get into any more theory, let's see an example of a protocol that actually solves the erasure reconciliation problem, albeit for a very simple case: $t = 1$. Bob's string y has exactly 1 erased symbol. With a single erasure, the optimal redundancy Alice can pick should have a single bit. Which bit should it be?

Alice simply lets s be the **parity** of x, which is 1 if there are an even number of 1's and 0 otherwise. For his part, Bob determines the parity of y. If it equals s, then he knows his missing bit must not change the parity, and is therefore 0. If it *doesn't* equal s, it must be 1.

3.2.2 Linear Erasure Reconciliation

Now that we have the idea, let's consider the general case: Bob's string y has exactly t erasures. Alice should pick a string s that has t bits. To work this out, we will adapt Gehrig and Dragotti's protocol in [9] for erasures.

Previously (Equation 16), we had the following equation (substituting s in that section for the s we are using now):

$$\mathbf{H}e = \mathbf{H}y - s \tag{17}$$

Bob had to rely on error decoding techniques to find the actual error locations in e. But now he already *knows* the locations - they are wherever he has an X. Moreover, he is already completely certain that every bit without an X is correct - each bit serves as an additional constraint which limits the possible solutions to the system of

equations.

In practice, Bob can apply those constraints by simply dropping all those columns of $\mathbf{H}$ which correspond to bits he already knows, producing a reduced matrix $\mathbf{H}'$ whose rank is certainly not too high. As long as the remaining t columns are linearly independent, we can solve for the missing bits e' using standard linear algebra techniques. The full error vector e is e' padded with 0's in all the previously known locations. Then, as before:

$$x = y - e \tag{18}$$

There is a complication: the string y has erasures in it. How do we calculate its syndrome $\mathbf{H}y$? The symbol X is not in $\mathbf{F}_2$; we cannot do arithmetic with it. The solution is simple: guess its actual value. Really: just go through and replace each X with either a 0 or a 1. If you happen to guess right, then when you solve for that bit in e, it'll be 0. If you guess wrong, it'll be 1. Either way the problem gets sorted out when you do that last step.

All we must do is find a matrix $\mathbf{H}$ with two properties:

1. $\mathbf{H}$ has t rows and n columns, so it produces a t-bit redundancy from the n-length string x.

2. Any selection of t columns from $\mathbf{H}$ are linearly independent, so that the resulting system of equations is never singular, no matter which bits of y are erased.

When $t = 1$ as in the simple case from Section 3.2.1, $\mathbf{H}$ is just the vector $H = [1\ 1\ \cdots\ 1]$. The parity of a string x is in fact the dot product $H \cdot x$ in binary arithmetic.

As another example, let $t = n - 1$. The matrix we use now should look something

like:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & & 0 & 0 & 1 \\ 0 & 1 & & 0 & 0 & 1 \\ & & \ddots & & & \vdots \\ 0 & 0 & & 1 & 0 & 1 \\ 0 & 0 & & 0 & 1 & 1 \end{bmatrix} \tag{19}$$

As a final example, let $t = n$: Bob has no information on x at all. Alice should probably just send the entire string. Indeed, the identity matrix itself satisfies our criteria.

3.2.3 Failed Alternative: The Hamming Code

To highlight the importance of the second property, let us consider the oft-lauded Hamming $[7, 4, 3]_2$ code. This code is often used as an example of how to correct single bit-flip errors for $n = 7$. The parity-check matrix for this code is given by:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{20}$$

This matrix has three rows, so the redundancy s will be three bits.

Let us say Bob has received y =X001X1X, and s =100. First, Bob arbitrarily replaces all X's in y with 0's, producing y' =0001010. Next, he calculates $\mathbf{H}y' - s$, resulting in the column vector $z = [0\ 0\ 1]^T$. Next, he drops all redundant columns of $\mathbf{H}$, leaving:

$$\mathbf{H}' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Protocol 1: Ideal Protocol

Setup Alice has a string x, consisting of n bits. Bob has a string y, which is identical to x except that t bits have been erased.

Parameters (agreed upon before communication)

H - a $(t \times n)$ matrix over $\mathbf{F}_2$, with the property that any selection of t columns are linearly independent.

Steps

1. Alice calculates the redundancy $s = \mathbf{H}x$.
2. Alice transmits s to Bob.
3. Bob calculates the string y' as y with each erasure replaced with a zero.
4. Bob calculates $z = \mathbf{H}y' - s$.
5. Bob calculates the reduced matrix $\mathbf{H}'$ by removing each column of $\mathbf{H}$ where y did not have an erasure.
6. Bob solves the equation $\mathbf{H}'e' = z$ for e, using linear algebraic techniques.
7. Bob calculates the error vector e by padding e' with zeros wherever y did not have an erasure.
8. Bob calculates Alice's string $x = y - e$.

Now he must solve $\mathbf{H}'e = z$, which his handy linear algebraic calculator does easily: $e' = [0\ 0\ 1]^T$, or e =0000001. Finally, he finds $x = y - e$ =0001011.

No problem. But what if the sixth bit had been erased in place of the seventh? Now y =X001XX1. Bob completes each step in the exact same way until he comes to his reduced matrix:

$$\mathbf{H}' = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

There is a row containing only 0, a dead giveaway that the system is singular: no solution to e' exists.

H was able to correct certain $t = 3$ erasure patterns, but not all. It is not difficult to work out that it *can* correct all $t = 2$ erasure patterns, which is consistent with our remark in Section 2.1.6 that any error-correcting code can correct twice as many erasures. However, it is *not* optimal. Evidently even a *perfect* code like Hamming's fails to provide us with the matrix we need.

3.2.4 MDS: A Necessary Condition

In this section we prove the following theorem:

Theorem 1. *If a $(t \times n)$ matrix* $\mathbf{H}$ *with elements in* $\mathbf{F}_2$ *has the property that any selection of t columns are linearly independent, then it must span a space whose minimum Hamming distance $d = n - t + 1$.*

Proof. A selection of t columns is *not* linearly independent if one of the rows can be produced by a linear combination of the other rows. Since we are in $\mathbf{F}_2$, this is equivalent to saying that one row is the sum of several others. Adding the duplicated row to this sum produces a zero vector $[0\ 0\ \ldots\ 0]$. If the rows of $\mathbf{H}$ can be combined to

produce a vector containing t zeros, or equivalently, if the span of $\mathbf{H}$ has any elements which contain more than t zeros, there exists a selection of t columns which is not linearly dependent. Thus, in order for $\mathbf{H}$ to have the desired property, the weight of each vector in its span must be at least $z \geq n - t + 1$.

From coding theory (Section 2.1.6), we know that the least weight of an n-vector in a set is equivalent to the minimum Hamming distance d of that set. Thus, $d \geq n-t+1$. But according to the Singleton bound (Equation 13), $d \leq n - t + 1$. For these both to be true, $d = n - t + 1$. □

Since the span of $\mathbf{H}$ has a minimum Hamming distance $d = n - t + 1$, the space is an MDS linear code, and $\mathbf{H}$ is its generator matrix. Binary MDS codes for a given n are well-known - only four sets of parameters for a given n exist: $t = 0$, $t = 1$, $t = n - 1$, and $t = n$. We have already seen all but the first, where reconciliation is not required. For any combination of n and t not in these four, we must forego at least one of our ideals.

3.3 RS Theory: a Nonbinary Solution

In this section, we use use theory from **Reed-Solomon (RS)** codes to perform reconciliation. They happen to be MDS codes, which as shown in 3.2.4 is what we need for a binary optimal linear solution. They are not, however, binary: in fact, a Reed-Solomon code is one in which the string length n and symbol alphabet size q are related by $n = q - 1$. In our examination, we require $q = 2^m$ for some symbol-size m.

3.3.1 Theory

RS codes are a subclass of **Bose-Chaudhuri-Hocquenghem (BCH)** codes, which are themselves a subclass of cyclic codes, which are themselves a subclass of linear codes. We have already seen linear codes described in Section 2.1.6; we will describe the rest in turn here. This information can be found in thorough coding theory textbooks such as [6]. We also relied heavily on lecture notes from [21].

Cyclic Codes Cyclic codes are so named because every cyclic rotation of a codeword is also a codeword. Though they are linear codes, they abandon the notation of linear algebra in favor of high-order polynomials over a variable χ. Note that the value of χ itself has no immediate meaning to the code: we are merely using it as a tool to conveniently perform arithmetic which happens to result in a cyclic code.

Codewords are multiples of a generator polynomial $g(\chi)$, modulo $\chi^n - 1$. That is, $\gamma(\chi) = g(\chi) \cdot P(\chi) \bmod \chi^n - 1$ for some message polynomial $P(\chi)$. The actual n-length string is the sequence of coefficients in $\gamma(\chi)$, so that the ith symbol of γ is the coefficient for χ^i in $\gamma(\chi)$.

The generator polynomial $g(\chi)$ must be a factor of $\chi^n - 1$. If the degree of $g(\chi)$ is s, then the highest degree of a message polynomial is $k = n - s$. In general, the

higher the degree of $g(\chi)$, the more errors it can correct but the fewer messages it can encode. It is related to the parity-check polynomial $h(\chi)$ by $g(\chi) \cdot h(\chi) = \chi^n - 1$. The parity-check polynomial performs the same function a parity-check matrix does: $h(\chi) \cdot \gamma(\chi) = 0$ modulo $\chi^n - 1$ if and only if $\gamma(\chi)$ is a codeword.

