HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communications Engineering
Networking Laboratory

Tuomas Tirronen

Optimizing the Degree Distribution of LT Codes

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, March 15, 2006

Supervisor: Professor Jorma Virtamo
Instructor: Esa Hyytiä, D.Sc. (Tech.)

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Tuomas Tirronen	
Name of the thesis:	Optimizing the Degree Distribution of LT Codes	
Date:	March 15, 2006	**Number of pages:** 74 + 7
Department:	Electrical and Communications Engineering	
Professorship:	S-38 Teletraffic Theory	
Supervisor:	Prof. Jorma Virtamo	
Instructors:	Esa Hyytiä, D.Sc. (Tech.)	

This thesis examines the problem of data transfer from the perspective of error correction codes. Recently, many new interesting transmission schemes employing erasure forward error correction have been proposed. This thesis focuses on codes approximating the "digital fountain", an analogy envisaging digital data as a fountain spraying drops of water, which can be collected by holding a bucket under it. The bucket eventually becomes full, regardless of the amount of water drops missing the bucket. In data transmission, the digital fountain functions in a similar fashion: packets are sent into network, and the recipient needs only a certain number of these packets to decode the original information. In practice, with good codes, this number is only slightly more than the amount of packets corresponding to the original file size.

Traditional Reed-Solomon codes can be used to approximate the digital fountain, but more efficient codes also exist. LT codes are efficient and asymptotically optimal codes, especially for a large number of source blocks. LDPC codes are also presented as an alternative for approximating the digital fountain.

Efficient utilization of LT codes requires carefully designed degree distributions. This work describes the distributions proposed earlier, and presents a new optimization method for generating good distributions. An iterative algorithm based on this method is also proposed. The optimization method is based on estimation of the average number of packets needed for decoding. The importance sampling approach is used to generate this estimate by simulating the LT process. After this, standard nonlinear optimization methods are employed to optimize this estimate. Numerical test results are provided to validate the correct function of the algorithm.

Finally, this thesis also includes a discussion of possible applications for erasure correcting codes approximating the digital fountain, with special attention to salient implementation issues.

Keywords: LT codes, erasure codes, forward error correction, importance sampling

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä	Tuomas Tirronen	
Työn nimi:	LT-koodien astelukujakauman optimointi	
Päivämäärä	15.3.2006	**Sivuja:** 74 + 7
Osasto:	Sähkö ja tietoliikennetekniikka	
Professuuri:	S-38 Teleliikenneteoria	
Työn valvoja:	Prof. Jorma Virtamo	
Työn ohjaajat:	TkT Esa Hyytiä	

Tämä työ käsittelee tiedonsiirtoon liittyviä kysymyksiä virheen korjaavien koodien näkökulmasta. Erityisesti käsitellään koodeja, jotka toteuttavat niin sanotun suihkulähdeperiaatteen. Suihkulähde suihkuttaa vesipisaroita ilmaan, joita voidaan kerätä asettamalla sanko suihkulähteen alle. Sanko täyttyy riippumatta siitä, paljonko pisaroita menee ohi tai mitkä pisarat sankoon osuvat. Samalla tavalla suihkulähdeperiaatteen mukaisessa tiedonsiirrossa tiedoston lähettäjä lähettää paketteja tietoverkkoon ja tiedoston vastaanottajan tulee kerätä tietty määrä lähetettyjä paketteja saadakseen lähetetyn tiedoston purettua. Sillä, mitkä paketit vastaanottaja saa, ei ole merkitystä. Hyvillä koodeilla tarvittavien pakettien yhteenlaskettu koko on vain vähän enemmän kuin alkuperäisen tiedoston koko.

Perinteisiä Reed-Solomon-koodeja voidaan käyttää suihkulähdeperiaatteen tavoin, mutta tehokkaampiakin koodeja on kehitetty. LT-koodit ovat tehokkaita ja asymptoottisesti optimaalisia koodeja, jotka toimivat erittäin hyvin kun lähdelohkojen lukumäärä on suuri. Myös LDPC-koodit esitellään lyhyesti yhtenä vaihtoehtona suihkulähdeperiaatteen toteuttamiseen.

LT-koodit tarvitsevat huolellisesti suunnitellun astelukujakauman toimiakseen tehokkaasti. Työssä esitellään kirjallisuudessa aiemmin ehdotettuja jakaumia ja esitetään uusi menetelmä astelukujakauman optimoimiseksi. Tämä menetelmä perustuu koodauksen purkuun tarvittavan keskimääräisen pakettien lukumäärän estimointiin. Estimaatti lasketaan tärkeysotantaan perustuvalla menetelmällä, ja tämän jälkeen estimaattia optimoidaan standardeilla optimointimenetelmillä. Työn lopussa esitetään algoritmilla laskettuja numeerisia testituloksia.

Lisäksi työssä ehdotetaan sovellusalueita esitetyille koodeille sekä pohditaan ongelmia, joita näitä koodeja käytettäessä on huomioitava.

Avainsanat: LT koodit, virheen korjaavat koodit, tärkeysotanta, suihkulähdekoodit

Acknowledgements

This thesis was written in the Networking Laboratory of Helsinki University of Technology for the PAN-NET project. Work for this thesis was mainly carried out in the last months of 2005 and beginning of 2006.

First of all, I would like to thank my supervisor, professor Jorma Virtamo for providing the subject and his invaluable comments and help. Also many thanks to my instructor, Esa Hyytiä, who is currently at Norwegian University of Science and Technology (NTNU), for his help and many corrections and suggestions for making this thesis better.

I would also take the opportunity to thank all the personnel in the Networking Laboratory for providing nice and friendly working atmosphere. Special thanks go to everyone in the lab's Wednesdays floorball team for keeping me at least partly fit.

I owe a great gratitude to my family for their support during my studies. Finally I would like to thank all my friends and especially my girlfriend Laura for all her love.

Espoo, 15th March 2006

Tuomas Tirronen

Contents

Abbreviations

3GPP	3rd Generation Partnership Project
ALC	Asynchronous layered coding
API	Application programming interface
ARQ	Automatic repeat request
AWGN	Additive white Gaussian noise
BEC	Binary erasure channel
BSC	Binary symmetric channel
CRC	Cyclic redundancy check
ECC	Error correcting coding
FEC	Forward error correction
IETF	Internet engineering task force
IS	Importance sampling
ISG	Importance sampling gradient (algorithm)
LDPC	Low-density parity-check (codes)
LT	Luby transform
MDS	Maximum distance separable
MLD	Maximum likelihood decoder
NACK	Negative acknowledgment
NASA	National Aeronautics and Space Administration
NORM	Negative-acknowledgment-Oriented Reliable Multicast (protocol)
P2P	Peer to peer (networking)
PCO	Pre-coding only
PEC	Packet erasure channel
PRNG	Pseudo-random number generator
RAID	Redundant array on inexpensive disks
RFC	Request for comments
RSE	Reed-Solomon erasure (code)
TCP	Transmission control protocol

Notation

$\boldsymbol{c}, c_i$	Codeword vector and components
$c(x)$	Codeword polynomial
d_c	Degree of check node
d_m	Degree of message node
f	Overhead factor
$\boldsymbol{g}$	Gradient. Subscript text is used to denote context.
$g(x)$	Generator polynomial
h	Size of header
$\boldsymbol{m}, m_i$	Message vector and components
M	Total data size
$m(x)$	Message polynomial
n	Number of blocks in source message
$n_i^{(k)}, n_i^{(t)}$	Number of packets of degree i in simulated sample k, or at time t
p_f	Channel error probability
s_i	Input symbol
s_o	Output symbol
w_i	Importance ratio
A_i	Input alphabet
A_o	Output alphabet
C	Channel capacity
$E[\cdot]$	Expectation
$\mathcal{C}$	(Error correcting) code
$\mathbf{G}$	Generator matrix
$GF(p^m)$	Galois field with size of p^m
$\mathbf{H}$	Parity check matrix
$\mathcal{P}, \mathcal{Q}$	Probability
$\widehat{R}$	Estimate for the number of packets needed for decoding to succeed
R_k	Number of packets needed for decoding in simulation sample k
$\mathbf{X}$	General random variable
$\rho(d)$	Degree distribution, also p and q are used
$\boldsymbol{\eta}, \boldsymbol{\theta}$	Vectors of parameters defining a degree distribution
$\Omega(x)$	Generator polynomial for degree distribution
$\exists!$	Exists unique

Chapter 1

Introduction

One of the most popular applications on today's Internet is the transfer of large amounts of data from one point to many recipients, or even in a mesh-like structure from many senders to multiple receivers. The networking technologies evolve at high speed, enabling more bandwidth to be used by home and office users troughout the world, but at the same time the size of the data utilizing this growing capacity is increasing.

Efficient methods are needed for this basic application of data transfer. Traditional TCP/IP protocols are not sufficient for applications where several hundred megabytes or even gigabytes of data is transferred, especially if there is more than one recipient. In particular, if the packets in transit have high loss rates, i.e., several of the packets are destroyed by the channel, the performance of traditional protocols, where each individual asks explicitly for the missing packets is poor. This loss could be caused by network congestion or by a poor quality link between network nodes.

The demand for good distribution mechanisms is a hot issue for many enterprises searching ways to distribute content to possible customers. Movie industry is planning to do the same thing that is the current trend in music industry, that is, selling and distributing the music in many Internet stores.

1.1 Erasure Codes

Interesting, and recently much researched, alternatives to the traditional transmission techniques are the different forward error correction (FEC) schemes based on the erasure codes, and transfer protocols supporting these codes. This approach is presented for example in [9], where the authors propose a *digital fountain* approach to data distribution. This means distributing pieces of a file like water drops from a fountain; anybody wishing to fill his bucket needs to place it under the fountain, and eventually there is enough water to satisfy the bucket holder.

The functionality of a digital fountain can be approximated by different erasure coding methods. Traditionally erasure codes like Reed-Solomon codes are employed at low levels, by correcting errors in physical media and demanding network links such as satellite communications, where the error correction is usually implemented directly in the hardware. These codes can nonetheless be used at application level, notably these days,

when even personal computers have remarkable computing capacities [41, 35].

The consensus has been that the FEC methods could be utilized also in Internet protocols, but not until recently actual implementations have evolved. IETF[1] has reliable multicast working group working on the issues of FEC coding in Internet communication protocols, and they already have provided frameworks of the protocols using FEC codes as one component [26].

The theory of erasure codes is constantly evolving, as better codes and new methods are invented. The practical aspects of different methods, however, need more studies and performance evaluations [38].

1.2 Problem Statement

The class of codes which we especially focus on are the LT codes [27]. These codes need a degree distribution for operation, and the only factor affecting the performance of the LT codes is this distribution. While some rather efficient distributions have been presented in the literature, we believe that better ones can be found. Especially in the range of files consisting of $n < 100$ blocks, the previously published distributions do not work very well.

The main part of this thesis discusses a method for optimizing the degree distribution used with LT codes. As this distribution is the only factor affecting the performance of LT coding, it is thus of profound importance to find the optimal forms of this distribution. We propose a method, which utilizes mathematics borrowed from the so-called importance sampling theory, to generate an estimate of the average number of packets needed for decoding a message sent using LT coding as a function of the degree distribution. This estimate is then used with different nonlinear optimization methods to produce better degree distributions. Based on this, an algorithm, which iteratively improves the degree distribution is proposed. We take the approach of starting from low number of file blocks n, and see how our method scales when this value grows. The operation of the algorithm is verified with some numerical results for example cases.

While our main contribution is the derivation of this algorithm, this thesis also contains discussion of possible applications and implementation issues for erasure correcting codes in general.

1.3 The Point of View

The discussion in this thesis is presented from network point of view. This means that we do not focus on the vast amount of theoretical results in coding theory. Instead of exploring graph theory to optimize and present the properties of different codes, we use a practical point of view, focusing on the general properties of current state-of-art code types and some numerical optimization results on LT codes, one very efficient erasure

[1] Internet Engineering Task Force. International community of designers and developers, open to all participants. IETF is divided into different working groups which deal with different areas of internetworking. See http://www.ietf.org.

coding scheme. We also assume that the packets received are correct, i.e., a mechanism to drop all packets corrupted by bit errors exists at the link level.

FEC, FEC erasure and erasure codes are largely referring to the same type of codes throughout this work, although in reality these do not mean exactly the same thing. Nonetheless, from high level networking point of view these terms refer to codes exhibiting the same function, so no damage is done by using these terms interchangeably in this context.

1.4 Outline of the Thesis

In Chapter 2 we introduce the basic key concepts and definitions of information and coding theory. The approach is not to extensively cover the vast field of information theory, but to define the concepts used in this work.

Chapter 3 introduces different FEC erasure coding schemes. Traditional example is given in the form of Reed-Solomon codes, after that the state-of-the-art codes are presented, including the LT codes, which are the main topic of this thesis.

Some of the different applications for the codes described in Chapter 3 are discussed in Chapter 4.

In Chapter 5 we discuss some implementation issues, which need to be addressed when doing an actual implementation of any of the presented coding methods. We also present a simple model for calculating the optimal block sizes in this chapter.

Our main contribution is presented in Chapter 6, where we propose an optimization algorithm for the degree distribution used in LT code. This chapter presents the derivation of the algorithm with some mathematical background behind it.

Some numerical examples and discussion of the performance of the developed algorithm are given in Chapter 7. Finally, Chapter 8 contains the conclusions.

Chapter 2

Background in Information Theory

This chapter presents basics in information and coding theory, which are prerequisites for understanding the later chapters. Definitions and notation are presented together with some examples of basic codes. Emphasis is on the concepts which are important and relevant for this work. Information theory itself is here considered to be an umbrella theory which contains coding theory as a part. The presentation here is only a scratch of the surface on these subjects, a plethora of books have been written on information and coding theory; references used in this work are [30, 47, 33].

2.1 Transmission of Data

The theme of this thesis is closely connected with efficient transmission of data in a computer network. Networks represent the information in digital form. This means that there is a finite amount of possible symbols used to characterize the information. In computer networks the information is passed trough different kinds of links from the source to one or more recipients. These links in information theoretical terms are *channels*. Several different channel models exists, some have more theoretical uses but models exist also for practical purposes.