BCH Codes BCH codes use a generator polynomial formed using the **primitive nth root** of 1, which we denote by α. It is a (potentially imaginary) number with the following properties:

$$\begin{aligned} \alpha^i = 1 \quad &\iff \quad i \bmod n = 0 \\ \alpha^i = \alpha^j \quad &\iff \quad i \equiv j \mod n \end{aligned} \tag{21}$$

This should look very familiar, since we used the same symbol for generators of a finite field, which have very similar properties (Equation 1). In RS codes, they are one and the same. For now, however, α, like χ, has no concrete representation. Indeed, it may very well be an imaginary number.

Note that α^i for any integer i is a solution to the equation $\chi^n - 1 = 0$, ie. it is a **root** of the polynomial $\chi^n - 1$. There are n unique roots, corresponding to $0 \leq i < n$. By the definition of "root":

$$\chi^n - 1 = \prod_{i=0}^{n-1} (\chi - \alpha^i) \tag{22}$$

Any factor of $\chi^n - 1$ with degree s is a product of s polynomials of the form $\chi - \alpha^i$, which is to say that each of those α^i are a root of the factor. It can be shown (see [6]) that if the factor selected as a generator polynomial $g(\chi)$ has a run of δ consecutive powers of α as roots, the resulting code has Hamming distance $d \geq \delta + 1$.

When designing a BCH code for a given n, one must first select a factor of $\chi^n - 1$

to use as $g(\chi)$. Then one must select an i for which α^i is a root, so that $g(\alpha^i) = 0$. From this, one can algebraically determine which other roots $g(\chi)$ has. The trick is to pick $g(\chi)$ and the power i of its first root such that $g(\chi)$ has as few non-consecutive powers of α as roots. These do nothing to increase δ (and therefore d, and the number of errors that can be corrected), yet they still increase the degree of $g(\chi)$, taking away from the number of messages that can be encoded.

BCH codes have a parity-check polynomial, but it is more common to use an $(\delta \times n)$ parity-check matrix of the form:

$$\mathbf{H}_{ij} = \alpha^{(i+k-2)j} \tag{23}$$

The integer k identifies the first power of α in the longest run of consecutive roots of $g(\chi)$, the length of which is δ.

RS Codes An RS code is defined to be a BCH code where all roots of $g(\chi)$ are consecutive powers of α. That is, an RS code is one whose generating polynomial is given by:

$$g(\chi) = \prod_{i=1}^{d-1} (\chi - \alpha^i) \tag{24}$$

The degree of $g(\chi)$ is $s = n - k = d - 1$, so RS codes are MDS.

To guarantee this $g(\chi)$ is a factor of $\chi^n - 1$, the polynomial must be over elements in a finite field $\mathbf{F}_q$ with order $q = n + 1$. This can be seen intuitively[3] by noting that each coefficient is the sum of several powers of α. In a binary BCH code, each of these coefficients would end up being either 1, as could be shown from $\alpha^n = 1$, or 0, as could be shown from the assumption $g(\alpha^i) = 0$. Constructing RS codes does not involve such an assumption. To allow for any d at all, $\mathbf{F}_q$ should contain α and all its

[3] the opposite of "rigorously"

powers. There are n such powers, and we must also allow for a zero-element. Thus, the order of $\mathbf{F}_q$ should be $n+1$. Comparing Equations 21 and 1, we find α now has a very concrete representation: it is a generator of $\mathbf{F}_q$.

3.3.2 RS Erasure Reconciliation

RS codes are MDS, and therefore we could make use of their generator matrix, which can be constructed from its generator polynomial $g(\chi)$ fairly easily, to solve our system of equations.

However, it turns out that using the parity-check matrix $\mathbf{H}$ is even easier. Note that the k we used in Equation 23 is now 1, and $\delta = d - 1$, which we shall soon see also happens to be t, the number of erasures we can correct. Thus, $\mathbf{H}$ is a $(t \times n)$ matrix with the form:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} \\ & & & \vdots & \\ 1 & \alpha^t & \alpha^{2t} & \cdots & \alpha^{t(n-1)} \end{bmatrix} \tag{25}$$

We will now show that this $\mathbf{H}$ can always be used to solve our reconciliation problem with t erasures using the protocol described in Section 3.2.2. The proof takes advantage of $\mathbf{H}$'s similarity to a **Vandermonde matrix**, which is any matrix of the form:

$$\begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{n-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{n-1} \\ & & & \vdots & \\ 1 & a_n & a_n^2 & \dots & a_n^{n-1} \end{bmatrix} \tag{26}$$

Vandermonde matrices have the special property that, if all a_i are distinct, its deter-

minant is non-zero ([6]). (Equation 21).

Theorem 2. *The reduced matrix* $\mathbf{H}'$ *formed by selecting any* t *columns of the matrix* $\mathbf{H}$ *defined in Equation 25 can be used to solve a system of equations with* t *unknown symbols.*

Proof. First, note that a selection of t columns of $\mathbf{H}$ produces a square matrix. If we can show its determinant is non-zero, we have shown it can be used to solve a system of equations with t unknowns (see Section 2.1.2).

Let the index of the kth column selection be represented by $v_k + 1$, identify the index of the kth column selection. Thus, each $v_k \in [0, n)$ and is unique. The reduced matrix $\mathbf{H}'$ has the form:

$$\mathbf{H}' = \begin{bmatrix} \alpha^{v_1} & \alpha^{v_2} & \alpha^{v_3} & \cdots & \alpha^{v_t} \\ (\alpha^{v_1})^2 & (\alpha^{v_2})^2 & (\alpha^{v_3})^2 & \cdots & (\alpha^{v_t})^2 \\ & & & \vdots & \\ (\alpha^{v_1})^t & (\alpha^{v_2})^t & (\alpha^{v_3})^t & \cdots & (\alpha^{v_t})^t \end{bmatrix} \tag{27}$$

We now take advantage of the fact that, if a matrix is transposed, its determinant is unchanged, and that if a row of the matrix is multiplied by the scalar z, the determinant changes by a factor of z [2]. We will transpose the matrix then multiply each row k by the scalar α^{-v_k}, so that the total scaling factor of the determinant is $\alpha^{-(v_1+v_2+v_3+\cdots+v_t)}$. The determinant $|\mathbf{H}'|$ is given by:

$$|\mathbf{H}'| = \alpha^{-(v_1+v_2+v_3+\ldots+v_t)} \begin{vmatrix} 1 & \alpha^{v_1} & (\alpha^{v_1})^2 & \cdots & (\alpha^{v_1})^{t-1} \\ 1 & \alpha^{v_2} & (\alpha^{v_2})^2 & \cdots & (\alpha^{v_2})^{t-1} \\ & & & \vdots & \\ 1 & \alpha^{v_t} & (\alpha^{v_t})^2 & \cdots & (\alpha^{v_t})^{t-1} \end{vmatrix} \tag{28}$$

The scale factor $\alpha^{-(v_1+v_2+v_3+\ldots+v_t)}$ cannot be zero. The determinant in brackets is equal to the determinant of a Vandermonde matrix, whose elements a_k are given by α^{v_k}. Each v_k is unique and in the range $[0, n)$ by definition; because α is a primitive nth root, each α^{v_k} are unique also (Equation 21). Therefore, the determinant in brackets is non-zero also. The product of two non-zero numbers is itself non-zero, and thus $|\mathbf{H}'| \neq 0$. □

3.3.3 Drawbacks to RS Solution

RS code parity-check matrices are easy to construct for any given finite field $\mathbf{F}_q$, and for any number of erasures t. However, word-size n is constrained by $q - 1$, or in terms of symbol-size m:

$$n \leq 2^m - 1 \tag{29}$$

We can undercut word-size simply by having Alice and Bob ignore columns of $\mathbf{H}$, but if we try to tack on additional columns, each α^{v_k} in the above proof are no longer necessarily unique, revoking our guarantee that the reduced matrix $\mathbf{H}'$ is nonsingular.

The other major problem is that erasures must occur on the symbol level, while in most practical communication scenarios, they are expected to occur on the bit level. Selecting $m = 1$ forces n to be 1, which is not a very interesting scenario.

We can salvage the protocol by letting m be whatever it must to allow the desired word-size n, and treating a single-bit erasure as an erasure of the entire m-bit symbol. This means rate suffers significantly: Alice must assume each erasure can occur in a different m-bit block, so she must prepare her redundancy s with $c = mt$ bits, where theoretically only t bits are necessary.

Furthermore, a great deal of information (the rest of a partially erased symbol) is

Protocol 2: Nonbinary Protocol

Setup Alice has a string x, consisting of n m-bit symbols. Bob has a string y, which is identical to x except that t symbols have been erased.

Parameters (agreed upon before communication)

α - a generator of $\mathbf{F}_q$, where $q = 2^m$.

Steps

1. Both Alice and Bob calculate the $(t \times n)$ matrix $\mathbf{H}$ such that $\mathbf{H}_{ij} = \alpha^{(i-1)j}$
2. Alice calculates the redundancy $s = \mathbf{H}x$.
3. Alice transmits s to Bob.
4. Bob calculates the string y' as y with each erasure replaced with a zero.
5. Bob calculates $z = \mathbf{H}y' - s$.
6. Bob calculates the reduced matrix $\mathbf{H}'$ by removing each column of $\mathbf{H}$ where y did not have an erasure.
7. Bob solves the equation $\mathbf{H}'e' = z$ for e, using linear algebraic techniques.
8. Bob calculates the error vector e by padding e' with zeros wherever y did not have an erasure.
9. Bob calculates Alice's string $x = y - e$.

ignored. Perhaps a more sophisticated protocol can make use of this extra information and recover some of the loss, but it is necessarily more complicated than we are willing to explore.