Many, if not all, books on information theory and channel coding related issues have similar presentation as Figure 2.1 as the "first picture". Figure 2.1 shows a basic scheme for information transmission through a channel. The data source generates messages, i.e., blocks of information to be transmitted to receivers. A generated message ultimately has an analog or digital signal form and goes trough a transmitter which performs some fundamental operations on the signal. The two main operations such a transmitter carries out are modulation and encoding. Modulation is used to transform the message to suitable form for a particular transmission channel. This results in an efficient and suitable signal form which hopefully minimizes the interference and noise which the channel might incur on the signal. The aim is to enable efficient transmission of the signal over the given channel type. Modulation is not, however, the subject of this work and for a deeper treatment of modulation issues a good reference is [10].

The coding function performed by the transmitter is more relevant to this work. Coding theory, i.e., the science studying different codes and the mathematical framework

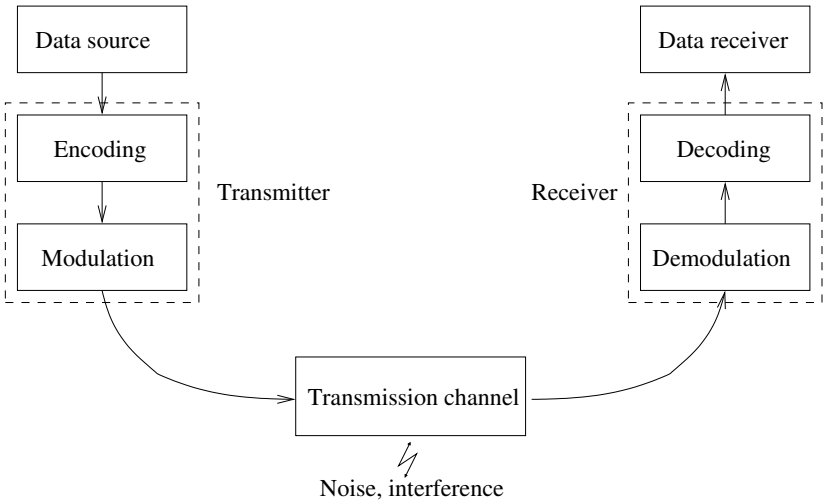

Figure 2.1: Transmission channel with transmitter and receiver. The main functions of transmitter and receiver is to modulate and encode or demodulate and decode data.

behind them will be discussed in the next section.

2.2 Objectives of Coding and Coding Theory

As the name suggests, coding theory deals with codes. Code itself is a rule used to convert information into some other form. Codes can be divided into two main categories: Source codes and channel codes. Both classes in general deal with *redundancy*. A message with minimal amount of information to express something does not have redundancy and a message conveying the same information with extra symbols does have redundant symbols. The main difference between the two code classes is that source codes try to get rid of all the redundancy to compress the information and channel codes usually introduce redundant symbols into messages in order to make the information transmission trough a channel less prone to possible errors, and to implement ways to detect and correct them.

The coding block in Figure 2.1 consists of both source and channel coding. Source coding is used to make the data take less space by removing unwanted and uncontrolled redundancy in data, channel coding is used after this to code the message symbols in desired way by usually adding redundancy in a controlled way. Also encryption and decryption of data, if used, are functions of coding block and usually take place between the source and channel coding operations.

The category of channel coding is also called error-correcting coding (ECC) and is the focus of this work. In particular the class of *erasure* ECC is studied. ECC is traditionally employed on the link layer level in communication networks, however, this work deals with end-to-end enabled ECC schemes which work on transport and higher levels, i.e., software-based ECC.

2.3 Channel Models

As real channels are noisy, errors might be introduced into the streams of transmitted information. Therefore modulation and coding are used to minimize the probability for loss of transmitted information. Hence the model of the channel used in the transmission is important and receives some treatment of its own. Particularly, a special type of erasure channel is central to the subject of this work.

This section presents few basic channel models and discusses some of their properties and applicability in real networks. Information is in the form of messages composed of binary digits, i.e., bits as is usual for digital communication.

A channel can be described by giving the set of symbols which it accepts and the set of symbols it outputs, called respectively *input* and *output alphabets* and denoted A_i and A_o.

Definition 2.1 (Binary symmetric channel)**.** *A **binary symmetric channel (BSC)** has input alphabet $A_i = \{0, 1\}$ and output alphabet $A_o = \{0, 1\}$. A symbol is transmitted independently without an error with probability $1 - p_f$ and transmission fails (changes the symbol) with probability p_f.*

BSC is a good basic channel model for number of situations. In the usual case, when channel noise is assumed to additive white Gaussian noise (AWGN), and the transmission is digital, we can use BSC model to describe the characteristics of the channel. The error probability p_f can then be calculated using complementary error function [30]. BSC assumes that the symbol errors are independent, this is not always the case in real channels, as the bit errors could occur in bursts.

The special erasure channel mentioned above determines when there is some problem with the transmitted symbols. In other words, this means that possible errors in transmission are somehow noticed by the channel itself.

Definition 2.2 (Binary erasure channel)**.** *A **binary erasure channel (BEC)** has input alphabet $A_i = \{0, 1\}$ and output alphabet $A_o = \{0, 1, ?\}$. A symbol is transmitted correctly with probability $1 - p_f$ and output is symbol $?$ with probability p_f.*

In BEC, the symbol $?$ represents the case when something has gone wrong and transmitted symbol has changed. Figure 2.2 depicts both BSC and BEC and shows the conditional probabilities of possible output symbols given the input symbol. In the BEC case it is thus not possible to have an output symbol which is the opposite of input symbol; in case of an error, $?$ is the output symbol.

In networks it is common to call the sent data blocks packets. A special type of erasure channel is defined next.

Definition 2.3 (Packet erasure channel)**.** *A **packet erasure channel (PEC)** either transmits sent packets correctly with probability $1-p_f$ or, in case of an error, drops the packet with probability p_f.*

In particular, the Internet can be modeled using PEC type of channels. If bit errors are introduced into packets in transmission, it can be assumed that network nodes notice

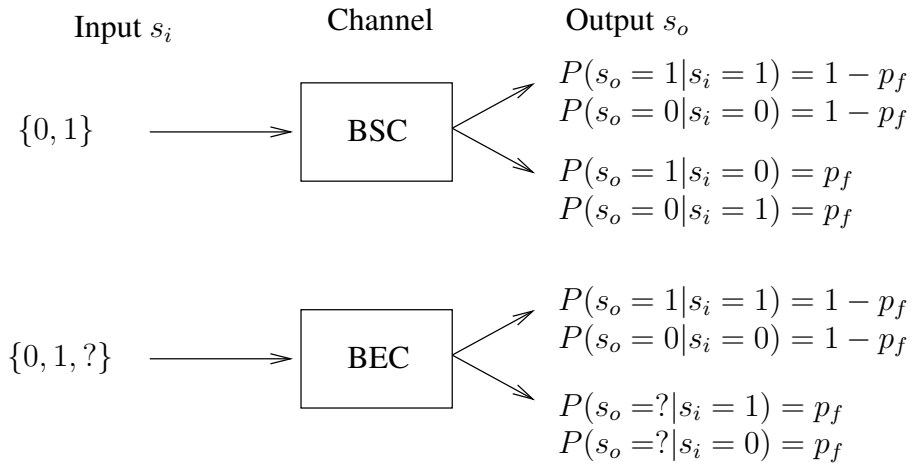

Figure 2.2: BSC and BEC channel models. Input symbols s_i are transferred either correctly or incorrectly, and the corresponding conditional probabilities for possible output symbols s_o are presented on the right side.

the errors and drops the erroneous packets. Usually schemes like CRC[1] computing and comparison with header data are made an possible transmission are noticed this way. The other major cause for packet loss is network congestion. Heavy traffic exceeding the capabilities of network nodes leads to overflows in buffers and as a result packets are dropped.

PEC is the channel type that is assumed in this work. The presented codes are all usable in BSC channels and erasure channels, but often are much simpler in erasure channels as the actual error correction does not have to be made. The received packets can be assumed to be error-free, as erroneous packets are dropped by mechanisms implemented elsewhere.

2.4 Basics of Error Correcting Codes

2.4.1 Different Ways to Implement Error Correction

Error correction can be implemented in multiple different ways and at different levels. The traditional scheme used in the Internet for end-to-end connections is to resend the missing pieces. This means that missing blocks of data are sent again by the source. This type of error correction works in some applications fairly well. However, in some situations it is better to employ a different scheme, as resending generates feedback from the destination to the source and in some situations this feedback can become too excessive to the source to handle for. Especially when multicasting the data to multiple sources, every recipient asking for a particular missing packet can be catastrophic for the transmission process.

[1]Cyclic redundancy check (CRC) is a common type of hash function checksum calculation method used to detect and correct errors. Checksum is calculated before transmission and is compared to calculated checksums at intermediate nodes.

The previous example is a higher level form of automatic repeat request (ARQ), a method which is usually employed at link level between two network nodes. In ARQ, the receiver explicitly asks for retransmission of such blocks where errors have been detected.

Another way to deal with errors is to simply drop the erroneous data and cope with what is available. This scheme can be used in some cases when transmitting streaming or analog data, for example in speech transmission. Also in many real-time situations, the data arriving late is completely useless to the recipient. This does not, however, work with digital data transmission, where the transferred information has ultimately to take exactly the same form at both ends of communicating parties. The method of just dropping pieces of information with errors is called *muting*.

The error control scheme considered further in this work is forward error correction (FEC). In FEC, data is encoded in a way such that based on the erroneous received data, the receiver can use probabilistic analysis to determine what the received data most likely should be. FEC can be employed on multiple levels in communications. Traditionally, FEC codes are implemented directly in hardware and thus work at the link layer level between two adjacent network nodes. Software based end-to-end FEC is not yet widely deployed, but because of the increasing processing capability of desktop computers and development of efficient codes, the obstacles are not significant anymore. Software FEC would thus be a reasonable choice for some applications [40]. Implementation issues of software FEC are further discussed in Chapter 5.

2.4.2 Simple Error Correcting Codes

We continue with some useful definitions. The basic model presented if Figure 2.1 involves a source who wants to send some information to some receiver.

Definition 2.4 (Message, symbol)**.** *Let m denote the **message** a source wants to transfer. Individual pieces of a message are called **symbols**. The message can be represented as a message vector $\boldsymbol{m}$.*

An example of a message vector could be $\boldsymbol{m} = (0 \quad 0 \quad 1)$, consisting of three symbols. These messages are encoded prior to transmission using some *code*:

Definition 2.5 (Code, codeword)**.** *A **code** is a set of rules for transferring data into another form. Equivalently a code is the set $\mathcal{C}$ of all legal **codewords** c. A codeword c is generated from the message m with the specified rules. Codewords can be represented by codeword vectors $\mathbf{c}$ similarly as messages. **Encoding** is the process of creating codewords from messages and **decoding** is the process of transferring codewords back into the form of the original data.*

In particular error-correcting codes are such that they can detect and correct possible errors in transmitted data. Perhaps the most common code used to detect errors is called *parity*. Parity is a single bit added usually at the end of a codeword indicating whether there is an odd or even number of ones in a particular piece of data. Usually parity bit 1 means that there is an odd number of ones and 0 is used to indicate even number of ones.

Example 2.6 (Parity)**.** *We have original message $m = (0 \quad 0 \quad 1)$ which we want to transfer using parity error-detecting code. Thus the encoder calculates the number of ones in the original message and adds a parity bit accordingly, here the codeword is $c = (0 \quad 0 \quad 1 \quad 1)$. Now if we transfer this codeword over BSC, there is some probability p_f independently for every transmitted symbol to change. Now if the received codeword at the receiver side is $\tilde{c} = (0 \quad 0 \quad 0 \quad 1)$, the receiver knows that there has to be an error somewhere because the number of ones is odd and we are using even parity. Unfortunately parity code is only an error detecting code and we can neither know where the error is nor correct it. Also if there is an even number of bit flips, the parity remains the same and the error cannot even be detected. Thus it is easy to see that using only parity is probably not a safe bet. Of course, in situations where p_f is very small, the probability for two bit flips p_f^2 is negligible, so in situations the probability for channel errors is very low and rare errors are not critical, the use of parity only could be a sufficient error detecting method.*

The simplest form of error-correcting code is *repetition* code. Here we simply repeat every sent symbol N times and use a majority vote at the receiver end to decode the transmitted codeword.

Example 2.7 (Repetition)**.** *Let us use the same message $m = 001$ as before. Using repetition with $N = 3$, the sent codeword is then $c = 000000111$. If the receiver gets the word $\tilde{c} = 001010111$, he can assume that first two symbols should be zeros and the last one should be one. The downside of this scheme is that the length of the sent codeword is three times as long as the original message. A double error in one repeated symbol is detected but generates a false outcome when majority vote decoding is used. Repetition is error-correcting code, as it can detect and correct errors.*

The parity code has not the ability to correct errors, but is efficient overhead-wise; only one symbol is needed in addition to the symbols in original message. Repetition has the desirable error-correction ability, but every symbol has to be repeated many times. We define the *rate* of a code as follows:

Definition 2.8 (Rate)**.** *When the original message length is n source symbols and it is encoded using a code which produces k encoding symbols, then the **rate** of the code is $R = \frac{n}{k}$.*

Usually the reciprocal of rate is defined to be the *overhead factor* f, but we use a different definition for the purposes of this work:

Definition 2.9 (Overhead factor)**.** *If the original length of a message is n symbols, and the receiver needs to collect $n' \geq n$ encoding symbols (packets) to decode the original message, then the **overhead factor** is $f = \frac{n'}{n}$. This definition implies that $f \geq 1$.*

We will see in the next chapters that it is convenient to separate the overhead caused by encoding redundancy, and the overhead caused by the number of received packets. Depending on the used code, the amount of redundant information transferred might depend on the used rate R or on the overhead factor f or on combination of both of these.

In our parity example, the rate is $R = \frac{n}{n+1} = \frac{3}{4}$, and in repetition example $R = \frac{1}{3}$, which is worse than in parity case. Next we discuss a little bit of the theory behind the error detection and correction capabilities of codes in general.

Definition 2.10 (Hamming distance)**.** ***Hamming distance*** d_H *of two vectors* $\boldsymbol{c_1}$ *and* $\boldsymbol{c_2}$ *is the number of differing symbols of these two messages, i.e,*

$$d_H(\boldsymbol{c_1}, \boldsymbol{c_2}) = \left|\{\ i \mid c_1^i \neq c_2^i, 0 < i \leq k\ \}\right|. \tag{2.1}$$

Minimum Hamming distance d_{min} *of a code* $\mathcal{C}$ *is useful when describing the error detection and correcting capabilities:*

$$d_{min}(\mathcal{C}) = \min_{\boldsymbol{c_1}, \boldsymbol{c_2} \in \mathcal{C}} \{d_H(\boldsymbol{c_1}, \boldsymbol{c_2}) | \boldsymbol{c_1} \neq \boldsymbol{c_2}\}. \tag{2.2}$$

It can be shown [47] that a code with a minimum Hamming distance d_{min} can

1. Detect up to $l = d_{min} - 1$ errors per codeword.
2. Correct up to $t = \lfloor \frac{d_{min}-1}{2} \rfloor$ errors per codeword.