3.3.4 Failed Alternative: Binary BCH Codes

You may note that the unique properties of the RS parity-check matrix apply just as well to those of any BCH code. Can we, therefore, get away with using a binary BCH code, and avoid the drawbacks we saw in Section 3.3.3?

The answer is no. First of all, the constraint given by Equation 29 still exists for binary BCH codes: n must be of the form $2^m - 1$ so that the polynomial arithmetic works out. This alone is not a problem: we need not work in m-bit blocks, and we can always undercut word-size by ignoring columns of $\mathbf{H}$, so we may always increase m without expense.

The problem is that the t elements of the syndrome of an n-vector will usually contain α, which again doesn't really *exist* for a generic BCH code. Alice will have trouble transmitting the syndrome.

Although it isn't easy, it turns out that every BCH syndrome actually can be reduced to a single *power* of α, or 0 if the argument is a codeword, so our protocol could have Alice transmit the power (or a zero). There are therefore $n + 1$ possible symbols for each element of the syndrome, which require $log_2 n + 1 = m$ bits to represent. Thus, the total number of bits Alice must send is still $c = mt$. The expense from having Alice reduce syndrome elements to a single power of α gains her nothing in her transmission efficiency.

3.4 RM Theory: a Nonlinear Solution

In this section, we use use theory from **Reed-Müller (RS)** codes to perform reconciliation. Standard literature such as that in [6] cleverly uses the symbols r and m to parameterize RM codes, but since m is already a reserved symbol, we shall instead use ρ and μ. [22], in addition to [6], provides a helpful overview of RM coding.

3.4.1 Polynomials: a Different Perspective

In order to appreciate how our RM solution fits in to what we have done so far, let us first reexamine what we have done so far from a slightly different perspective. As usual, let x be Alice's string, m be its symbol-size, and s be her redundancy. Let alphabet size $q = 2^m$, so that each symbol of x is an element of $\mathbf{F}_q$. $\mathbf{H}$ is an agreed upon matrix used to calculate s.

Consider the binary linear reconciliation scenario as follows: think of the bits of x as coefficients $c_1, c_2 \ldots c_n$ to a first-order (ie. linear) n-variate polynomial over $\mathbf{F}_2$:

$$P(\overline{\chi}) = c_1\chi_1 + c_2\chi_2 + \ldots + c_n\chi_n \tag{30}$$

The argument $\overline{\chi}$ is a point in $\mathbf{F}_2^n$. Think of each row of $\mathbf{H}$ as a coordinate, itself a point in $\mathbf{F}_2^n$. Then s contains $P(\overline{\chi})$ evaluated at each of those points. If the distance between each point in $\mathbf{H}$ is at least $d \geq t + 1$, then Bob can solve the system for his missing values of x.

Consider the RS reconciliation scenario differently: think of the symbols of x as coefficients to a univariate polynomial over $\mathbf{F}_q$, with degree $n - 1$:

$$P(\chi) = c_1 + c_2\chi + c_3\chi^2 + \ldots + c_n\chi^{n-1} \tag{31}$$

Due to the strict structure of $\mathbf{H}$, each element s_k of s is $P(\alpha^k)$, where α^k is a unique point in $\mathbf{F}_q$. Since the argument to the polynomial is just one symbol, and each of the arguments for s_k are unique, Bob can solve the system (which *is* linear over c_k) for his missing values of x.

Now let us imagine a more complicated polynomial $P(\overline{\chi})$, which is neither linear nor univariate. Let ρ be the degree of $P(\overline{\chi})$ and μ be the number of variables. For the sake of sanity, let's keep to $\mathbf{F}_2$, so that $\overline{\chi} \in \mathbf{F}_2^m$. This means any value raised to a power $k > 1$ is $\chi^k = \chi$; the non-linearity comes only from terms with more than one variable. Note that this means $\rho > \mu$ never adds any terms. As an example, here is a polynomial with $\rho = 2$, $\mu = 2$:

$$P(\overline{\chi}) = c_1 + c_2\chi_1 + c_3\chi_2 + c_4\ \chi_1\chi_2 \tag{32}$$

Following the same procedure, think of the bits of x as the coefficients to a ρ-degree, μ-variate polynomial $P(\overline{\chi})$ over $\mathbf{F}_2$ such that:

$$n(\rho, \mu) = \sum_{k=1}^{r} \binom{m}{k} + 1 \tag{33}$$

Given an $\mathbf{H}$ whose rows are points in $\mathbf{F}_2^m$, Alice will calculate s to be $P(\overline{\chi})$ evaluated at each of those points. The questions we must ask are these:

1. What distance must the points of $\mathbf{H}$ have to guarantee Bob can solve the system for his missing values of x?

2. For what n and t does such a matrix $\mathbf{H}$ exist, and how can we construct it?

To answer these questions, we will turn to the theory of RM codes.

3.4.2 RM Codes

Each codeword γ in a (ρ,μ) RM code is associated with a unique n-length string x, whose bits, as in Section 3.4.1, should be thought of as the coefficients to a ρ-degree, μ-variate polynomial $P(\overline{\chi})$ over $\mathbf{F}_2$. There are 2^n unique x, so there are 2^n unique codewords. Each bit in γ is the solution to $P(\overline{\chi})$ at a different point in $\mathbf{F}_2^m$, so γ has 2^m bits. We order these solutions such that γ_i is $P(\overline{\chi_{i-1}})$ where $\overline{\chi_k}$ is the binary representation of k.

The redundancy s that Alice would like to send is therefore a selection of t bits from the RM codeword γ associated with her string x. For his part, Bob lists all the possible γ that could correspond to his y, assuming each X is 0 and then 1. He then selects from this list using the bits Alice sends. The question is this: which bits should those be?

Consider an example using the (1,3) RM code: from Eqaution 33, we see $n = 4$. Let's say x =0101, and y =0X01. The codeword γ associated with x is:

$$\gamma = 01011010$$

As far as Bob is aware, there are two codewords he must select from:

$$\gamma = \begin{cases} 00001111 & \text{if X= 0} \\ 01011010 & \text{if X= 1} \end{cases}$$

Say Alice decides to send the first bit from γ. That doesn't tell Bob anything: he already knew the first bit of γ from his unerased bits in y. She needs to send an even-numbered bit to be useful. But wait! What if the erasure were in a different spot?

Say instead y =010X. Bob would need to select from the following two codewords:

$$\gamma = \begin{cases} 01010101 & \text{if X= 0} \\ 01011010 & \text{if X= 1} \end{cases}$$

Now Alice needs to send any bit from the latter half of γ. Which bits Bob knows depends on the erasure pattern. Alice needs to pick a bit that Bob could have no information on, no matter where the erasure occurred. For this selection of parameters, the last bit is the only such bit: the polynomial produced is essentially the same as the one we could make from Section 3.2.1. Given $\gamma_m = 0$, Bob knows that his X is a 1.

In general, each different (ρ, μ) will have a different set of t bits for which Bob cannot have any information. Furthermore, not all t necessarily have such a selection of bits. Thus, this protocol is useful only for specific parameters.

3.4.3 Failed Alternative: First Many Bits

It would make for an especially simple protocol if Alice could simply send the first c bits of γ as s. We can ask for what values of c would such an s allow Bob to select from an arbitrary set of (ρ,μ) RM codewords.

The minimum Hamming distance of the code is $d = 2^{\mu-\rho}$ [6]: There can only be at most $2^\mu - d$ bits which are identical between two arbitrarily selected codewords. Therefore, if s is any $c \geq 2^m - d + 1$ bits of the correct γ, then Bob will be able to use it to select from any two (or more) possible γ. But we also require $c < n$, since if we are sending n bits they might as well just be x. We are left with the following inequality:

$$2^\mu - 2^{\mu-\rho} + 1 \leq c \leq \sum_{k=1}^{r} \binom{m}{k} \tag{34}$$

Protocol 3: Nonbinary Protocol

Setup Alice has a string x, consisting of n m bits. Bob has a string y, which is identical to x except that t bits have been erased.

Parameters (agreed upon before communication)

μ - designates a parameter of an RM code.

ρ - designates the other parameter of an RM code. Must be between 1 and μ.

Steps

1. Both Alice and Bob calculate which set of t bits of a codeword in a (ρ,μ) RM code will be sufficient to select from a set of codewords whose associated coefficient strings are separated by at most t bits.
2. Alice calculates the codeword γ associated with her coefficient string x.
3. Alice calculates the redundancy s as the t bits of γ decided upon in Step 1.
4. Alice transmits s to Bob.
5. Bob calculates the set of codewords corresponding to y, with each X replaced with a 0 or a 1.
6. Bob identifies the codeword γ whose t bits decided upon in Step 1 match s.
7. Bob calculates Alice's string x as the coefficient string associated with γ.

This is obviously[4] never true: the n-length coefficient vectors x have a one-to-one mapping with codewords γ. We can not possibly select from any possible set of γ with fewer than n bits. Just as Bob could not simply solve for x in Section 2.2 without first accounting for his constraints from y, so too can he not simply solve for x here.

[4] ...in retrospect...

3.5 Random Matrices: a Probabilistic Solution

In this section, we follow the same linear binary reconciliation protocol followed in Section 3.2.2, but we relax the condition that **H** must produce a nonsingular system for every erasure pattern. That is, the protocol described in this section *won't work* at a failure rate ϵ.

Note that almost any $(t \times n)$ matrix con correct some $t = 3$ erasure patterns and not others. The Hamming code parity-check matrix from Section 3.2.3, for instance, has fails for 7 out of 35 possible erasure patterns and thus has $\epsilon = .2$. This is a far better ϵ than comparable matrices we will use in this section. However, it is deeply unsatisfactory because those 7 erasure patterns for which Hamming's matrix fails can *never* be corrected.

The premise of the protocol is to construct the matrix **H** randomly, and then use it exactly as prescribed in Section 3.2.2. In order for Bob to solve a system with the same **H** Alice used to calculate s, they will clearly need to have a source of shared randomness. This will take the form of a pseudorandom generator seeded with the same value. The nature of this pseudorandom generator is discussed in more detail in Section 3.5.3.

Since effective rate (Equation 11) is already suffering from non-zero ϵ, we shall also consider how increasing the rows of **H**, and therefore the redundancy length c, beyond the theoretically minimum value t affects the effective rate. Increasing c will of course cause communication complexity to suffer, but it will also decrease ϵ and may yield a better effective rate. We are particularly interested in the effective excess communication complexity Δ:

$$0 \leq \Delta = c(1-\epsilon)^{-1} - t \tag{35}$$

We would like to find the optimal parameter c for any given (n, t), such that Δ is closest to its theoretical minimum of 0. Section 3.5.1 discusses numerical methods we have used to explore this question, while Section 3.5.2 relates analytically-derived proofs.

3.5.1 Numerical Simulation

To study how effective excess communication complexity Δ depends on n, t, and c, we conducted a numerical simulation with the following procedure:

1. Experiment on n, t, and c:
   (a) Generate $(c \times n)$ matrix $\mathbf{H}$ from random source $\mathbf{R}$
   (b) Randomly select t columns of $\mathbf{H}$ to form $\mathbf{H}'$
   (c) If $\text{rk}(\mathbf{H}') < t$, count as a failure
2. Calculate ϵ for (n, t, c):
   (a) Perform experiment in Step 1 N times
   (b) Count failures as z
   (c) Calculate $\epsilon = z/N$

All results reported used trial number $N = 1000$. We will explore each of the following curves analytically in Section 3.5.2. For now, note only that, theoretically, n should not have any impact on ϵ. If each row of $\mathbf{H}$ is truly random, any submatrix $\mathbf{H}'$ is just as random: the properties of the smaller matrix should have nothing to do with the larger one. In practice, this may or may not be true, depending on how random $\mathbf{R}$ truly is.

Protocol 4: Probabilistic Protocol

Setup Alice has a string x, consisting of n bits. Bob has a string y, which is identical to x except that t bits have been erased.

Parameters (agreed upon before communication)
c - communication complexity. Must be greater than or equal to t.

$\mathbf{R}$ - shared random source which produces elements of $\mathbf{F}_2^n$ pseudorandomly.

Steps

1. Both Alice and Bob construct a $(c \times n)$ matrix $\mathbf{H}$, by stacking c n-bit sequences drawn from $\mathbf{R}$.
2. Alice calculates the redundancy $s = \mathbf{H}x$.
3. Alice transmits s to Bob.
4. Bob calculates the string y' as y with each erasure replaced with a zero.
5. Bob calculates $z = \mathbf{H}y' - s$.
6. Bob calculates the reduced matrix $\mathbf{H}'$ by removing each column of $\mathbf{H}$ where y did not have an erasure.
7. Bob solves the equation $\mathbf{H}'e' = z$ for e, using linear algebraic techniques. If $\mathbf{H}'$ is nonsingular, the protocol fails.
8. Bob calculates the error vector e by padding e' with zeros wherever y did not have an erasure.
9. Bob calculates Alice's string $x = y - e$.

Figure 1 studies how failure-rate ϵ varies with t when we choose $c = t$. That is, if Alice sends as few bits as theoretically possible with this protocol, what is the worst failure-rate we can expect for any given t? Results are shown for $n \in \{5, 15, 127\}$. The analytical curve will be derived in Section 3.5.2. Notice that for every n, the failure-rate starts close to $\epsilon = 0.5$, but grows rapidly. Although there is high variance, it then seems to plateau somewhere close to $\epsilon = 0.7$.

Figure 2 studies how failure-rate ϵ varies as we permit Alice to transmit extra bits. The horizontal axis represents $\delta = c - t$, the number of extra bits Alice is sending. Results are shown for $n = 127$ and $t \in \{10, 55, 127\}$. The analytical curve will be derived in Section 3.5.2. Although the failure-rate starts high (close to $\epsilon = 0.7$ as observed in Figure 1), it quickly decays to nearly 0. In fact, for $N = 1000$, we found consistently that by $\delta > 15$, no experiments reported any failures, suggesting $\epsilon < .001$.

Figure 3 studies how effective excess communication complexity Δ varies as we permit Alice to transmit extra bits. The plot is essentially the same as in Figure 2, but the vertical axis is transformed by $\Delta = c(1-\epsilon)^{-1} - t$. Note that while ϵ seems to depend only on δ, the curve for Δ changes with t. The minima of the curves select the c for a set of parameters (n,t) for which effective rate is a minimum.

Figure 4 plots these minima explicitly, for $n \in \{5, 15, 127\}$. Again note that the overall curve does not appear to depend on n. The results are highly heteroscedastic (higher variance at larger t), but are consistent with the theoretical result. Interestingly, the data appear to loosely fit the curve $1 + \log t$. Even in Section 3.5.2, there is no easily apparent reason for this to be true, but it provides a very helpful approximation.

Finally, Figure 5 plots these same minima over word-size n, where x and y are related by different erasure channels, producing $t = f(n)$ erasures, where $f(n)$ is some

function of n. The functions shown in this figure are for $f(n) = \{\lceil \frac{n}{10} \rceil, \lceil \frac{n}{4} \rceil, \lceil \log_2 n \rceil\}$. Note that for $f(n) = \log_2 n$, Figure 4 (plotting Δ over t) is exactly equivalent to Figure 5 (plotting Δ over n) had we chosen to plot the horizontal axis on a log scale.