With parity code, we see that $d_{min} = 2$, as two different valid codewords have to differ in two different bits, otherwise the parity would change. Thus we can detect $l = 2-1 = 1$ error in a codeword and correct $t = \lfloor 1/2 \rfloor = 0$ errors as was discussed in Example 2.6.

Only valid codewords in $N = 3$ repetition code are 000 and 111, the minimum distance is then $d_{min} = 3$. Now $l = 3 - 1 = 2$ errors per codeword can be detected and $t = 1$ errors corrected, which is in agreement with our discussion in Example 2.7.

2.4.3 Linear Block Codes

We want to have the error correcting capability of repetition codes but with a rate like in the parity codes. A more advanced class of codes than the basic schemes presented above are linear block codes. An (n, k) block code is a code where length of data blocks is n symbols and these blocks are encoded to codewords of k symbols. Thus the rate of such a code is $R = n/k$.

A **linear** code means that if two codeword vectors are part of the code then also the sum of these vectors belongs to the same code. Also the zero vector belongs to a linear code. When symbols are bits the summation is performed using modulo-2 arithmetic, i.e., in $GF(2)$, see Appendix A. In general, codeword symbols can be elements of some $GF(q)$.

Linear codes have one useful property: the minimum Hamming distance d_{min} is the same as the number of symbols other than zero in a non-zero codeword.

Definition 2.11 (Hamming weight)**.** ***Hamming weight*** $w_H(c)$ *is the number of non-zero symbols in* c*.*

This means that $d_H(\boldsymbol{c}, 0) = w_H(\boldsymbol{c})$ and further for linear codes

$$d_H(\boldsymbol{c_1}, \boldsymbol{c_2}) = d_H(\boldsymbol{c_1} + \boldsymbol{c_2}, 0) = w_H(\boldsymbol{c_1} + \boldsymbol{c_2}), \tag{2.3}$$

where $\boldsymbol{c_1} + \boldsymbol{c_2} \in \mathcal{C}$, thus

$$d_{min}(\mathcal{C}) = \min_{\boldsymbol{c} \in \mathcal{C}} \{w_H(\boldsymbol{c})\}. \tag{2.4}$$

Definition 2.12 (Generator matrix, parity check matrix)**.** *A generator matrix* $\mathbf{G}$ *of a code can be used to calculate codewords:*

$$\boldsymbol{c} = \boldsymbol{m}\boldsymbol{G}, \tag{2.5}$$

where $\mathbf{c}$ *is the codeword vector and* $\boldsymbol{m}$ *the message vector. A parity check matrix* $\mathbf{H}$ *is such that* $\mathbf{G}\mathbf{H}^\mathsf{T} = \mathbf{0}$*, this means that for all codewords* $\mathbf{c} \in \mathcal{C}$

$$\boldsymbol{c}\mathbf{H}^\mathsf{T} = \mathbf{0}. \tag{2.6}$$

A *systematic* code is such where the codeword has the original message blocks intact at the beginning, and redundant information is added at the end of the original information. A systematic form of a linear block code can be constructed by generator matrix

$$\mathbf{G} = (\mathbf{I_n}|\mathbf{P}), \tag{2.7}$$

where $\mathbf{I_n}$ is a $n \times n$ identity matrix and $\mathbf{P}$ is a $(n) \times (k - n)$ matrix. If $\mathbf{G}$ is in systematic form, then the parity check matrix is easy to calculate:

$$\mathbf{H} = \left(\mathbf{P}^\mathsf{T}| - \mathbf{I}_{k-n}\right). \tag{2.8}$$

Note that if we consider binary codes, then $-\mathbf{I} = \mathbf{I}$.

2.4.4 Hamming Codes

Now we will look at one class of linear block codes called Hamming codes. Hamming codes require the smallest possible amount of redundancy for a given block length to correct any single error. Parameters of Hamming codes are for integer $m \geq 2$:

1. Length of the code, $k = 2^m - 1$.
2. Number of information symbols, $n = 2^m - m - 1$.
3. Number of parity symbols, $m = k - n$.
4. Error correcting capability, $t = 1$.

For a given set of parameters, a parity check matrix $\mathbf{H}$ for Hamming code can be constructed by setting all possible non-zero binary vectors of length m as columns.

Example 2.13 (Hamming code)**.** *Assume the code length* $k = 2^3 - 1 = 7$*, number of information symbols* $n = 2^3 - 3 - 1 = 4$ *and number of parity symbols* $m = 7 - 4 = 3$*. Parity check matrix is then:*

$$\mathbf{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

This has the form $\left(\mathbf{P}^{\mathsf{T}}|\mathbf{I}_3\right)$. *The generator matrix can be obtained by* (2.7)*:*

$$\mathbf{G} = (\mathbf{I}_4|\mathbf{P}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

2.5 Principles of Decoding of Linear Block Codes

2.5.1 Decoding in general

The decoder has to validate the received codeword $\boldsymbol{r}$, that is, a decision has to be made to decide if the codeword belongs to the used code $\mathcal{C}$. This is the error detection function, and can be implemented as ARQ, FEC or muting as described in Section 2.4.1.

Linear block codes described here are FEC codes, and the receiver simply checks if the received codeword $\boldsymbol{r}$ belongs to the set of all legal codewords $\mathcal{C}$. If this is the case, then the transmitted codeword is assumed to be $\boldsymbol{c} = \boldsymbol{r}$. The original message $\boldsymbol{m}$ is decoded using the correspondence between $\boldsymbol{m}$ and the codeword $\boldsymbol{c}$ according to (2.5).

If, on the other hand, the codeword is modified by some error pattern $\boldsymbol{e}$, then the received codeword is $\boldsymbol{r} = \boldsymbol{c}+\boldsymbol{e}$ and the task of the decoder is to determine the underlying codeword $\boldsymbol{c}$. This can in general be done in two ways, either using maximum a posteriori decoder, where the chosen codeword is the one which maximizes the the probability of the codeword $\boldsymbol{c}$ conditioned on the received codeword $\boldsymbol{r}$. The other method is to maximize the probability of the received codeword $\boldsymbol{r}$ conditioned on the probability of the codeword $\boldsymbol{c}$. This latter decoder is the maximum likelihood decoder (MLD) and is the one considered here.

The MLD decoder decodes a received codeword $\boldsymbol{r}$ by relating it with the closest codeword in the code in the sense of Hamming distance (2.2). Thus, it finds $\boldsymbol{c}$ which minimizes $d(\boldsymbol{r}, \boldsymbol{c}_i) = w(\boldsymbol{r} - \boldsymbol{c})$. If the received codeword is closer to an incorrect codeword, then the MLD decoder makes a *decoder error* and corrects the received codeword wrong. With every possible error-correcting code, there is the possibility that the error is corrected wrong, a situation which needs to be considered when implementing a ECC scheme. The goal is naturally to make this error probability as small as possible.

Definition 2.14 (Complete error correcting decoder)**.** *A* ***complete error correcting decoder*** *selects the codeword* $\mathbf{c}$ *closest to received codeword* $\mathbf{r}$*, that is, minimizes* $d(\mathbf{r}, \mathbf{c})$.

This method leads to a design which can detect all error patterns with weight less than or equal to $d_{min} - 1$. Similarly an error pattern with weight less than or equal to $\lfloor (d_{min} - 1)/2 \rfloor$, because a decoder error occurs if the received codeword is closer to an incorrect codeword, that is, when the distance from the correct codeword is greater than $d_{min}/2$.

2.5.2 Syndrome Decoding

A general and standard way of decoding linear codes is called syndrome decoding which constitutes of calculating a syndrome vector and then decoding the received codeword by a look-up table. The parity check matrix introduced in Definition 2.12 is used here. The property of all legal codewords $\boldsymbol{c}$ satisfying the equation $\boldsymbol{c}\mathbf{H}^\mathsf{T} = \mathbf{0}$ is the key factor in calculating the syndrome vector $\boldsymbol{s}$:

$$\boldsymbol{s} = \boldsymbol{r}\mathbf{H}^\mathsf{T} = (\boldsymbol{c} + \boldsymbol{e})\mathbf{H}^\mathsf{T} = \boldsymbol{c}\mathbf{H}^\mathsf{T} + \boldsymbol{e}\mathbf{H}^\mathsf{T} = \mathbf{0} + \boldsymbol{e}\mathbf{H}^\mathsf{T} = \boldsymbol{e}\mathbf{H}^\mathsf{T}.$$

So the syndrome vector $\boldsymbol{s}$ is function of the error pattern $\boldsymbol{e}$, and also unambiguous with respect to different error patterns meaning that different error patterns have different syndrome vector.

Using the syndrome vectors we can tabulate all error patterns associated with different syndrome vectors and use a table look-up to decode received codewords. However, this method is not very scalable as large codes need large tables thus placing memory requirements which can be hard to fulfill.

2.6 Shannon Limit for Noisy Channels

There is a trade-off between the decoder error probability and the rate of a code. Using lower rates lead to design of codes which can correct error patterns with greater weights, as can be seen from the previous discussion on the linear block codes. This would seem to reason that while we lower the rate of some code, the decoder error probability can be made arbitrary small, and finally, at the limit when the rate goes to zero, the error probability also approaches zero.

There is, however, a certain point where the communication succeeds at zero *bit error probability* p_b with non-zero rate R. Bit error probability is the average probability that a decoded bit does not match the corresponding message bit. This result was formulated by information theory pioneer Claude Shannon [43]. The maximum rate we can communicate over a certain channel with arbitrary small p_b is called the capacity C of the channel.

Theorem 2.15 (Noisy Channel Coding Theorem)**.** *For a channel with capacity C there exists a coding system such that for rates $R < C$ the information can be transmitted with arbitrary small amount of errors. For $R > C$ it is not possible to transmit the information without errors.*

This is simple form of this theorem, for complete discussion and proofs, see [30].

For the binary symmetric channel the capacity C can be calculated as follows

$$C(p_f) = 1 - p_f \left[log_2 \frac{1}{p_f} + (1 - p_f) \frac{1}{1 - p_f} \right], \tag{2.9}$$

where p_f is the channel error probability as defined earlier.

2.6.1 Optimality of Error Correcting Codes

Shannon's results prove that reliable codes for different channels exist. The problem is to find these codes. However, the noisy channel coding theorem can be used to prove that good block codes exist for any noisy channel, but the decoding would probably require a table look-up procedure that is not computionally efficient. The real problem is to find efficient encoding and decoding methods to make it actually worthwhile to use these codes.

A code with optimal properties for a given channel would have low encoding and decoding complexities and a rate that achieves the channel capacity. Different tricks to implement codes with better encoding and decoding complexities exist, including for example convolutional codes, concatenation of codes, interleaving and so on. Discussion of these and many other code types can be found in any good book on error-correcting codes or information theory, see e.g. [30].

A particular category of codes which achieve the capacity at the limit, and are also very efficient with finite number of message blocks are *digital fountain codes*, which are presented in Section 3.3.

The best known codes for Gaussian channels, i.e., AWGN channels with real valued input and output, are LDPC codes, which are presented in Section 3.2. These codes can also be utilized as erasure codes in discrete channels, for example the packet erasure channel, enabling similar functionality as with digital fountain codes, as we will see in the next chapter.

Chapter 3

FEC codes for erasure channels

In this chapter FEC codes for erasure channels are presented. In this chapter we present the basic properties and functions of the codes, a more thorough treatment can be found in the references. Especially erasure correcting properties are emphasized and error correcting details are deliberately left out. First Reed-Solomon codes and Low-Density Parity-Check (LDPC) codes are discussed, and after that the ideas and the inner workings of the digital fountain codes are explained, with focus on the LT codes.

3.1 Reed-Solomon codes

Reed-Solomon codes were presented in 1960 [39] and are still widely used in many different applications varying from compact discs and other storage devices to computer networks and space communication. It still is one of the most popular FEC coding schemes. The Reed-Solomon codes are non-binary cyclic linear block codes. Cyclic codes are such where every codeword can be cyclically shifted, and the resulting word is also a valid codeword. This means that if

$$c = (c_1 \quad c_2 \quad \dots \quad c_n)$$

is a codeword in $\mathcal{C}$, then also

$$c^{(1)} = (c_n \quad c_1 \quad c_2 \quad \dots \quad c_{n-1})$$

is a valid codeword of the same code. Reed-Solomon codes are also part of the large class of algebraic codes.

With Reed-Solomon codes, we have to fix beforehand the rate, i.e., the amount of redundant information we are going to use. The rate is always less than one, resulting in transfer of some redundant information. However, the overhead in terms of extra packets is zero, i.e. $f = 1$. This is not in general the case with LT and Raptor codes presented in Section 3.3.

3.1.1 Encoding

Several different ways to define Reed-Solomon codes exist. The original definition in Reed and Solomon's work [39] uses evaluation of polynomials over finite fields as the name of the work suggests. Codewords in the Reed-Solomon codes can be produced by constructing a polynomial of data,

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{n-1} x^{n-1}, \tag{3.1}$$

where m_i denote the source symbols. Instead of binary symbols $\{0, 1\}$, a finite field algebra and symbols are used for coefficients m_i of terms of polynomial $c(x)$, see Appendix A. When (3.1) is evaluated over nonzero elements in $GF(2^m)$, codeword c is obtained,

$$c = \begin{pmatrix} m(1) & m(\alpha) & m\left(\alpha^2\right) & \ldots m\left(\alpha^{2^m-2}\right) \end{pmatrix}. \tag{3.2}$$

Other definitions include defining the Reed-Solomon code as a non-binary extension of BCH codes [33, 47]. To construct a Reed-Solomon capable of correcting up to t errors this way we need a generator polynomial which takes the form:

$$g(x) = \prod_{i=b}^{b+2t-1} \left(x - a^i\right), \tag{3.3}$$

where b is integer, usually 0 or 1. As Reed-Solomon codes are cyclic codes, all codeword polynomials can be obtained by multiplying some codeword polynomial $c(x)$ by the generator polynomial $g(x)$. This way using codeword polynomial $\hat{c}(x)$ another polynomial $\tilde{c}(x)$ is generated $\tilde{c}(x) = \hat{c}(x)g(x)$. Now using roots α^i of generator polynomial (3.3) it follows,

$$c(x) \text{ is a codeword polynomial} \iff c(\alpha^i) = 0, b \leq i \leq b + 2t - 1. \tag{3.4}$$

The following matrix can be constructed using (3.4):

$$\begin{pmatrix} c_0 & c_1 & \ldots & c_{n-1} \end{pmatrix} \begin{pmatrix} 1 & \alpha^b & \left(\alpha^b\right)^2 & \ldots & \left(\alpha^b\right)^{n-1} \\ 1 & \alpha^{b+1} & \left(\alpha^{b+1}\right)^2 & \ldots & \left(a^{b+1}\right)^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & \alpha^{b+2t-1} & \left(\alpha^{b+2t-1}\right)^2 & \ldots & \left(\alpha^{b+2t-1}\right)^{n-1} \end{pmatrix} = \mathbf{0}. \tag{3.5}$$

The matrix in (3.5) is the parity check matrix $\mathbf{H}$. Matrices of this form are called Vandermonde matrices [33].