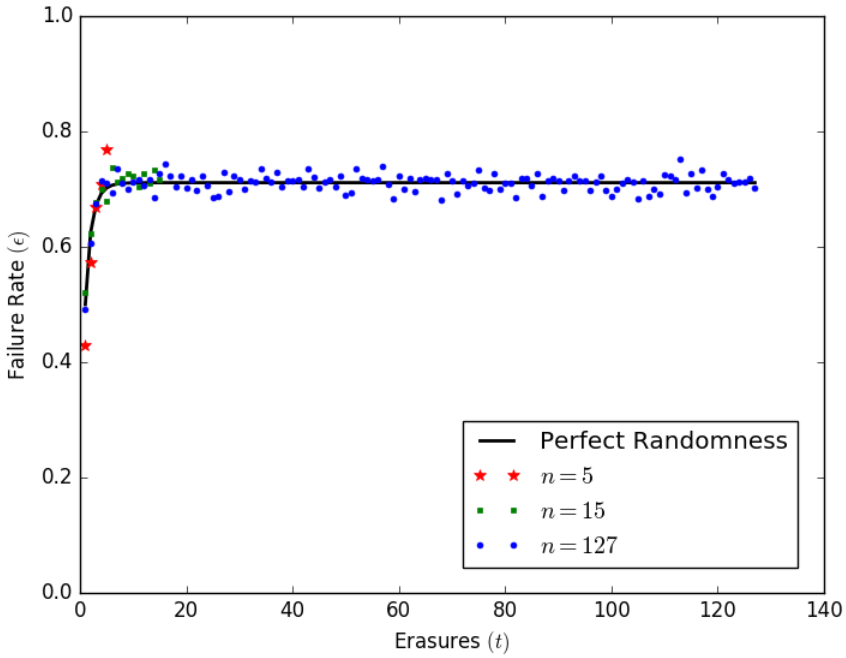

Figure 1: Failure-rate when $c = t$

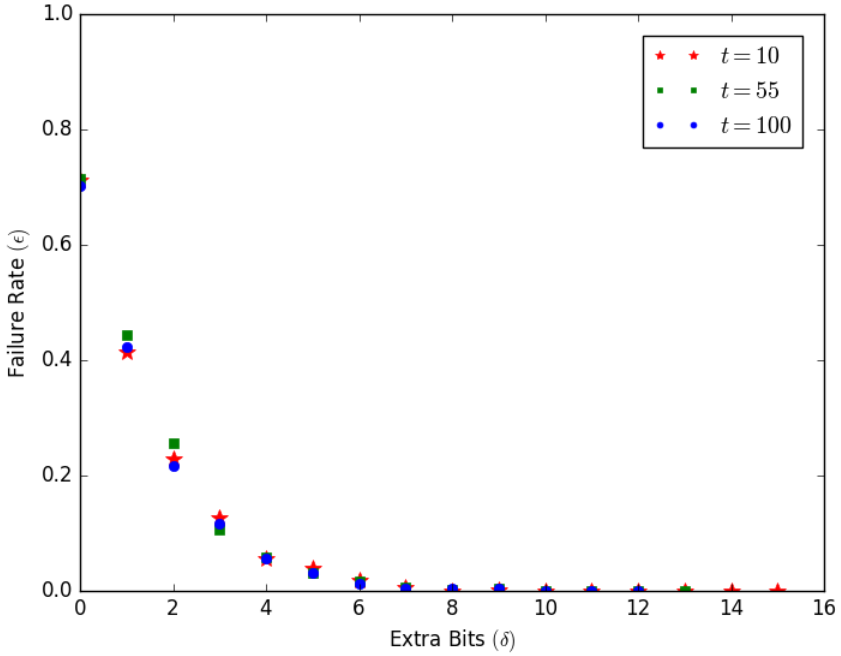

Figure 2: Failure-rate as extra bits are added: $n = 127$

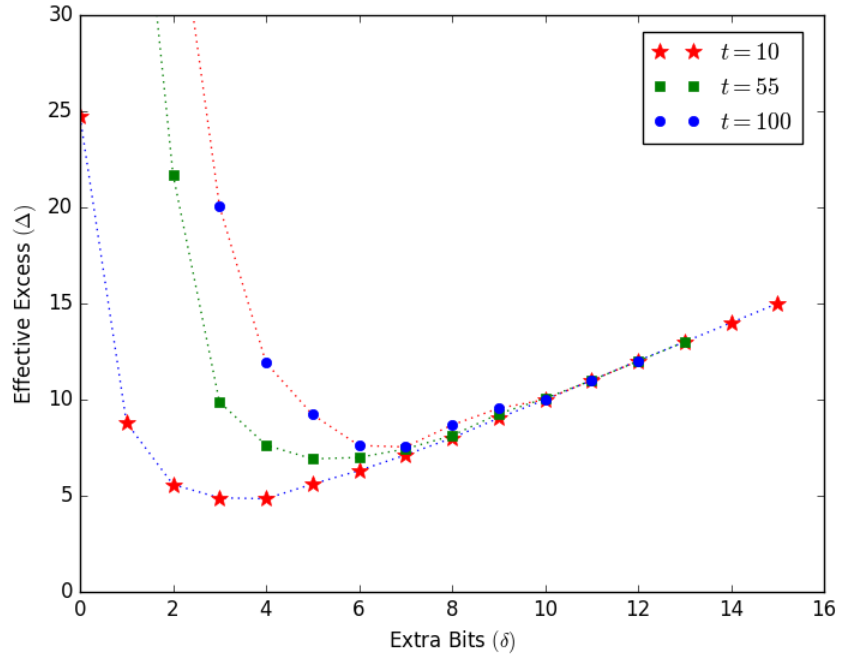

Figure 3: Effective excess communication complexity: $n = 127$

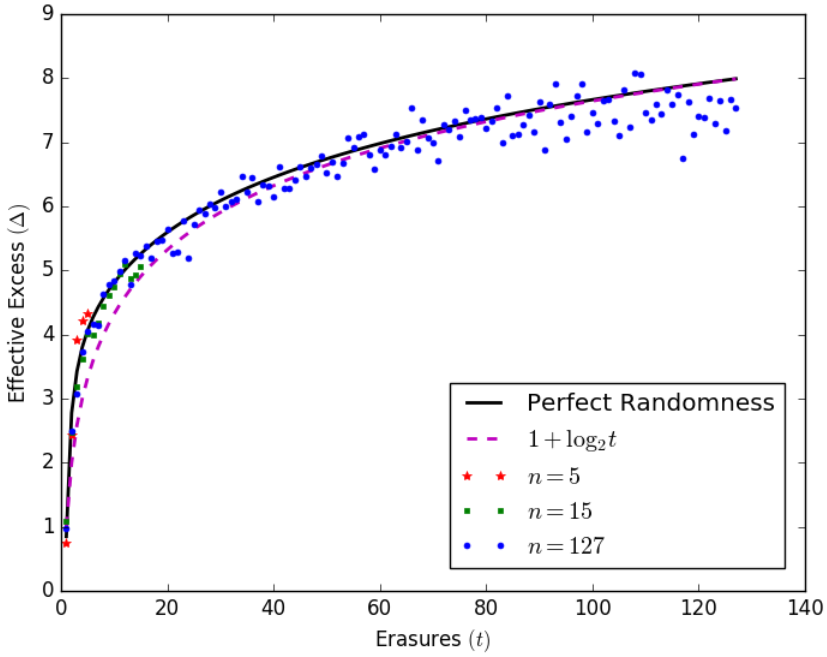

Figure 4: Lowest achievable effective excess communication complexity

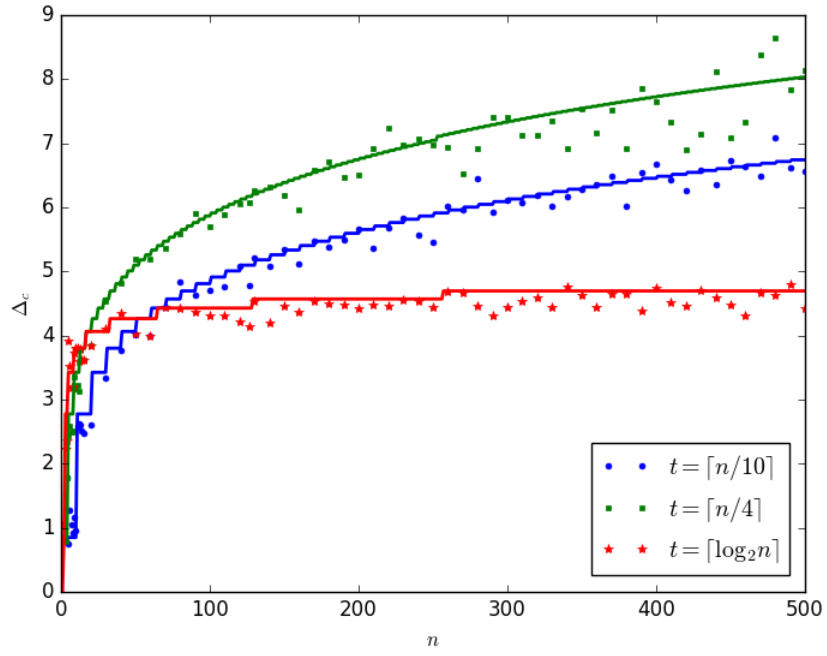

Figure 5: Lowest achievable effective excess communication complexity (word-size)

3.5.2 Theoretical Analysis

In this section, we calculate theoretical expressions to describe the data collected in Section 3.5.1. The introduction of [23] provides an excellent primer in the properties of random binary matrices. Although we were able to produce all results independently, we would have been lost without its guidance.

As noted at the start of Section 3.5.1, the theoretical treatment is entirely independent of n. We are interested in four quantities:

1. $\epsilon(t, \delta)$ - the probability that a matrix with t columns and $c = t + \delta$ rows does not have full rank

2. $\Delta(t, \delta)$ - the effective excess communication complexity resulting from the parameters t and $c = t + \delta$

3. $\delta_{\min}(t)$ - the choice of δ which minimizes $\Delta(t, \delta)$ for a given t

4. $\Delta_{\min}(t)$ - the lowest achievable effective excess communication complexity for a given t

The first quantity ϵ is given by:

$$\epsilon(t, \delta) = 1 - \prod_{k=0}^{t-1} \left[1 - 2^{-(t+\delta)+k}\right] \tag{36}$$

Given ϵ, Δ comes easily:

$$\Delta(t, \delta) = \frac{\delta + t\epsilon}{1 - \epsilon} \tag{37}$$

However, it is not obvious how find $\delta_{\min}$. Ideally, we would like to have a closed form expression for ϵ, so that we can take[5] $\frac{\partial}{\partial \delta}\Delta = 0$ and optimize δ. Before we explore that, however, let us first explain where Equation 36 even comes from.

[5] ...the most confusing differential I have ever seen. Who picked these symbols?!

Setup In order for a $(c \times t)$ matrix to have full rank, no column can be represented as a linear combination of the other columns. This is equivalent to saying that no column may be in the span of the remaining columns. Let's say you have $k < c$ linearly independent columns already: what are the chances that a random c-vector is not in their span?

The span of $k < c$ independent columns is 2^k. There are 2^c possible c-vectors. If the matrix is truly and uniformly random, then the chance of selecting a redundant column is $2^{-(c-k)}$. The chance that you *don't* do this is $1 - 2^{-(c-k)}$. The chance that you *don't* do this t times in a row is:

$$P(t, c) = \prod_{k=0}^{t-1} \left[1 - 2^{-(c-k)}\right] \tag{38}$$

Substituting $\delta = c - t$ and taking the negation produces $\epsilon(t, c)$.

First Order Approximation The probability $P(t, c)$ can take on a closed form if we are willing to approximate. Imagine the product in Equation 38 expanded out into each of its terms. On the far left is the product of all the ones, which is of course 1. This is the "zeroth-order" approximation, since it has no factors of the form $2^{-(c-k)}$. The first-order approximation takes into account for the next group of terms, comprised of the $2^{-(c-k)}$ term in each individual binomial multiplied by only ones.

$$P(t, c) \approx 1 - \sum_{k=0}^{t-1} 2^{-(c-k)} \tag{39}$$

Let us consider how good an approximation this is by determining the largest possible contribution from a second order term of the form $2^{-(c-k_1)}2^{-(c-k_2)}$. This is when $c = t$ and k is as close to t as it can be, for both factors of the form $2^{-(c-k)}$.

Thus, this second order term amounts to:

$$2^{-[t-(t-1)]-[t-(t-2)]} = \frac{1}{8} = .125 \tag{40}$$

This is not negligible for a probability: the approximation is not good for very small $\delta = c - t$. However, as δ increases, the magnitude of this largest term falls drastically. And while there are a lot of second-order terms that could add up, each one is at least a magnitude smaller than this one. In short, after about $\delta > 3$, we'll hardly notice the difference. Needless to say, the same goes for all higher-order terms as well.

After a bit of algebra, and substituting c for δ, the first-order approximation in Equation 41 simplifies to the closed form expression:

$$P(t, \delta) \approx 1 - 2^{-\delta}(1 - 2^{-t}) \tag{41}$$

Since P is just the negation of ϵ, we can see the first-order approximation of ϵ is:

$$\epsilon(t, \delta) \approx 2^{-\delta}(1 - 2^{-t}) \tag{42}$$

Abbreviating $1 - 2^{-t}$ as z, we can produce the first-order approximation of Δ from Equation 37:

$$\Delta(t, \delta) \approx \frac{\delta + 2^{-\delta} t z}{1 - 2^{-\delta} z} \tag{43}$$

Transcendental Equation Setting $\frac{\partial}{\partial \delta}\Delta = 0$ produces the following equation:

$$2^{-\delta_{\min}} z(1 + t \ln 2 + \delta_{\min}) = 1 \tag{44}$$

This equation, though it has a solution, cannot be solved analytically to the best of our knowledge. We can still supplement our empirical results with theory by solving for $\delta_{\min}$ numerically. The lowest achievable effective excess communication complexity $\Delta_{\min}$ is then given by:

$$\Delta_{min}(t) = \Delta(t, \delta_{\min}) \tag{45}$$

Square Matrices The first-order approximation is not a good one when δ is very small. Consider the case $\delta = 0$, a square matrix. Our empirical results from Figure 1 indicate that the resulting expression for $\epsilon_n = \epsilon(n, 0)$ converges to a value close to .7. In this section, we prove it.

Consider an alternate form of ϵ: $g(n) = \ln(1 - \epsilon_n)$, which is more conducive to clever manipulation.[6]

$$\begin{aligned}
g(n) &= \ln(1-\epsilon_n) \\
&= \sum_{k=0}^{n-1} \ln\left[1 - 2^{-(n-k)}\right] && \text{Note the product has become a sum.} \\
&= \sum_{k=1}^{n} \ln(1 - 2^{-k}) && \text{Note the more manageable indices.} \\
&= -\sum_{k=1}^{n} \sum_{l=1}^{\infty} \frac{2^{-lk}}{l} && \text{Maclaurin expansion of } \ln(1-x) \\
&= -\sum_{l=1}^{\infty} \sum_{k=1}^{n} \frac{1}{l} (2^{-l})^k && \text{Now in the form } ar^k \\
&= -\sum_{l=1}^{\infty} \left(\frac{2^{-l}}{l}\right) \frac{1 - 2^{-nl}}{1 - 2^{-l}} && \text{Geometric progression}
\end{aligned}$$

The remaining sum is a relic from our Maclaurin expansion. Our x was not terribly close to 0, so we will need to take several terms to have a decent approximation.

[6]Please note that the calculus gets heavy here and I can't afford to explain everything in full detail. See [2] for an excellent overview of most important math.

Since we are interested in seeing what value ϵ converges to, we should take the limit as $n \to \infty$:

$$\begin{aligned} \lim_{n\to\infty} g(n) &= -\sum_{l=1}^{\infty} \left(\frac{2^{-l}}{l}\right) \frac{1}{1-2^{-l}} \\ &= -\sum_{l=1}^{\infty} \left[l(2^l-1)\right]^{-1} \\ &\approx -1.2420 \end{aligned} \tag{46}$$

This limit corresponds to a value of $\epsilon \approx .7112$, exactly as we expect.

3.5.3 Random Number Generation

The random source **R** that Alice and Bob agree on should be a pseudorandom generator, but this can take several forms. For the results presented in Figures 1-4, we used the random number generator native to the numpy package in Python, which is based on the **Mersenne Twister** algorithm. However, we will first explain its simpler alternative, the **Linear Feedback Shift Register (LFSR)**.

LFSR The Linear Feedback Shift Register is a linear pseudorandom bit generator, which calculates each bit based on the parity of the previous n bits it has generated. Which bits it takes is based on an n-length coefficient string x, similar to what we saw in Section 3.4.1. The bit x_i determines whether or not the bit generated i iterations ago is used in the parity check. In essence, if z is an n-vector of the last n bits generated, such that z_1 was the last bit generated, then the next bit z_0 is given by the dot product $z_0 = x \cdot z$.

It can be shown [1] that if x is the coefficient vector of a **primitive** polynomial $P(\chi)$ of degree n (see Equation 47), then $2^n - 1$ bits are generated before the sequence repeats.

$$P(\chi) = \chi^n - x_n\chi^{n-1} - \ldots - x_1 \tag{47}$$

Primitive polynomials of varying degree can be found from a table such as that in [24].

Our implementation of a linear feedback shift register produces a new n-vector in $\mathbf{F}_2^n$ by calculating z_0 of the last n-vector, rotating that last n-vector by 1 space, and replacing the last bit with z_0. Each n-vector is very clearly related to the one before it: it is a "random" source in a very loose sense of the word. However, it serves our purposes.

NUMPY The Mersenne Twister algorithm is at its heart an LFSR generator with several additional complicated steps that make the results appear more random. The random source we utilize, from the numpy package, is based on this algorithm, but actually hashes each number from the Mersenne Twister down to a single bit, rather than using the number to represent n bits. This means that to generate a single n-vector, we require n times as many numbers from the Mersenne Twister, but it also means that the parameters used to construct the Mersenne Twister are independent of the n we require, making it a much more flexible algorithm.

Failed Alternative: RC4 Another class of pseudorandom generators is one based on a **stream cipher**, used in cryptographyq to encrypt messages in real time. The encryption algorithm is easily adapted to produce a stream of bits, which we can take n at a time. One such stream cipher is **RC4** [25], which involves repeated bit-wise XOR and permutation operations based on the bits themselves. We implemented a random source utilizing this strategy as well, but preliminary results indicated that it was several orders of magnitude slower than either of the others. Since there was no reason to expect RC4 to perform significantly differently, we chose to omit this source from the results presented below.

Comparison The LFSR method is particularly suited to our problem of generating a matrix $\mathbf{H}$, because it will never produce a zero-vector, and it will only produce duplicate vectors after it has run the course of its period, which for a suitable selection of $P(\chi)$ is far greater than the number of vectors we need c. While *not* having duplicate rows does nothing to guarantee our protocol will be successful, a duplicate row will never contribute to a system of equations, making it that much less likely the reduced matrix $\mathbf{H}'$ has rank t. Thus, we can expect a random source based on LFSR will perform slightly better than a source which is more truly random.

We test this expectation by studying how failure-rate ϵ varies with t when we choose $c = t$, as we did before in Figure 1, but this time we perform the experiment using $\mathbf{R}$ as LFSR and as NUMPY. The results are shown in Figure 6 ($n = 15$) and Figure 7 ($n = 127$).

As expected, LFSR clearly performs slightly better. However, when n is large and t is small, the differences between the two are negligible. We found that, when n is large, our LFSR source is still preferable because it is somewhat faster. That said, it requires knowing a primitive polynomial of degree n, which is not generally easy to construct manually.

Note that when t is very large with respect to n, LFSR suddenly performs *much* better, and when $t = n$, $\epsilon = 0$. This apparently indicates that an $(n \times n)$ matrix with rows generated by LFSR is necessarily non-singular. Exploring why this happens is beyond the scope of this thesis. However, we suspect that there may be other unexpected - and undesirable - side effects of using LFSR, such that a given seed may intrinsically guarantee certain erasure patterns cannot be corrected. If the advantageous behavior at high t is truly needed, it may be better to instead use a method like the one proposed in [26], which algorithmically constructs a nonsingular square matrix from any random bit generator $\mathbf{R}$.