Example 3.1. *Let a Galois field $GF(8)$ be generated by primitive element α so that its elements are as presented in Table A.2. To construct a $(7, 3)$ Reed-Solomon code which can correct up to $t = 2$ errors, we can use the following generator polynomial*

$$\begin{aligned} g(x) &= (x - \alpha)\left(x - \alpha^2\right)\left(x - \alpha^3\right)\left(x - \alpha^4\right) \\ &= x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3. \end{aligned} \tag{3.6}$$

Parity check matrix $\mathbf{H}$ *of this code can be constructed by result* (3.2) *and is as follows:*

$$\mathbf{H} = \begin{pmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \\ 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 \\ 1 & \alpha^4 & \alpha & \alpha^5 & \alpha^2 & \alpha^6 & \alpha^3 \end{pmatrix},$$

where Galois field arithmetic has been used to calculate the elements.

3.1.2 Decoding

Reed and Solomon in [39] presented just one approach for a decoding algorithm. Using polynomial encoding scheme as in (3.2), if the transmission succeeds without any errors, it is easy to solve the original message by solving any n of 2^m equations present in c. In case of errors in the received pattern, majority vote method is used.

However, Reed and Solomon's decoding algorithm is inefficient for large codes and large number of errors and other decoding algorithms have been constructed for decoding Reed-Solomon (and BCH) codes [33, 47]

Reed-Solomon codes as erasure codes

A good tutorial on how to use Reed-Solomon erasure codes in RAID-like systems are provided in [35] and [37]. The same method can be used also in other kinds of applications where erasure channel modeling is used. Here data is divided into n blocks, which are encoded using Vandermonde matrices. The resulting $n + m$ blocks are then distributed, and the receiver can recover the original data by collecting any n of the encoded blocks. Decoding the Reed-Solomon erasure code requires inversion of the generator matrix G, which can be derived from the parity check matrix H. Operations are done using Galois field arithmetic.

An excellent property of the Reed-Solomon codes, especially suitable for erasure correction, is their capability to retrieve original information consisting of n blocks of data by using any n of the $n + m$ coded blocks[1]; no overhead blocks are needed in contrast to the digital fountain codes to be presented later in Section 3.3. In this sense the Reed-Solomon codes are optimal.

3.1.3 Efficiency and performance of Reed-Solomon codes

Reed-Solomon codes have been studied exhaustively. Some of the more recent efficiency and performance studies with comparisons to other codes, mainly LDPC codes, include [41, 11, 9].

Although Reed-Solomon codes have some excellent properties, they are not very efficient for transferring large chunks of data using erasure correction.

In particular, if the number of message blocks is n and number of generated check blocks is m, then encoding takes $O(mn)$ operations and decoding requires a matrix

[1] This is the minimum distance separable (MDS) property

inversion, which is $O(n^3)$ operation. As m and n grow, this method becomes soon computationally too expensive. Especially when using software to perform the encoding and decoding, computation is demanding due to the arithmetic used which requires extra computing effort as Galois field arithmetic is not directly supported by typical hardware. Instead look-up tables have to be used for multiplication and addition and this takes extra steps and time in encoding and decoding algorithms. Reed-Solomon codes can be efficiently encoded and decoded using combinatorial logic in digital circuits when the size of the used Galois field is small, e.g., $q \leq 2^{16}$.

Recent result of efficient software FEC implementation of Reed-Solomon erasure coding [19] has shown encoding and decoding speeds of 200 Mbps (i.e. 25 MB per second) with a PC with Pentium IV 2.8 Ghz processor, which is a common processor for desktop PC nowadays. This might be sufficient for some applications, but better methods do exist as we will soon see.

3.1.4 Specific applications for Reed-Solomon codes

Reed-Solomon codes are widely used in many different technologies. Storage systems and portable media use Reed-Solomon coding to correct burst errors during data retrieval or playback. Applications in telecommunications include wireless technologies, digital subscriber lines and satellite communications. Also NASA has used Reed-Solomon based codes in their space missions.

3.2 Low-Density Parity-Check codes

Low-Density Parity-Check (LDPC), also called Gallager codes after their inventor, were presented in 1960 [14, 15]. These codes were largely forgotten for over forty years, largely due to the fact that computing power has been expensive and inadequate during the past decades for efficient use of LDPC codes and other codes have been thought to be better alternatives. Nowadays LDPC codes can be regarded as an viable alternative to Reed-Solomon erasure codes in different applications [38]. Since the "rediscovery" of LDPC codes in the 1990s, theoretical analysis of different LDPC coding methods has become popular but the practical side could get more attention.

3.2.1 Encoding of LDPC code

LDPC encoding procedure can be depicted using bipartite graphs, which determine how the parity symbols are generated from the original message. The n nodes representing message bits are called message nodes and $k - n$ nodes representing parity symbols are called check nodes. An example of a LDPC code defined using a bipartite matrix is presented in Figure 3.1. Calculation of check nodes can be read directly from the graph: Check node $c2 = m1 \oplus m2 \oplus m3 \oplus m4$, where $\oplus$ denotes the exclusive-or operation, which operates the same way as addition in $GF(2)$, see Appendix A. Both message nodes and check nodes are sent to recipient, the sent symbols are called packets in this context.

A regular LDPC code has the same degree d_m in all message nodes and similarly the same degree d_c in all check nodes. The graph in Figure 3.1 is regular in both parts, where

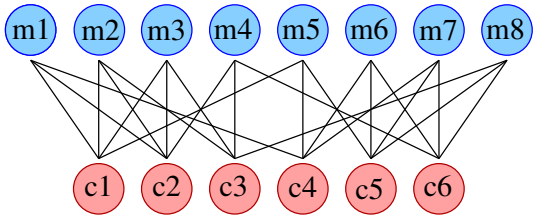

Figure 3.1: Example of a regular $(8, 6)$ LDPC code

$d_m = 3$ and $d_c = 4$. The LDPC codes can also be regular in only the message or check node part. The best LDPC codes today are however based on irregular bipartite graphs, see Section 3.3.

3.2.2 Decoding LDPC erasure code

If we forget the actual error correcting details on different channel types, and focus on erasure channels, decoding of LDPC codes can be done iteratively on packet by packet basis. Every time a packet is received, a check is made if some other node can be decoded. Using Figure 3.1 again as an example, if received set of packets contain nodes $c2, c3$ and $c6$, then message node $m4$ can be recovered by calculating $m4 = c2 \oplus c3 \oplus c6$. This method is equivalent to solving a linear system of equations with n variables. It should be noted that the required number of symbols is not known beforehand and the process continues until all of the message nodes have been recovered. This implies that, in contrary to the Reed-Solomon erasure codes, LDPC erasure codes require more than n of the original packets, resulting in overhead factor $f > 1$.

In the general case, decoding of an LDPC code is an NP-complete problem. Rather efficient approximation algorithms have nonetheless been developed, resulting in more efficient decoding than with the Reed-Solomon codes, particularly with large sizes of sent data. The most popular decoding algorithm is called sum-product or belief propagation algorithm [30].

3.2.3 Challenges in LDPC codes

Traditionally, theoretical results on the LDPC codes give asymptotical results for cases when the length of the message n tends to infinity. In this case it has been shown that overhead factor f tends to value one. Not until recently has work been done to construct optimal or near-optimal LDPC codes for small n, where the size of the message is $n < 1000$ [36]. The overhead factors f achieved with LDPC codes in region $1 \leq n \leq 100$, which is also later the focus of this work, are at maximum 1.1 with the best codes, corresponding to 10% overhead in packets.

3.3 Digital fountain codes

This section presents the work done by Michael Luby et al. They have made improvements based on the LDPC codes and presented very good codes under their digital fountain content distribution system. They have also founded a company Digital Fountain Inc. [2], whose business is to develop and license technology based on their efficient FEC erasure codes. Tornado, LT and Raptor codes are presented next, LT codes get the deepest treatment as they are the main topic of this work.

3.3.1 Background

In [30] a few methods are presented to make the LDPC codes to work more efficiently. One method is to use Galois fields or similar constructs to clump bits together. Another method to improve the performance is to make the graph irregular. This is discussed in [28] and also further demonstrated that irregular graphs outperform regular graphs in LDPC coding. Irregularity of the graphs is the vital reason for digital fountain codes to be so efficient and successful in erasure correction.

This idea was taken further by Luby et al. and new class of codes, called Tornado codes, was developed in 1997 [29, 9]. These were the first kind of codes to efficiently approximate a *digital fountain*. What Luby et al. call a digital fountain is an idealized model of content distribution: A source generates potentially an infinite amount of encoded packets and sends those into a network. The recipients of data need to collect only a certain amount of these packets to decode the original data. The term digital fountain comes from an analogy to a fountain: Server in this case is the fountain, who sprays packets corresponding to water drops, and recipients are analogous to buckets which is used to collect water. When the bucket is full, the process is finished and it does not matter which specific water drops were collected. Similar situation exists with the digital fountain concept: packets are received and it does not matter which specific packets are received. In ideal situation, if the original data consists of n encoded packets, only n packets need to be collected by the recipients. This can be achieved using codes with the MDS property, e.g., the Reed-Solomon codes presented earlier in Section 3.1. However, as discussed, computational complexity of the Reed-Solomon codes makes them impractical for use in case of large amount of data and block length. Thus Luby et al. have developed other kinds of codes for approximating the digital fountain.

3.3.2 Tornado codes

First class of codes published under the digital fountain concept, Tornado codes, work much more efficiently than Reed-Solomon codes in erasure correcting. In [9] the performance of Tornado codes is directly compared to Reed-Solomon codes. The presented results show that the Tornado codes are a much better alternative to approximating the digital fountain than the Reed-Solomon codes.

Figure 3.2 depicts the encoding strategy used in Tornado codes. Exclusive-or operation is used to generate the redundant symbols. Tornado codes are a specific class of LDPC codes and multiple bipartite graphs define the exact composition of these sym-

bols. Rate of the used Tornado code has to be fixed in advance, similarly as with Reed-Solomon codes. The composition of the bipartite graphs has to be be well though out in order to enable efficient encoding and decoding and to provide erasure correcting cababilities. A detailed discussion of good bipartite graphs for this purpose is given in [29].

If the number of the blocks the message is divided into is n, the recipient needs to collect a little more than n of the encoded packets in order to decode the original message (i.e., fn packets needs to be collected). This erasure correcting property enables the Tornado codes to approximate the digital fountain. The trade off compared to Reed-Solomon codes is this number of extra packets needed for decoding, but a good code design results in much better overall performance. The overhead factor f of Tornado codes can be tuned to around $f \approx 1.05$ for large n and k, an example is given in [29]. The encoding and decoding times of Tornado codes are proportional to $k \log 1/(f-1)M$, where M is the size of the original message.

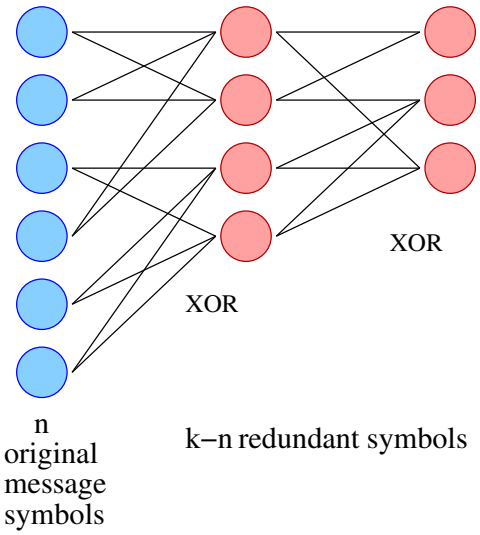

Figure 3.2: Idea of the Tornado codes. The $k - n$ redundant symbols are generated by exclusive-or operation in the way the bipartite graphs define. In order to decode the original message, the recipient has to collect a little more than n packets.

Although the Tornado codes are better in approximating the digital fountain than the Reed-Solomon codes, they are not ideal. The rate has to be fixed beforehand, and with too large a rate, it turns out that the recipient receives duplicate packets, which are useless and deteriorate the channel efficiency. Conversely, if the rate is small, then memory and encoding requirements make the Tornado codes perform poorly. Luckily, better codes for the digital fountain scheme exist, as we will see next.

3.3.3 LT codes

LT codes were published by Michael Luby in a landmark paper in 2002 [27]. These codes are rateless, meaning that the rate does not need to be fixed beforehand, and encoding symbols can be generated on the fly. LT codes are also first class of codes which are a

full realization of the digital fountain concept presented in [9].

Encoding of LT code

The encoding process is surprisingly simple. Following the tradition of LDPC codes presented earlier, also LT codes can be defined using a bipartite graph. This graph is irregular in LT codes, and a degree distribution is used to determine the degrees of encoding symbols.

Definition 3.2 (Degree distribution, generator polynomial)**.** *The **degree distribution** $\rho(d)$ of LT code is a probability distribution, where $\rho(i)$ is the probability of generating an output symbol consisting of i input symbols. Degree distribution can also be presented as a **generator polynomial** $\Omega(x) = \sum_{i=1}^{k} \Omega_i x^i$, where Ω_i is the probability for choosing value i.*

Degree distribution is sampled to obtain value d, which is used as the degree for output symbol $c(i)$ in the encoding graph. Output symbol is generated by choosing d input symbols $m(i)$ uniformly at random and calculating sum of these symbols in $GF(2)$ arithmetic. This is illustrated in Figure 3.3. Also listing Algorithm 3.1 shows the general framework of an encoding algorithm for the LT code. Stopping condition for the encoding algorithm could be specified by the number of output symbols agreed beforehand or when the recipient has enough symbols to decode the original message.

It should be noted that it does not matter what the symbol length is. One input symbol could be just one bit or a vector of bits, the encoding and decoding processes are the same regardless and the XOR operation is done bitwise to the whole vector.

Algorithm 3.1 A general LT encoding algorithm

1: **procedure** LTENCODE
2: **repeat**
3: choose a degree d from degree distribution $\rho(d)$.
4: choose uniformly at random d input nodes $m(i_1), \ldots, m(i_d)$.
5: $c(i) \leftarrow m(i_1) \oplus m(i_2) \oplus \cdots \oplus m(i_d)$.
6: **until** enough output symbols are sent
7: **end procedure**

Decoding of LT codes

Decoding is done similarly as decoding of the LDPC erasure codes. The decoding procedure needs the information of degree values of each encoding symbol it receives and the information of which source symbols are added together in a output symbol.

This information needs to be included somehow in the encoding procedure, further discussion of this topic follows in Chapter 5.

Decoding is started by receiving a degree-one output symbol. This symbol clearly has to be the same as the input symbol which it copied in the encoding process. This way, we have one input symbol uncovered, i.e., its value is known. Next we add this value (using

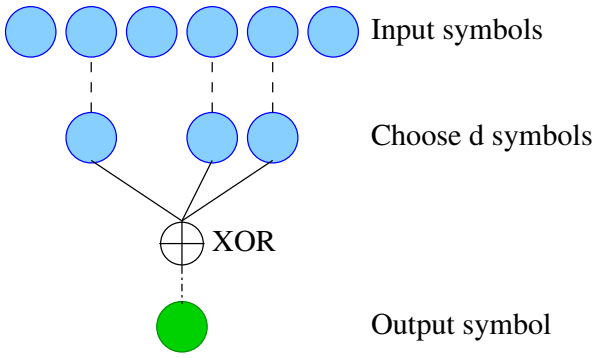

Figure 3.3: LT encoding. Value d is sampled from a degree distribution, output symbol is generated by successively using XOR to d selected input symbols

exclusive-or) to all neighbors of this uncovered symbol and remove the edges in the defining graph between the uncovered input symbol and its neighbors thus decreasing degrees in all of the neighboring output symbols. This way we have possibly more degree-one output symbols and the process may continue. An example illustration of the decoding process is presented in Figure 3.3.3. A framework of an decoding algorithm is sketched in listing Algorithm 3.2.

It should be noted that decoding LT code in this way is suboptimal in the sense that all of the information of the received packets is not used. For example, if source message consists of $n = 3$ blocks, the recipient could decode the original message if he had three different packets which each consists of two symbols. This, however, would make the decoding algorithm computationally too intense, as this method equates to solving a linear system of equations, an operation which is in general case too inefficient for this problem.

Algorithm 3.2 A general LT decoding algorithm

```
1:  procedure LTDECODE
2:      repeat
3:          repeat
4:              receive a packet
5:          until degree one check node c_n is available
6:          m_k ← c_n                    ▷ message node m_k has to be same as c_n
7:          calculate c_i = m_k ⊕ c_i for all c_i connected to m_k
8:          remove all edges between m_k and check nodes
9:      until original message is fully recovered
10: end procedure
```

Degree distributions

As stated earlier, the degree distribution plays an extremely important role in the LT coding process. Without a proper distribution, the whole concept of LT codes would be

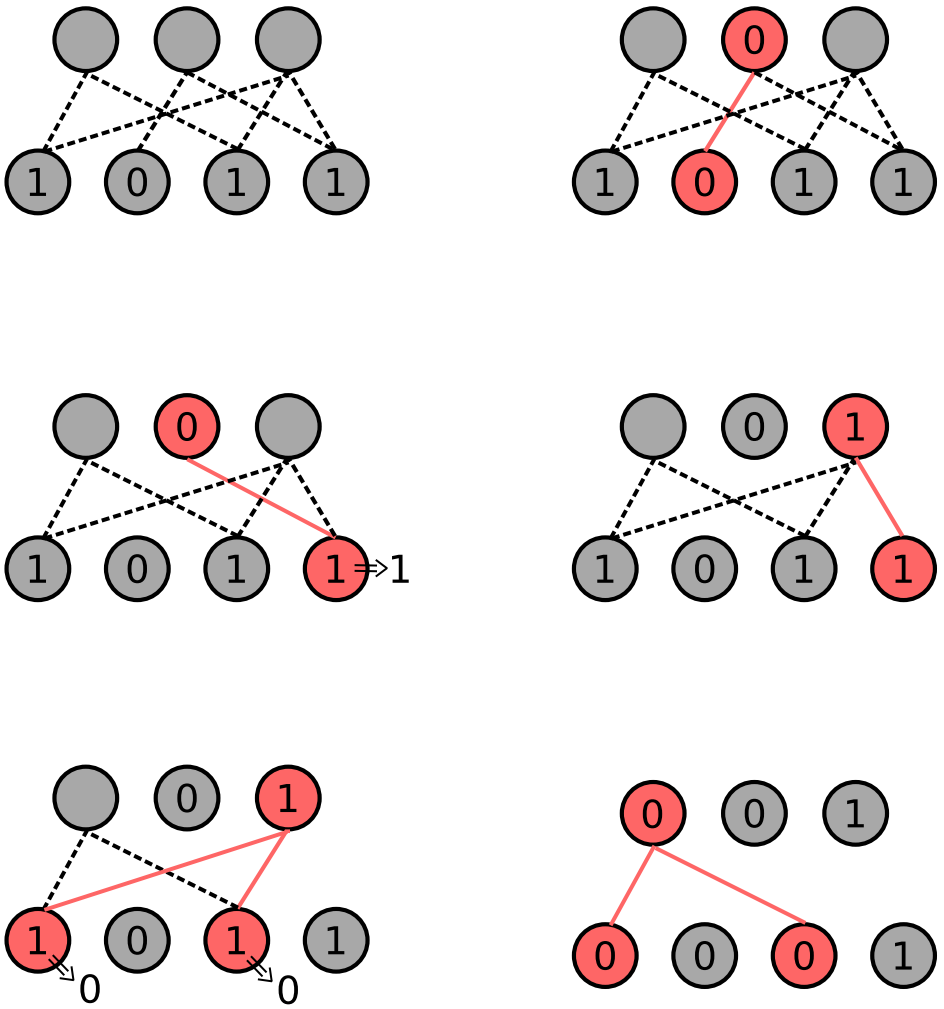

Figure 3.4: Illustration of decoding LT code. From top left to bottom right: The encoding procedure defines a bipartite graph, where each output symbol has one or many input symbols as neighbors. First we look for degree one (one neighbor) output symbol; we have one, so we know that the middle input symbol is 0. Now we can remove the edge between lower and upper parts. Next we XOR 0 with every connected output symbol, we have only one connection and XOR for 0 with 1 is 1. After this we remove the edge between operated 0 and 1 nodes. Now we look again for degree one symbols. There exists one such node, so again we remove the edge, and XOR the corresponding input symbol with all connected output symbols. Now the two connected output symbols change to 0. Next we remove the edges and in final step we have two degree one nodes which uncover the last unknown input symbol.

rather useless. The remarkable result proved by Luby in [27] is that efficient distributions do exist for LT codes.

As the degree distribution is the only factor which defines the efficiency of LT coding, the following two general principles can be stated about the design of a good degree distribution:

1. The number of output symbols which ensures the decoding of the original message should be as low as possible to keep the overhead factor f low.

2. The average degree of the output symbols should be as low as possible, so the number of steps needed in the decoding algorithm stays as low as possible.

The LT decoding process needs degree one symbols to keep the decoding going on. The *ripple* in LT process is the amount of input symbols which have been uncovered but not yet processed, i.e., number of input symbols in state of line 6 in Algorithm 3.1. The symbols in the ripple are then processed one by one as the rest of the algorithm states, possibly growing or decreasing the ripple as new input symbols are covered by the process.

Optimally the ripple is one in each step, thus one symbol can be used to decode one input symbol and further remove edges from the encoding graph. If number of available degree one symbols is larger than one, the received symbols are redundant thus increasing the inefficiency through a larger overhead factor f. On the other hand ripple should not go to zero at any point of the decoding process, otherwise the decoding halts and is unsuccessful. Consequently, the size of the ripple should be kept above one to avoid the complete disappearance of the ripple.

Let us denote $n_i^{(t)}$ the number of output nodes of degree i at time t. Time instant $t = 0$ corresponds to the start of the decoding algorithm, when the first degree one packet is available but none of the original blocks is yet decoded. Thus in the beginning, nodes of degree i have $i \cdot n_i^{(0)}$ edges in total connecting to the input nodes. This means that on average, one input node has $i \cdot n_i^{(0)}/N$ neighbors of degree i. For clarity we denote the number of input nodes here by capital N instead of the lowercase n used elsewhere in this work. For example, in Figure 3.3.3 at the first phase in top left, the average number of output symbols of degree two as neighbors of input symbols is $(2 \cdot 3)/3 = 2$. Now, when an output symbol of degree one is processed and the edges removed accordingly, the number of degree i packets whose degree decrease by one is in expectation the average $i \cdot n_i^{(0)}/N$. If already t (i.e., at time t) input symbols have been decoded, the edges have only $n - t$ input nodes to connect to, so the average is $i \cdot n_i^{(t)}/(N - t)$.

The optimal condition in terms of notation presented above, is

$$n_1^{(t)} = 1 \quad \forall\, t \in \{0, \ldots, N-1\}. \tag{3.7}$$

The optimal distribution can be now constructed by considering which conditions lead to the optimal situation in each step as presented above, i.e., what are the values of $n_i^{(0)}$ for different degrees i. Naturally from (3.7) we have $n_1^{(0)} = 1$. Value of $n_2^{(0)}$ has to be such that at time $t = 1$ the amount of degree one nodes is again one, this means that one

of the degree two nodes at time $t = 0$ decreases its degree, so we have the equation

$$\frac{2 \cdot n_2^{(0)}}{N} = 1 \quad \Longleftrightarrow \quad n_2^{(0)} = \frac{N}{2}, \tag{3.8}$$

and more generally at time t

$$\frac{2 \cdot n_2^{(t)}}{N-t} = 1 \quad \Longleftrightarrow \quad n_2^{(t)} = \frac{N-t}{2}. \tag{3.9}$$

By continuing this reasoning recursively one obtains the rest of the values $n_i^{(0)}$. The number of degree two nodes at time t is the same as number of degree two nodes at time $t-1$ minus the nodes which optimally decrease their degree plus the nodes which previously were of degree three, i.e.,

$$\begin{aligned} n_2^{(t+1)} &= n_2^{(t)} - \frac{2 \cdot n_2^{(t)}}{N-t} + \frac{3 \cdot n_3^{(t)}}{N-t} \\ \frac{N-t-1}{2} &= \frac{N-t}{2} - 1 + \frac{3 \cdot n_3^{(t)}}{N-t} \qquad \Big| -\frac{N-t}{2} \\ -\frac{1}{2} &= -1 + \frac{3 \cdot n_3^{(t)}}{N-t} \\ \Rightarrow n_3^{(t)} &= \frac{N-t}{2 \cdot 3} \end{aligned} \tag{3.10}$$

Hence, the value we are looking for, $n_3^{(0)} = N/(2 \cdot 3)$. The general state equation for degree i nodes at time t is

$$n_i^{(t+1)} = n_i^{(t)} - \frac{i}{N-t} n_i^{(t)} + \frac{i+1}{N-t} n_{i+1}^{(t)}, \tag{3.11}$$

which can be solved for $n_{i+1}^{(t)}$:

$$n_{i+1}^{(t)} = \frac{N-t}{i+1} \left(n_i^{(t+1)} - n_i^{(t)} \right) + \frac{i}{i+1} n_i^{(t)}. \tag{3.12}$$

Equation (3.12) gives recursively the rest of the values. The next theorem, however, provides a simpler form.

Theorem 3.3. *The number of degree i nodes at time t leading to optimal degree distribution in expectation, i.e., provides ripple of one in expectation is*

$$n_i^{(t)} = \frac{N-t}{i(i-1)}. \tag{3.13}$$

Proof. Proof follows from the discussion above and from Equation (3.12) with induction. Details of the proof by induction are given in Appendix B. □

Now, the actual values we are looking for are $n_i^{(0)}$. By Theorem 3.3 these are

$$n_i^{(0)} = \frac{N}{i(i-1)} \text{ for } i \in \{2, \ldots, N-1\}, \tag{3.14}$$

and $n_1^{(0)} = 1$. In order to get the needed probability distribution, we divide these optimal numbers of different degree nodes at time 0 by the total number of blocks in the message, i.e., we normalize the values $n_i^{(0)}$ in order to get a probability distribution. We arrive at:

Definition 3.4 (Ideal Soliton distribution)**.** *The **Ideal Soliton distribution** $\rho(i)$ is:*

$$\rho(i) = \begin{cases} \frac{1}{n} & \text{when } i = 1 \\ \frac{1}{i(i-1)} & \text{for } i = 2, \ldots, n. \end{cases}$$

Beginning of this distribution is presented in Figure 3.5 for $n = 1000$. As the basis for constructing this distribution was ideal behavior in expectation, it is not surprising that in practice the Ideal Soliton distribution does not work well. The expected ripple size of one will vanish with variance, resulting in poor performance.

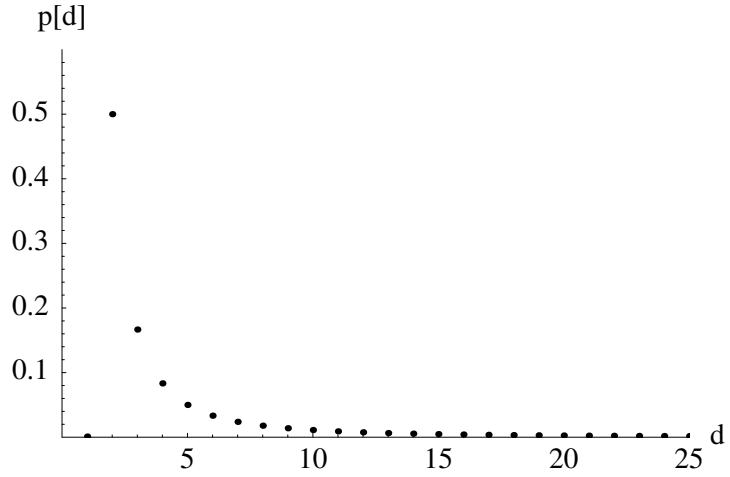

Figure 3.5: Start of the Ideal Soliton distribution for $n = 1000$

The main results in Luby's work [27] concern the Robust Soliton distribution, which is an advanced version of the Ideal Soliton distribution. The goal is to keep the ripple so large that it will not vanish at any point of the decoding process and also to minimize the expected ripple size so that the redundancy is kept low.