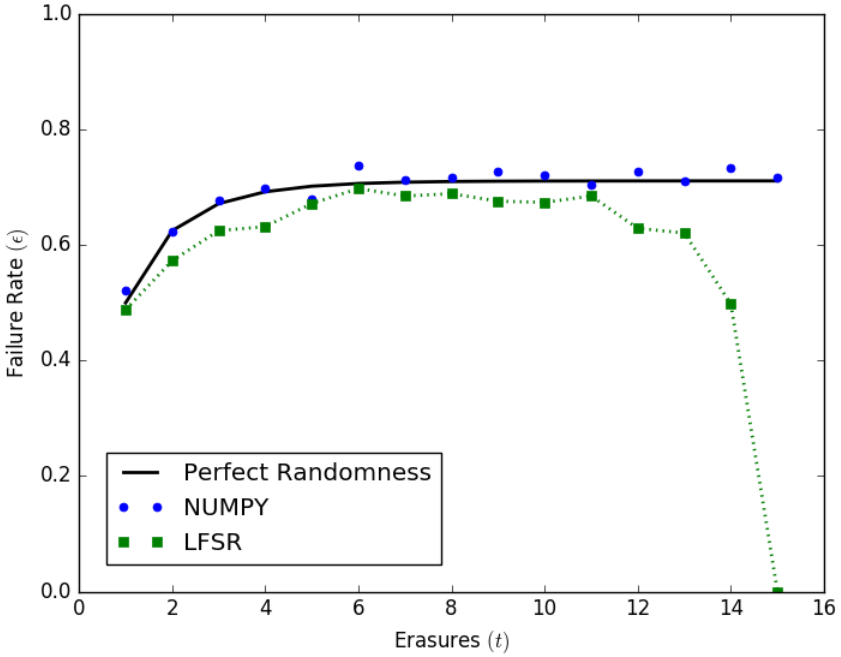

Figure 6: Comparing Random Sources: $n = 15$

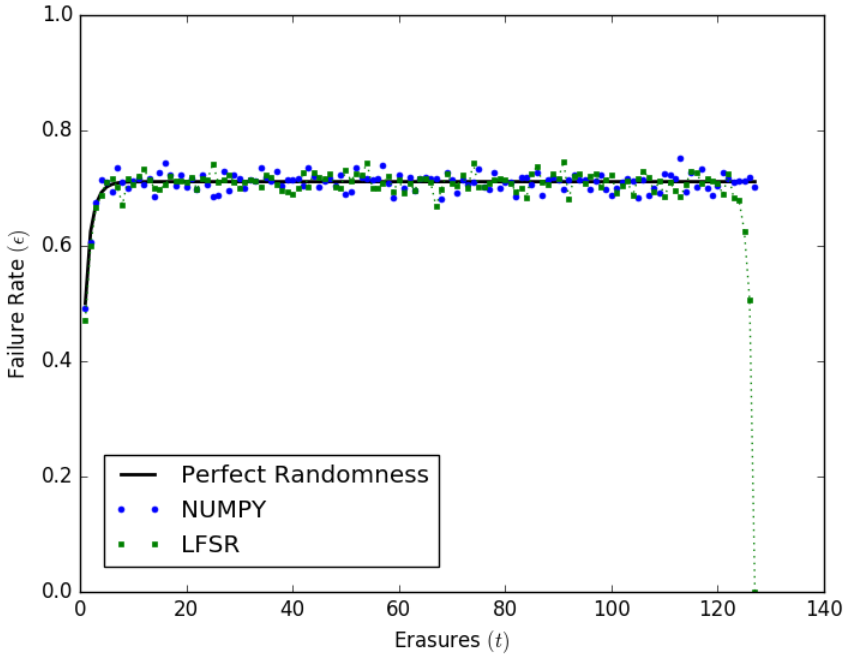

Figure 7: Comparing Random Sources: $n = 127$

4 Conclusions

In this thesis, we have seen four protocols to perform erasure reconciliation. Each has its advantages and limitations; we review them here:

The ideal protocol presented in Protocol 1 requires an MDS $[n, t, n-t+1]_2$ linear code. Such codes only exist for $t \in \{0, 1, n-1, n\}$.

The nonbinary protocol presented in Protocol 2 requires that erasures occur in m-bit blocks, and constrains $n \leq 2^m - 1$. It is also somewhat harder to implement, since it requires arithmetic in finite fields higher than $\mathbf{F}_2$.

The nonlinear protocol presented in Protocol 3 requires that a pair of parameters ρ and μ must be selected such that $n \leq \sum_{k=1}^{r} \binom{m}{k} + 1$. Though ρ and μ may be as large as necessary, a set of t indices must be known such that, in a (ρ,μ) RM codeword, knowing those t bits is sufficient to select from a set of codewords whose associated coefficient strings are separated by at most t bits.

The probabilistic protocol presented in Protocol 4 has no parameter constraints, but will have suboptimal communication complexity based on the additional parameter c, and of course it is probabilistic: it will fail at rate $\epsilon(n, t, c)$. It also requires that Alice and Bob have a source of shared randomness.

Communication Complexity In all three deterministic protocols, communication complexity is exactly $c = t$ bits, which is the best we could hope for in erasure reconciliation. In the probabilistic protocol (Protocol 4), the exact communication

complexity is the parameter c, but increasing c will tend to lower failure-rate ϵ. The optimal choice of c results in an effective communication complexity near $t+\log_2 t+1$. Naturally, one should only resort to the probabilistic protocol when none of the other protocols are usable for a given n and t.

Protocol 2 has a binary adaptation discussed in Section 3.3.3 which is adaptable to any n and t, but in which communication complexity is mt, where m is the smallest integer such that $n \leq 2^m - 1$. This adaptation performs better than Protocol 4 when:

$$mt < t + \log_2 t + 1 \tag{48}$$

To explore this further, suppose $t = \beta n$, where β may be consdered the probability that any given bit is erased. Also let n be as large as possible for a given m, such that $m = \log_2 (n+1) \approx \log_2 n$. In terms of β and n, Equation 48 is equivalent to:

$$\beta n < \frac{\log_2 n + 1}{\log_2 n - 1} \tag{49}$$

The left-hand side is simply the expected number of erasures t. If $t = 1$, we should simply use the parity-bit case from Protocol 1. If $\beta n = t \geq 2$, Equation 49 reduces further to:

$$n < 8 \tag{50}$$

That is, the binary adaptation outperforms Protocol 4 when there is only one erasure and we should simply use a parity bit, or when the word-size is very small.

Runtime Analysis In all three linear protocols, each step is quite simple to perform in a computer except solving a linear algebraic system of equations with t unknown variables. Actually, this can be done in $O(t^3)$, which is quite efficient. For "sparse

erasure" scenarios where $n \gg t$, the limiting steps are actually performing the matrix multiplications $\mathbf{H}x$ and $\mathbf{H}y$, each of which are $O(nt)$ (or $O(nc)$ for the probabilistic protocol). Between the three, one would expect Protocol 2 to take the longest, since it requires arithmetic in higher-order finite fields.

Figure 8 compares Protocol 2 (labelled "RS" and marked with circles) to Protocol 4 (labelled "RN" and marked with squares). To fairly compare the protocols, we use the binary adaptation of Protocol 2, and input the exact same strings x and y for each protocol. The paramter c for Protocol 4 is selected to produce the theoretically minimal effective communication complexity. Each point is the time to run the respective protocol, averaged over 10 trials. As expected, Protocol 4 consistently outperforms Protocol 2, by several orders of magnitude.

The blue points (labelled "10%") are when y contains $t = \lceil \frac{n}{10} \rceil$ erasures; the green points (labelled "25%") are when y contains $t = \lceil \frac{n}{4} \rceil$ erasures. When $t \approx n/10$ (blue points), the limiting steps should have an efficiency on the order of $O(nt) = O(n^2)$. The close relation to the curve[7] an^2 illustrates this. As t increases, the curve steepens, ultimately being approximately bounded by the curve an^3. Note that the standard curves an^2 and an^3 are for illustrative purposes only; they are *not* meant to match the data precisely.

Protocol 3 requires generating and selecting from a list of possible RM codewords. This also is not terribly inefficient, but the list length is $O(2^t)$, and so the protocol does not scale as well for many erasures.

[7] a is simply a small constant, roughly 2^{-19}, translating the standard curve down on the log plot.

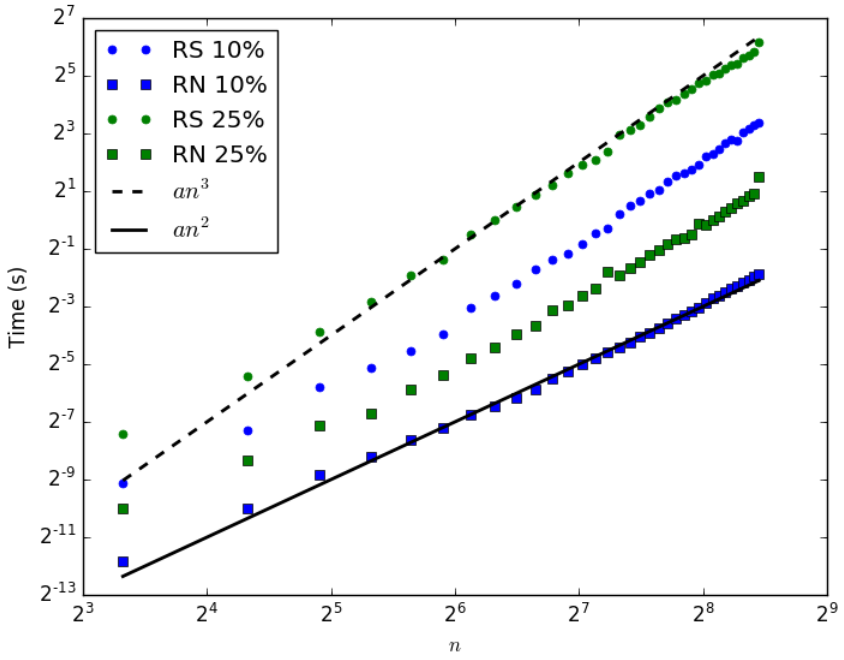

Figure 8: Average time to implement protocols

Further Work Consider the binary adaptation of Protocol 2 considered above. What if we allowed it to be probabilistic? The low-ish odds that t erasures all occur in different m-bit blocks may be enough to make the protocol more worthwhile. If we allow redundancy size c to vary, how does effective excess communciation complexity Δ compare to that of Protocol 4?