Definition 3.5 (Robust Soliton distribution)**.** *For the **Robust Soliton distribution**, first define the function:*

$$\tau(i) = \begin{cases} \frac{R}{in} & \text{for } i = 1, \ldots, \frac{n}{R} - 1 \\ \frac{R \log R/\delta}{n} & \text{for } i = \frac{n}{R} \\ 0 & \text{for } i = \frac{n}{R} + 1, \ldots, n, \end{cases}$$

where δ is the failure probability of decoding process after n' encoded symbols and $R = c \cdot \log(n/\delta)\sqrt{n}$ for some constant $c > 0$. The Robust Soliton distribution $\mu(i)$ is the normalized value of the sum $\rho(i) + \tau(i)$:

$$\mu(i) = \frac{\rho(i) + \tau(i)}{\sum_{i=1}^{n} \rho(i) + \tau(i)}.$$

An example of the Robust Soliton distribution with parameters $\delta = 0.95$ and $c = 0.2$ for $n = 1000$ is presented in Figure 3.6. In short, the addition of $\tau(i)$ in Definition 3.5 should ensure that:

1. The process starts with large enough ripple.
2. The ripple decrease of one every time a input symbol is uncovered is countered by increasing the ripple by one.
3. All input symbols are covered at the end of the process by placing spike $\tau(n/R)$ at some high degree.

The Robust Soliton distribution was used in [27] to proof that original message can be recovered from $n + O(\sqrt{n}\log^2(n/\delta))$ output symbols with probability $1 - \delta$. The encoding and decoding complexities are then $O(\log(k/\delta))$ in terms of arithmetic operations.

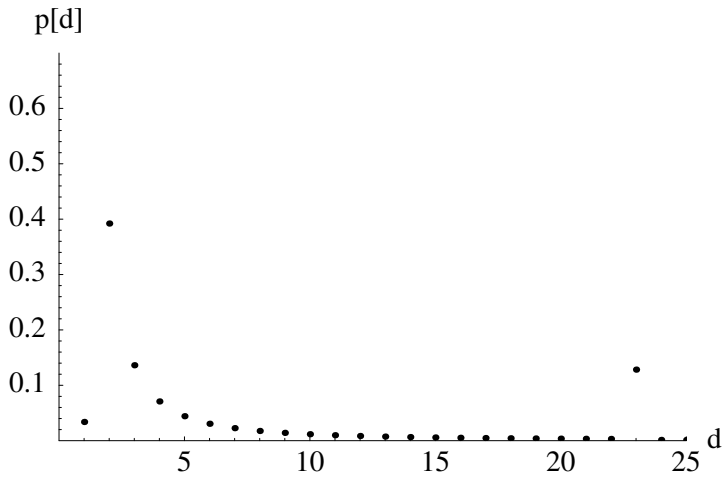

Figure 3.6: Start of the Robust Soliton distribution for $n = 1000$. Parameters are $\delta = 0.95, c = 0.2$. Note the spike at $n = 23$.

Linear Systems of Equations Approach to LT Codes

As stated above, the LT codes could also be described with the help of linear systems of equations. The encoded symbols used in LT codes are actually linear equations of n possible variables, as seen in for example the description of Algorithm 3.1. The degree

distribution gives a random value which is used to choose d blocks from the original message, which are then combined using XOR, which equals to modulo-2 addition.

This approach leads to very low overheads, which is also rather easy to calculate analytically. To decode the message by solving a linear system, i.e., by matrix inversion, we need to have exactly n linearly independent equations. In other words, if we want to decode the original message in exactly n steps, we need a $n \times n$ matrix of full rank. Let us first calculate the probability of generating a random $n \times n$ full rank matrix. We consider the generation on row-by-row basis, where each block is chosen to be included with probability $1/2$. This means that there are 2^n possible choices for one row. The all zeros vector is not accepted, as it is linearly dependent with all other vectors, and corresponds to a message with no information. So, at the first step we have $2^n - 1$ possibilities to choose from, i.e., the probability to succeed (generate one linearly independent vector) is $(2^n - 1)/2^n$. To generate a new linearly independent vector, we exclude the all zero vector and the one generated before, i.e., the probability is now $(2^n - 2)/2^n$. Third vector has to be different from the zero vector and all vectors generated before. Also the linear combination of two previously generated vectors is not accepted now. The number of linearly dependent vectors that can be generated from i vectors is:

$$\binom{i}{0} + \binom{i}{1} + \binom{i}{2} + \cdots + \binom{i}{i} = 2^i, \tag{3.15}$$

where each binomial term $\binom{i}{k}$ represents the number of linearly dependent vectors that can be formed by choosing any k equations from the i possible ones. The second term $\binom{i}{1}$ corresponds to any previously generated equation and the special case of zero vector is handled by the first term, $\binom{i}{0} = 1$.

This means that after $i-1$ linearly independent equations, the probability of generating the ith linearly independent random equation is

$$\frac{2^n - 2^{i-1}}{2^n} = 1 - 2^{i-1-n}, \tag{3.16}$$

where the subtracted term in the numerator corresponds to the number of linearly dependent vectors that can be generated from $i-1$ previous linearly independent vectors. This leads us to the probability for successionally generating n linearly independent vectors:

$$\mathcal{P}_n = \prod_{i=1}^{n} \left(1 - 2^{i-1-n}\right). \tag{3.17}$$

The probabilities for $n = 1 \ldots 20$ are plotted in Figure 3.7. As n tends to infinity, the probability converges approximately to 0.2888, hence the probability of generating a full rank $n \times n$ random matrix is about 29%.

What if the generation does not succeed in n steps? If the generated n equations include $n-1$ linearly independent ones, then the we can calculate the probability for the new equation to be linearly independent of the rest with (3.16):

$$1 - 2^{n-1-n} = 1 - 2^{-1} = \frac{1}{2} \tag{3.18}$$

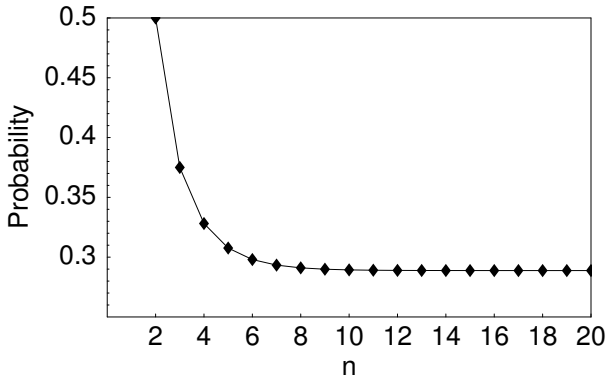

Figure 3.7: Probability that n random generated binary n-vectors with probability $p = 1/2$ are all linearly independent. The probability converges to 0.2888.

Thus, the probability of not succeeding in n steps, but requiring $n + n'$ steps decreases roughly like $2^{-n'}$.

To calculate the expected number of random equations needed for full rank, we note that the probability that we need i additional packets to generate the next linearly independent equation is geometrically distributed:

$$\mathcal{Q}_i = (1-p)p^{i-1}, \qquad i = 1, 2, \ldots \tag{3.19}$$

where p is the probability that we fail to generate next linearly independent equation. The expectation of (3.19) is:

$$\begin{aligned}\sum_{i=1}^{\infty} i\mathcal{Q}_i &= (1-p)\sum_{i=1}^{\infty} ip^{i-1} = (1-p)\frac{1}{p}\sum_{i=1}^{\infty} ip^i = \frac{1-p}{p}\sum_{i=1}^{\infty} p\frac{d}{dp}p^i \\ &= (1-p)\frac{d}{dp}\sum_{i=1}^{\infty} p^i = (1-p)\frac{d}{dp}\frac{p}{1-p} \\ &= (1-p)\frac{(1-p)-p(-1)}{(1-p)^2} = \frac{1}{1-p}.\end{aligned} \tag{3.20}$$

We therefore define the number of extra equations needed for a full rank random matrix, when we already have k linearly independent equations, to be

$$r_k = \frac{1}{1-p_k}, \tag{3.21}$$

where the probability of failure in the next step $p_k = 2^{k-1}/2^n = 2^{k-1-n}$ (compare to (3.16)). Now the following sum gives the total expected number of random equations needed for a random full rank $n \times n$ matrix:

$$\sum_{k=1}^{n} r_k = \sum_{k=1}^{n} \frac{1}{1-p_k} = \sum_{k=1}^{n} \frac{1}{1-2^{k-1-n}}. \tag{3.22}$$

In Figure 3.8 we have plotted the expected number of overhead equations, i.e., $\sum r_k - k$,

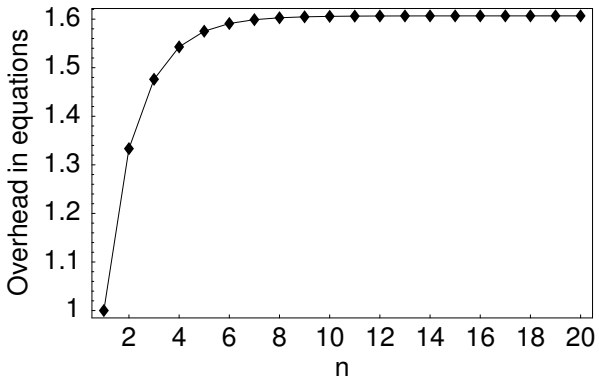

Figure 3.8: The expected number of overhead equations to generate a random full rank $n \times n$ matrix.

for $k = 1, \ldots 20$. We see that the overhead seems to converge to 1.6 equations for large n, for example when $n = 1000$ the overhead is 1.61 equations. Thus the overhead in general with this approach would be under 2 packets regardless of the value of n. However, decoding of a code generated in this way requires the solving of a full system of linear equations, which generally is an inefficient operation.

Note that this same approach can also be taken with LDPC codes, where of course the generation of the equations is different, depending on the parity check matrix $\mathbf{H}$.

3.3.4 Raptor codes

Raptor codes were developed by Amin Shokrollahi while he was working at Digital Fountain. The discussion provided here is adapted from a preprint paper [44]. Raptor codes are an essential part of Digital Fountains current content delivery system and were recently chosen as part of 3GPP's[2] Multimedia Broadcast/Multicast Service (MBMS) for 3rd generation cellular networks.

Encoding and decoding complexities of Raptor codes are linear, and thus efficiency is better than with LT codes, where the decoding cost in average per symbol is $O(log(n))$, resulting in at least $O(n \log(n))$ total cost [44]. This is achieved by relaxing the requirement that all input symbols needs to be recovered. Raptor codes are defined as an extension to LT codes: original message $m(x)$ is passed trough a preliminary coding process called *pre-coding*, producing intermediate nodes and these intermediate nodes are passed to LT encoder as input nodes. The code used for pre-coding is denoted $\mathcal{C}$. This process is visualized in Figure 3.9. When defined this way, we see that first class of Raptor codes are LT codes without pre-coding. Pre-coding can also be done in several steps, e.g., first use a basic Hamming code to encode the message and encode output symbols of the Hamming code using an LDPC erasure code to produce the intermediate symbols for LT encoder. Also codes without LT coding can be regarded as a subclass of

[2] 3GPP (3rd Generation Partnership Project) is a collaboration agreement between a number of telecommunications standards bodies (Organizational Partners). Also telecommunications industry has its own Market Representation Partners. 3GPP's scope is to produce specifications and reports for 3G mobile system based on GSM and to maintain and develop GSM specifications and reports. See http://www.3gpp.org.

Raptor codes, called *pre-coding-only* or PCO Raptor codes.

By using some erasure correcting code as the precode, the requirement to recover all input symbols of the LT code is lifted: only a constant fraction of LT encoded symbols needs to be recovered, the original message can then recovered by the erasure correcting property of the code used for pre-coding.

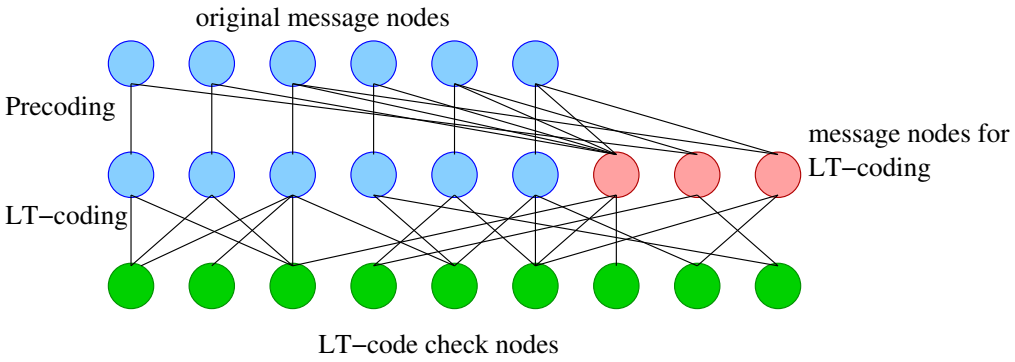

Figure 3.9: Example of Raptor coding. Original message is encoded using some traditional erasure correcting code, e.g. LDPC code. This process produces intermediate nodes, shown in the middle row, and these are encoded using LT code to produce the output symbols.

The decoding of Raptor code is done by first using the LT decoding process to produce the intermediate code and after that input nodes are recovered by applying decoding algorithm for code $\mathcal{C}$.

Chapter 4

Applications

This chapter presents some of the possible applications for the state-of-the-art coding schemes presented in this work. Traditionally erasure and FEC coding are employed in situations where channel errors are abundant, the applications proposed in this chapter could work efficiently even if the error rate is reasonable. All of the presented applications are some form of content distribution or data storage systems and usually deal with large data files.

4.1 Reliable Multicast

Multicast means information delivery at the same time to many different receivers, in contrast to point-to-point delivery or unicast. In computer networking, different kind of multicast schemes have been developed using various techniques, at different levels or layers and for different purposes. In the first descriptions of the fountain coding principle, the particular application considered was multicasting large chunks of data to multiple recipients from one source (server) [8].

Often multicast refers to IP multicast, where data is sent to a multicast IP address, corresponding to a group of recipients (multicast group). IP multicast is by no means a simple process, multiple algorithms and protocols take part in a successful multicast transfer. The major problem with IP multicast, compared to other protocols enabling similar function, lies in the history of the TCP/IP protocol stack. In particular, the original usage of IP protocol did not take into account multicast scenarios and the current functionality is acquired through extensions. The IP protocol provides only best-effort network, and the state information which needs to be stored when using multicast does not fit into the best-effort principle. Composition and location of nodes participating in a multicast group has to be stored in the intermediate nodes, and the packets have to be routed to the specific locations of the receivers. This leads to high inefficiencies and complexity. Possible packet losses have to be dealt somehow, and retransmission based error correction leads to disaster when losses are high or number of nodes in multicast group is high. For example with TCP, packets are acknowledged by the recipient, and unacknowledged packets are retransmitted. In situations where network latencies or loss rates are high, the congestion and flow control mechanisms of TCP do not scale well.

This has naturally generated criticism against multicast in the past, and even today, large scale multicast to large amount of receivers is neither viable efficiency-wise nor even possible.

The fountain coding principle, however, fits nicely into multicast context. The concept of a server providing infinite supply of packets, which anyone can collect to decode the original message, lies at the heart of the fountain coding concept. Basically this concept lends itself to multicast. It enables the transfer of infomation to multiple recipients, the number of such recipients being not important at all as the technique scales exceptionally well. Possible lost packets are not a concern at all, and in extreme situations there is no need at all for a backward channel to exist, and the only remaining function of the originating server is to blindly push packets into the network. Of course, similar algorithms as for IP multicast have to be in place for the multicast to be used in a general network where different types of data transmission is enabled, that is, in networks where also unicast is employed.

The use of FEC erasure codes frees the server from the duty of retransmitting the missing data pieces individually to different recipients. Especially in the case of high packet loss situations, the erasure schemes have major advantages over currently used IP multicast techniques.

One specific application would be broadcasting data to in-car navigation systems. Either land- or satellite-based systems send location information to cars, where in-car systems help the driver by giving driving instructions. Sometimes the transmission does not succeed, and as the systems are passive, they do not have the functionality to send information backwards and ask for missing pieces. The traditional way of taking these losses into account is the *data carousel*: the different packets are periodically broadcast, while the recipients miss some packets first time, after a while the missing pieces are resent and possibly received. While this might be efficient enough for some channel error probabilities, FEC codes would make it possible to receive the size of the original data plus then some, perhaps leading to a better efficiency.

4.1.1 IETF and Reliable Multicast

The party developing the core Internet protocols and technologies is IETF. One of the many working groups working with multicast issues is reliable multicast transport (rmt) working group, whose mission is to standardize protocols for one-to-many transport of large amounts of data.

The working group is taking a new approach to protocol standardization. As the different possible applications taking advantage of multicast have different requirements, the different functions are divided into blocks. The blocks are coarse grained with abstract APIs[1] and can be adjusted to be used with specific application needs. The possibility to upgrade different blocks allows easy incorporation of new research results and findings. This block division is discussed in [46].

Also, two different instants of protocols are specified. These specification define the use of the blocks to create a minimal functionality protocols with APIs to be used be-

[1] Application program interface (API) is a set of routines and protocols which programmer can use to call and use the underlying software blocks and libraries

tween protocols and applications. The two instants are:

1. NORM: Nack Oriented Reliable Multicast protocol. Specified in RFC 3940 [6], NORM provides reliable multicast over IP multicast network by using selective acknowledgements to request repairs for missing data.

2. ALC: Asynchronous Layered Coding. Specified in RFC 3450 [23], ALC provides congestion controlled asynchronous transfer from one sender to unlimited number of recipients.

The latter of these is used to provide massively scalable transfer scenario over IP multicast. ALC should scale to millions of recipients for any file sizes, providing maximum fair transfer rates between the sender and a specific receiver. A full protocol implementation for file transfer application is discussed in RFC 3926 [34]. This RFC describes FLUTE, a unidirectional file transfer protocol which works over Internet. FLUTE employs ALC and, accordingly, inherits the great scalability properties.

Two RFCs of particular interest are [25] and [26]. These describe and provide specification of use of a FEC blocks in reliable multicast. It should be noted that many of the RFCs published by the reliable multicast transport working group are authored by the same people doing research in the areas of coding theory described in this work (in particular, Michael Luby is one of the authors in many of the aforementioned RFCs). Also an internet draft describing the usage of Raptor codes in a FEC block is provided by the reliable multicast transfer group [24].

The FEC building blocks can be included in the implementations of NORM and ALC protocol types. The block based design principle shows its strength here; as more knowledge and understanding of different methods is generated, the block can be upgraded without interfering with the function of the protocol utilizing the FEC erasure features.

4.2 Peer to Peer Networking

Peer to peer (P2P) networking is one of the prominent success stories of the 21st century. Instead of relying on fixed servers, peer to peer technologies enables the participants (usually end users) to share their resources with each other. Examples of resources that could be shared are bandwidth, storage capacity and computing power. Several real world implementations of different P2P systems exists, and the number is growing. In a way, different P2P implementations are actually providing the service that IP multicast, in theory, should provide.

Especially file transfer related P2P networking could benefit from FEC codes. In the file transfer oriented peer to peer systems, a peer simultaneously serves blocks of a particular file, and downloads missing blocks of the file from other peers. To make this work efficiently, the sending and receiving of different blocks should be as flawless as possible. FEC erasure codes could be utilized to code the original files into encoding blocks, and then these blocks could be distributed. This of course requires that all participants use the same kind of codes with the same parameters.

Especially types of codes similar in function to LT codes could be practical here. The supply of different encoding blocks is virtually infinite and the probability that exactly

the same composition is sent multiple times from a single seed is low. This ensures that the recipient does not receive the same information multiple times.

Another area where FEC erasure codes could help is the last block problem. If only one seed has the whole file and leaves the system, the rest of the participants are left stranded with no means to receive the complete file. If the seeds originally send erasure encoded blocks, information from all of the source file blocks would be incorporated into the system possibly sooner. This of course does not remove the last block problem completely. Problems arise, for example, if there is only one seed who has the complete file and that seed leaves the system before the sent data size equals to total length of the file itself, i.e., before enough independent blocks of the original file are sent. This said, the erasure codes could probably help in the general case, if the design of the system is done carefully utilizing the full potential of erasure correcting codes.

It should be noted that successful P2P protocols exist, arguably the most successful being BitTorrent[2], without using FEC in file transfer. Other kinds of mechanisms are in place to guarantee efficient operation of the file transfer.

For P2P systems where different mechanisms are available for the last block problem and efficient distribution of the blocks, it probably is not worthwhile to implement erasure coding. On the other hand, if a P2P system is designed from the beginning to take the full advantage of the good properties which erasure codes have, the result could be elegant and effective.

4.3 Distributed Storage

It could be argued that distributed storage is actually one form of P2P networking. Nevertheless, it is here discussed under its own section. Distributed storage, as used here, could refer to anything between RAID systems and sensor networks.

Replication is one way to utilize distributed systems. This means dividing the data into blocks and then distributing these blocks over the available storage nodes. An user, who wants to retrieve the original data would then gather all the different blocks from the nodes where the needed blocks reside.

When using erasure codes, the original file would be divided into n blocks again, these blocks would then go trough the coding. Finally the encoded packets are distributed over the storage nodes. The benefit from using erasure codes over replication shows when an user wants to acquire the original data; optimally any n different pieces from any of the nodes are sufficient to decode the original data. With, for example, Reed-Solomon codes exactly n pieces is enough, with LT codes little more than n is necessary. With replication, the user needs to collect all of the original pieces, i.e., it does matter what are the pieces the recipient collects. This might be just n blocks with good luck, usually probably a lot more.

OceanStore [4] is one example of a project, where Reed-Solomon erasure codes are used to provide one form of distributed storage. Further considerations for practical use of different LDPC codes in distributed storage are given for example in [38].

[2] An open source P2P file sharing protocol originally developed by Bram Cohen, http://www.bittorrent.org.

Chapter 5

Implementation Issues

FEC systems in general have existed for a long time and have a vast field of different uses. However, to employ FEC erasure codes in software and in end-to-end fashion is not, at least yet, a very popular application. This chapter presents some insight to the possible problems of employing the schemes presented in Chapter 3 on the Internet or in communications networks in general. Each issue is discussed in its own section with some practical solution suggestions and guidelines. The emphasis is on codes similar to LT codes, although the issues presented are valid for many different types of coding schemes and communication protocols in general.

5.1 Efficiency

The prevailing usage of different FEC methods is in the link layer, where errors are detected and corrected between two adjacent nodes. The coding methods used here usually work with small codelengths and have very fast integrated circuit implementations directly in the hardware.

The idea of software FEC, especially erasure codes, has perhaps always been an option considered for some applications, but efficient ways to implement it have not existed. Especially, while some codes have been developed, the computing power has been inadequate to attain satisfactory sustained speeds of encoding and decoding of these codes with software.

With the advent of more efficient computing methods and computers, the encoding and decoding speeds are not as big a factor as before. Still a great care has to be taken to implement the encoder and decoder in an efficient way.

The LT codes presented in Section 3.3 have encoding and decoding algorithms which are efficient to use. Raptor codes are an even more efficient extension of LT codes allowing linear time encoding and decoding [44].

5.2 Overhead

All of the coding schemes introduced in Chapter 3 have overhead factor f larger than one, except the Reed-Solomon codes, meaning that it is always necessary to actually

send more packets than the size of the message is, even in the case when the channel does not destroy any of the packets in transit. For good methods, the overhead is naturally as small as possible, that is, f is close to unity.

Optimal coding methods have been developed for infinite size codes. It has been shown, for example, that for LT and Raptor codes the overhead factor f converges to unity as the number of blocks n grows. Other coding methods exist for small number of blocks with reasonable overheads, for example in [36] the authors have explored LDPC codes for small numbers of data blocks n and additional amount of coding blocks needed. Later in this work, in Chapters 6 and 7, we present our own optimization method and results for small number of blocks n (small here refers to $n \leq 100$).

The fact that the overhead factor f is not one in actual implementations could be a problem if widely used in a network, as the amount of overall data transmitted is directly proportional to the value of f. Data increase of several percentages in already congested network link might be catastrophic. While the different networking technologies evolve and link speeds grow, at the same time the data sizes also tend to grow, e.g., the industry is looking forward to distribute multimedia (movies, games, etc.) via the Internet.

5.2.1 Simple Model for Optimal Block Sizes

When assessing the performance of different erasure correcting methods, it is vital to not only look at the overhead factor of different codes, then choose the code with the best factor, and blindly use it to deliver content. The size of the data has to be taken into account, and the division of the data into blocks needs to be considered. Here we will present a simple scheme of choosing a good block division based on the data size M, the overhead factor f and the length of the header used h.

Let us denote the generic total time of transportation of a set of data by T. The speed of the link used is irrelevant here and not taken into account, as T is directly proportional to the size of the data regardless of the link speed. If we have the data size M divided into n blocks, then one packet sent has the size $(M/n + h)$, where the length of the header is taken into account. Now, if we have the idealized case of the channel not losing any of the packets, then the time to transfer the whole data is the total number of packets times the time it takes to transfer one packet:

$$T \propto n \cdot f \left(\frac{M}{n} + h \right) = f \left(M + nh \right), \tag{5.1}$$

where $f \geq 1$. This shows the basic relations: in order to get lower transportation times, f needs to be near unity and large n tends to slow down the process. This is not the complete truth. The overhead factor f is not constant, but varies with the used n. Typically different coding schemes (e.g. LT codes) have f approaching unity with growing n so the case is not as simple as it first seems. Also the header length might increase with larger n values, as the additional information conveyed might increase with the data length. For example, with LT codes the block composition has to be included in some way in the packets so that the decoder can decode the packet. This is called synchronization and is briefly discussed in Section 5.3.

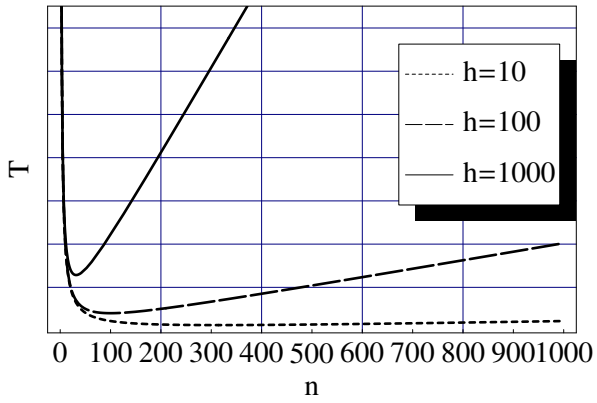

Figure 5.1: Time to complete file transfer in Example 5.1. Data size M is 10^6 bits with header length h. The minimum transfer time achieved clearly depends on the chosen number of blocks n and is different with every h. The best time here is achieved with header length $h = 10$, when $n = 317$.

Now, let us introduce the probability of channel losing packets p_f and also introduce the fact that f depends on the number of the blocks. We get the following result

$$T \propto n\frac{f(n)}{1-p_f}\left(\frac{M}{n}+h\right) = \frac{f(n)}{1-p_f}\left(M+nh\right). \tag{5.2}$$

This is a minimization problem of one variable and as such easy to solve for particular data and header lengths, when the dependency of the overhead factor of the data length is known.

Example 5.1 (Optimal block size)**.** *We take the idealized scenario of $p_f = 0$ and try to determine the optimal size of blocks, i.e., the number of blocks n used with some erasure code $\mathcal{C}$. This code has the property that $f(n) = 1 + 1/n$. Now Equation (5.2) gives:*

$$T \propto \left(\frac{1}{n}+1\right)\left(M+nh\right) = \left(\frac{1}{n}+1\right)M + \left(1+n\right)h.$$

This shows that when n grows, the header length h becomes more and more dominant with regards to the transfer time. In Figure 5.1 T is plotted with $M = 10^6$ bits with header length $h = 10, 100$ and 1000 bits. The respective minimums are achieved with $n = 317, 100$ and 32 packets (rounded up to complete packets). Optimal block sizes in this example would then be $3155, 10000$ and 31250 bits respectively.

This simple example shows that depending on the data size and protocol specification (i.e. header length), the minimum transfer times depend on the chosen block sizes.