Much more theory can be explored with RM codes, to determine exactly for what parameters n and t there exist t bits from γ that Bob has no information on, and on which bits those are. A cursory examination suggests that for $\rho = 1$, selecting γ_i where $i = 2^\mu$, then $2^{\mu-1}$, then $2^{\mu-2}$ and so on is promising, allowing for an optimal protocol whenever $t \leq \mu$. However, this is inconsistent with our findings from Section 3.2: the $\rho = 1$ case is equivalent to a linear system, for which there were no solutions but $t \in \{0, 1, n-1, n\}$.

Finally, the transcendental equation for $\delta_{\min}$ in Section 3.5.2 is deeply unsatisfying. Is there a way to analytically approximate a value for $\delta_{\min}$? Perhaps we could even simply find a lower bound? With it, we could have a closed form approximaate expression for $\Delta(t)$ and perhaps see why it looks so similar, and potentially even bounds $1 + \log_2 t$.

Appendix

The computer code used in this project can be found on GitHub:

`https://github.com/ksherb1/information-reconciliation-Towson-thesis`

References

[1] Wade Trappe and Lawrence Washington. *Introduction to Cryptography with Coding Theory*. Prentice-Hall, Inc., 2002.

[2] Mary Boas. *Mathematical Methods in the Physical Sciences*. John Wiley & Sons, 3rd edition, 2006.

[3] David Griffiths. *Introduction to Quantum Mechanics*, pages 435–458. Pearson Education, Inc., 2nd edition, 2005.

[4] Claude Shannon. A modern theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[5] John Cover and Joy Thomas. *Elements of Information Theory*. John Wiley & Sons, 2nd edition, 2006.

[6] Florence MacWilliams and Neil Sloane. *The Theory of Error Correcting Codes*, volume 16. North-Holland Publishing Company, 3rd edition, 1981.

[7] David Slepian and Jack Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4):471–480, July 1973.

[8] S. Sandeep Pradhan and Kannan Ramachandran. Distributed source coding using syndromes (DISCUS). In *1999 IEEE Data Compression Conference, DCC'99*. IEEE, 1999.

[9] Nicolas Gehrig and Pier Luigi Dragotti. Symmetric and asymmetric Slepian-Wolf codes with systematic and non-systematic linear codes. *IEEE Communications Letters*, 9(1), January 2005.

[10] Daniyar Chumbalov. Combinatorial version of the Slepian-Wolf coding theorem for binary strings. *Sib. Èlektron. Mat. Izv*, 10:656–665, 2013.

[11] Daniyar Chumbalov and Andrei Romashchenko. Randomized polynomial time protocol for combinatorial Slepian-Wolf problem. In *Lecture Notes in Computer Science*, volume 9235, 2014.

[12] Djama Belazzougui and Qin Zhang. Edit-distance: Sketching, streaming, and document exchange. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE, October 2016.

[13] Zhenming Liu and Michael Mitzenmacher. Codes for deletion and insertion channels with segmented errors. In *International Symposium on Information Theory*. IEEE, 2007.

[14] Khaled Abdel-Ghaffar, Hendrik Ferreira, and Ling Cheng. Correcting deletions using linear and cyclic codes. *IEEE Transactions on Information Theory*, 56(10), October 2010.

[15] Guang Yang, Ángela Barbero, Eirik Rosnes, and Øyvind Ytrehus. Error correction on an insertion/deletion channel applying codes from RFID standards. In *Inormation Theory and Applications Workshop*. IEEE, 2012.

[16] Stark Draper. Universal incremental Slepian-Wolf coding. In *Proceedings of the 43rd annual Allerton Conference*, September 2004.

[17] F. Danashgaran, M. Laddomada, and M. Mondin. An iterative algorithm for compression of correlated sources at rates approaching the Slepian-Wolf bound: theory and analysis. *International Journal on Advances in Telecommunications*, 3(1), 2010.

[18] Alexander Kozachiniskiy. On Slepian-Wolf theorem with interaction. In *Lecture Notes in Computer Science*, volume 9691, 2015.

[19] Michael Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel Spielman. Efficient erasure correcting codes. *IEEE Transactions on Informtion Theory*, 47(2), February 2001.

[20] Shrinivas Kudekar, Santhosh Kumar, Marco Mondelli, Henry Pfister, Eren Şaşoğlu, and Rüdiger Urbanke. Reed-Muller codes achieve capacity on erasure channels. In *STOC'16 Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 658–669. ACM, 2016.

[21] Yunghsiang Han. BCH codes, 2010. Lecture notes.

[22] Chandan Saha and Markus Bläser. Reed-Muller codes, 2012. Lecture notes.

[23] Paulo Ferreira, Bruno Jesus, José Vieira, and Armando Pinho. The rank of random binary matrices and distributed storage applications. *IEEE Communications Letters*, 17(1):151–154, January 2013.

[24] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*, chapter 4. CRC Press, 5th edition, 2001.

[25] William Stallings. *Cryptography and Network Security*, chapter 7. Pearson Education, Inc., 6th edition, 2014.

[26] Dana Randall. Efficient generation of random nonsingular matrices. Technical Report UCB/CSD-91-658, EECS Department, University of Californina, Berkeley, November 1991.

Kyle Sherbert - Curriculum Vitae

Education

Towson University, Towson, MD

2015-2017 MS in Computer Science

GPA: 4.00

2011-2015 BS *cum laude* in Physics, Biology (Molecular) Biochemistry and Bioinformatics

GPA: 3.94

Study Abroad University of Strathclyde, Glasgow, Scotland

Honors University Honors Scholar (2011-2015)
Jess Fisher FCSM Scholar (2011-2015)
Maryland Distinguished Scholar (2011-2015)
Maryland Space Grant Consortium Scholar (2014-2015)
Edward L Rubendall Outstanding Physics Student (2012-2014)

Publications and Presentations

Information Reconciliation for Erasure Channels
May 2017 Thesis

Hello world - Code Responsibly!
In Progress Paper *Communications of the ACM*

Information Entropy of 1D Quantum Systems
May 2015 Poster *Towson University Research Expo*, Towson, MD

Surviving Abroad Without a Smartphone
Mar 2015 Poster *Celebration of Scholarship and Learning*, Towson, MD

Exploring Natural Product Formation with Structural Biology
Feb 2015 Talk *MB3 Club Weekly Seminar*, Towson, MD
Aug 2014 Poster *CTRC Summer Research Colloquium*, Buffalo, NY
July 2014 Talk *Hauptman-Woodward Institute Weekly Seminar*, Buffalo, NY

Computational Simulation of Electron Diffraction
May 2014 Poster *Towson University Research Expo*, Towson, MD
Mar 2014 Poster *APS March Meeting*, Denver, CO

Posters and papers are accessible through my personal website, kmsherbert.neocities.org/from-cv.html

Professional Experience

2015-Present Department of Computer and Information Sciences, Towson University
Teaching Assistant

Course	Instructor	Term
General Computer Science*	Cara Tang	Summer 2017
Introduction to Computer Science I	Andrea Calloway	Spring 2017
General Computer Science	Andrea Calloway	Spring 2017
General Computer Science*	Blair Taylor	Fall 2016
General Computer Science*	Cara Tang	Summer 2016
Introduction to Computer Science I	Blair Taylor	Spring 2016
General Computer Science	Blair Taylor	Spring 2016
General Computer Science*	Blair Taylor	Fall 2015

* Included an online component for high-school students in SPLASH program.

2015-Present Center for Talented Youth, Johns Hopkins University
Teaching Assistant

Course	Site	Summer
Cryptography	Los Angeles, CA	2017
Astrophysics	Lancaster, PA	2016
Data Structures and Algorithms	Lancaster, PA	2016
Genomics	Baltimore, MD	2015
Investigations in Engineering	Baltimore, MD	2015

2015-Present Department of Computer and Information Sciences, Towson University
Research Assistant

Faculty Supervisors: Siddharth Kaza and Blair Taylor

Developed and maintained Security Injections @Towson, a repository of online modules to teach secure coding practices in introductory CS courses. Organized and taught classes for SPLASH, a program for high-school girls to receive college-level education in CS. Designed, developed, and maintained the website for Towson University's Cyber4All initiative, using AngularJS, Wordpress and Django. Supervised undergraduate research assistants.

2015 Department of Physics, Astronomy, and Geosciences, Towson University
Research Assistant

Faculty Supervisor: Jia-An Yan

Numerically studied information entropy as a function of time in scattering quantum systems. Analytically solved for the information entropy for a free particle as a function of time. Replicated *Romera and de los Santos's* work in using information entropy to study fractional revivals in bound quantum systems. Presented work at 2015 TU Research Expo.

2014 CLIMB-UP Program*, University at Buffalo
Research Internship

* In partnership with the Bio-XFEL Summer Undergraduate Research Experience
Principal Investigator: Andrew Gulick
Laboratory Mentor: Geoffrey Lippa

Mutated, expressed, and purified a bacterial enzyme for use in crystallization experiments. Performed biochemical assays to determine function of enzyme. Analysed computational protein-structure models of enzyme. Presented work at 2014 HWI Weekly Seminar, CTRC Summer Research Colloquium, and 2015 MB3 Club Seminar.