5.3 Synchronization

Especially in the case of LT codes, there has to exist some kind of synchronization between the senders and receivers. This synchronization is needed for telling the compo-

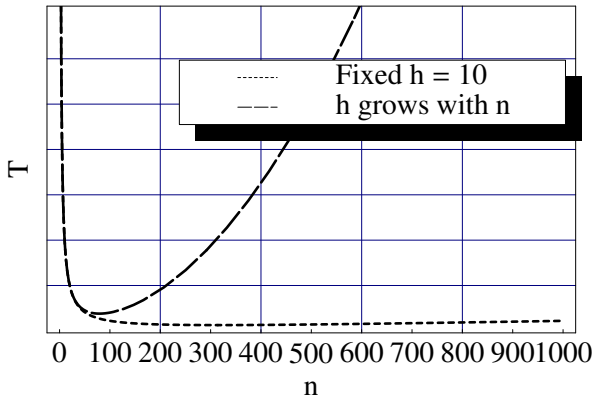

Figure 5.2: The case of $h = 10$ compared to $h = n$, the latter occurring when using bit mask synchronization. Clearly better results are achieved when the header has fixed length.

sition of a specific packet for the decoder. Referring to the decoding algorithm 3.2, the decoder needs to know which codeword nodes are connected to message nodes in order function properly.

The basic way of implementing synchronization would be directly conveying the information of the composition of a packet in its header. This could be implemented for example by a bit mask of length n, where each '1'-bit would tell the decoder that corresponding block is included in the packet and correspondingly '0'-bit would count for a packet not included. This method becomes inefficient when because as the number of blocks n grows and the header becomes longer. Smaller header lengths are naturally better as there is less data to be sent, this point was also illustrated in Example 5.1. The results of Example 5.1 are further compared to bit mask synchronization in Figure 5.2.

The bit mask implementation is nonetheless probably exaggerated. As great part of the sent packets should not include too many blocks of the original message, for example some identification numbers of the included blocks could be transferred instead. This of course brings the question of the length of the information when the numbers given are in some integer form, and this kind of implementation would be reasonable only if the total length of the header in bits would be less than the bit mask presented above.

This means that methods beyond the basic one presented above should be considered. One way is to use the same pseudo-random number generator (PRNG) at both the decoder and the encoder. The encoder uses PRNG to combine multiple blocks into one packet. First the number of blocks is drawn and afterwards the included blocks are selected uniformly at random. With a message of length n blocks, we then have one-to-one correspondence of packets generated with two PRNGs of the same type with common seed.

This method would need some kind of sequencing method of the packets: this sequence number could then be used as a seed for the decoders PRNG to derive the correct composition of the packet. The use of a PRNG is not necessary, any deterministic function could be used for this kind of operation to succeed.

The synchronization could also be implemented by examining the timing of receiving

the packets or the relative position of the received packet to other packets. If a deterministic function is used to generate the degree distributions, the position of received packets could hint the receiver which degree composition corresponds to received packet. There is however the problem of the transport paths of the packets differing: especially in IP-style best-effort networks the packets can take different routes between sender and receivers and arrive out of order. If the receiver can not determine the correct composition, the whole coding process could be ruined.

Possibly the easiest implementation would be to just include some key or sequence number to each packet as a header. Although this adds some overhead to the overall process, it is controlled and could be adjusted depending on the application and performance requirements.

5.4 Security

Security is one of the issues which nowadays should not be forgotten. When implementing any of the erasure coding schemes presented earlier, several security issues has to be taken into account, as with any communications protocol:

1. How can the receivers detect dubious senders?
2. How to deal with a possible third party sending dubious packets?
3. How to prevent man-in-the-middle attacks?

One method is to include the needed security information in packet headers. This method naturally makes the overall performance little poorer, as the header length evidently grows when incorporating more information. Other possible method would be encryption of the whole payload, or the combination of these two methods.

Identifying the sender could indeed be implemented by header identification with perhaps encrypted header. The downside is that in the case where feedback channel does not exist at all, the receivers need to have proper sender ID's preprogrammed.

In IETF, security multicast (msec) working group provides specifications for securing group communications over IP multicast, in particular in the global Internet. RFC 3740 [16] provides the reference architecture.

All in all, any security mechanism causes the overall complexity of the transmission process to grow. It is in the hands of implementor of a specific application to decide what is the proper trade-off between security and complexity.

5.5 The Need for Feedback Channel

One of the strengths of the erasure codes is that no feedback channel is in theory required. The missing pieces are not explicitly asked by the recipients and the only task is to collect enough packets to be able to decode the original message.

Depending on the code and application, there might still be need for some kind of mechanism to communicate backwards to the sender. The completion of the file reception for example would require notification of the sender, in order to stop the packet

flooding. Also, in case of some codes with varying parameters in different cases, there should be a handshaking phase, where the possible parameters of communication are agreed upon. This could include for example changing of keys for security algorithms and so on.

In some cases it might be correct not to implement any kind of feedback channel at all, and the sender just sends enough packets to make sure that the recipients can decode the message with high probability. The successful receiving of packets depends on the channel properties (loss rate), and with the absence of feedback channel, the information has to be given somehow to the sender. This means that wrong operation model could result in the recipient never receiving the complete message, or sender spraying packets too long time.

5.6 Patents

One barrier to widescale adoption of the different erasure coding methods is the myriad of patents acquired by different parties. Especially the the technologies researched by the Digital Fountain Inc. are patented and the commercial user of these technologies has to acquire licenses. Some issued patents include, but are not limited to [20, 21, 22, 3, 12, 13]. These effectively cover the methods used in LT and Raptor codes, perhaps most importantly the use of irregular graphs in code generation.

The patents have caused other attempts to create efficient erasure codes based on irregular graphs to become extinct and probably has discouraged many researchers to not pursue research in this direction. However, at the same time, other ways to perform efficient coding to approximate the digital fountain, a concept described in Section 3.3.1, have evolved. Regular LDPC codes are one example of such efficient codes. The patent limitations will some day disappear, but until then it is in the hands of the patent owners to protect their intellectual property.

Because of the patents, public implementations of LT or Raptor codes do not exist. This means that performance comparison to other codes similar in overall function but not using the patented techniques is hard if not impossible. It is also interesting to see what effect the including of Raptor codes in different standards of authorized standardization bodies has. Quite recent specifications [5] and [24] describe the use of Raptor codes in 3rd generation mobile networks and on the Internet. Will companies and network specialists widely adopt these standards and obtain the necessary licenses from Digital Fountain, or will other, possibly free and/or open source methods turn out to be more alluring choice, remains an open question.

To our knowledge none of Digital Fountain's patents have yet been used against a company or an individual to this day. Due to this reason, the issued patents have not yet been tested in a court of law, so it is partly unclear which techniques really are off limits from public implementations. This question and the some of the points discussed here are also presented for example in [32, 38].

Chapter 6

Optimization of the Degree Distribution

An optimization method and algorithm for optimizing the degree distribution used in LT codes is presented in this chapter. The approach taken is to start from small cases, where the size of the message in blocks is small and to gradually increase the length to see how far we can go with the presented methods. The proposed optimization algorithm uses results from importance sampling theory to generate an estimate for the expectation to the number of packets needed for LT coding process to successfully decode the message. This way, the expectation is parameterized, and the parameters can then be optimized using standard optimization methods, such as method of steepest descent and bisection search. First a brief introduction to importance sampling is provided followed by the idea and presentation of the developed optimization algorithm. The results achieved by using the methods presented in this chapter follow up in Chapter 7.

6.1 Importance Sampling

Importance sampling (IS) belongs to the family of Monte Carlo methods, which are simulation methods used to either generate samples from a (usually complicated) probability distribution or to efficiently estimate the expectations of functions of a random variable $\mathbf{X}$. IS is used for the latter task as a variance reduction technique to decrease the amount of samples needed for a successful estimation of the desired expectation as is explained in [42, Chapter 4], a good reference on many variance reduction techniques. However, this property is not interesting in our scenario. What we want to borrow from the IS theory is the general concept of importance sampling: samples generated with one probability distribution can be used to estimate some expectation with a different probability distribution. It should be noted, however, that the use of a distribution completely different from the original one results in a poor estimate, i.e., the used distribution should have some resemblance to the original distribution.

Let us consider the situation where we want to calculate the expectation of a function

$h(\mathbf{X})$ of the random variable $\mathbf{X}$, with probability density function $p(\mathbf{x})$:

$$E[h(\mathbf{X})] = \int h(\mathbf{x})p(\mathbf{x})\ d\mathbf{x}. \tag{6.1}$$

By drawing samples $\{\mathbf{X}^{(i)}\}_{i=1}^{K}$ from $h(\mathbf{x})$, we could calculate an estimate for the expectation:

$$\hat{h} = \frac{1}{K}\sum_{i=1}^{K} h(\mathbf{X}^{(i)}). \tag{6.2}$$

In IS, $p(\mathbf{x})$ is replaced by another probability distribution $g(\mathbf{x})$. Let the random variable obeying this distribution be $\tilde{\mathbf{X}}$. We can write (6.1) equally as

$$E[h(\mathbf{X})] = \int h(\mathbf{x})\frac{p(\mathbf{x})}{g(\mathbf{x})}g(\mathbf{x})\ d\mathbf{x}. \tag{6.3}$$

This shows that we can generate samples $\{\tilde{\mathbf{X}}^{(i)}\}_{i=1}^{K}$ from $g(\mathbf{x})$ and use these to calculate the estimate (6.2)

$$\hat{h} = \sum_{i} w(\tilde{\mathbf{X}}^{(i)})h(\tilde{\mathbf{X}}^{(i)}), \tag{6.4}$$

where $w(\tilde{\mathbf{X}}^{(i)})$ denotes the *importance ratio* (also *likelihood ratio* in some references)

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{g(\mathbf{x})}, \tag{6.5}$$

where we assume that $g(\mathbf{x}) > 0\ \forall\ \mathbf{x} : p(\mathbf{x})h(\mathbf{x}) \neq 0$. The variance of estimate (6.4) can be lower than that of in (6.2) with an appropriate choice of the sampling function $g(\mathbf{x})$.

The concept of importance ratio and the general idea used here is exploited in the following sections to generate an optimization strategy.

6.2 Objectives of Optimization

The natural goal in optimizing the degree distribution of LT code is to make the number of overhead packets needed as small as possible. This reasoning leads to the definition of two optimization goals pursued in this work. The first, and perhaps the more natural goal, is to try to minimize the average number of packets needed for successful decoding in general. Using this optimization strategy we arrive at degree distributions which give as low overhead of packets as possible for original data of size n chunks. The other goal is to maximize the probability for a successful decoding in at most $n+k$ steps (i.e., $n+k$ sent packets), where k can be adjusted from zero to an arbitrary value. We will see that these two objectives generate slightly different results and average number of packets needed for successful decoding. Of course, we have to choose which one of the goals we pursue as it is obviously impossible to find a degree distribution which is optimal with respect to both strategies at the same time.

Definition 6.1 (Min.Avg., Max.Pr.)**.** *We define two mutually exclusive objectives:*

1. *Objective **Min.Avg.** corresponds to finding a degree distribution which minimizes the average number of packets needed for a successful decoding.*

2. *Objective **Max.Pr.** corresponds to finding a degree distribution which maximizes the probability of decoding with exactly n sent packets.*

6.3 Construction of the Optimization Algorithm

6.3.1 Basic Idea

Our goal in this chapter is to optimize the number of packets needed for a successful decoding in the LT process by tuning the degree distribution $\rho(d)$. We have developed a simulation algorithm for running the LT coding and decoding processes, which is described in [45]. This algorithm takes a degree distribution as an input parameter and returns the number of packets needed for successful decoding of the original message and the number of packets of each different degree. We can consider the simulator as a black box as shown in Figure 6.1.

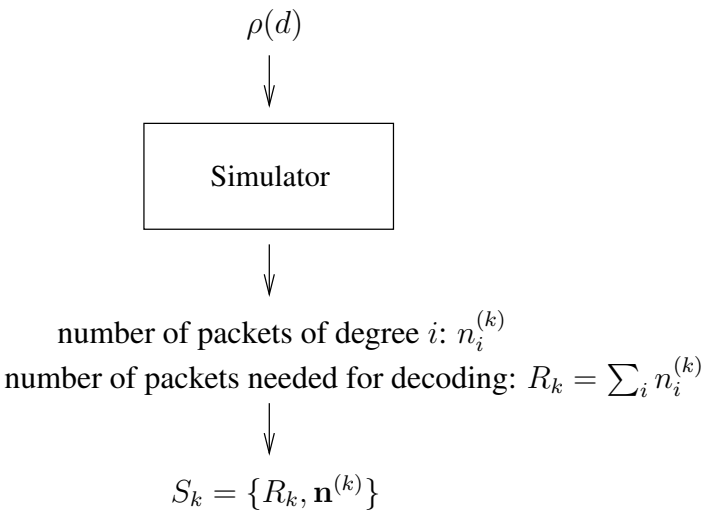

Figure 6.1: Simulator component generates samples S_k, which include the information about the number of packets needed for decoding, R_k, and vector describing the number of different degree packets received in the decoding process, $\boldsymbol{n}^{(k)}$

The first step is to construct an estimator for the average number of packets needed using the available information of m simulated samples $S_k, k = 1, \ldots, m$. The goal is to construct an algorithm, which takes some degree distribution, defined by the point probabilities $p_1, \ldots, p_n$, as input and outputs a better one. The idea is to construct such an estimator $\widehat{R}$ for the average number of packets needed for decoding so that this kind of optimization is possible.

The simplest case to consider is to construct $\widehat{R}$ as a function of probability vector $\mathbf{q}$ with length n, where each component corresponds to generation probability of a packet of degree i. Now we borrow the idea of importance ratios presented in Section 6.1, and

give similar treatment to the components of $\mathbf{q}$. We arrive at the estimate:

$$\widehat{R}(\boldsymbol{q}) = \frac{1}{m}\sum_{k=1}^{m} R_k \prod_i \left(\frac{q_i}{p_i}\right)^{n_i^{(k)}}, \tag{6.6}$$

where m is the number of samples generated, k is the index of a sample, i is the index of vector $\boldsymbol{q}$ and p_i is the probability for generating a degree i packet. Note that both objectives in Definition 6.1 can be now taken into account, objective Min.Avg. needs no modifications, for objective Max.Pr. we define R_k to be one when the decoding succeeds in the defined amount of steps and zero otherwise. Equation (6.6) is now function of $\boldsymbol{q}$, a vector of probabilities, and it is differentiable, so optimization can be implemented, for example, by the method of steepest descent (gradient method). Thus we calculate the gradient of (6.6) at point $\boldsymbol{p}$, and we have the direction where the next candidate for a degree distribution would lie. After this we have to do a line search to find the optimal point in this direction. When we have taken the step towards the optimal point (according to the estimate), new set of samples can be generated using the simulator and thus the algorithm proceeds iteratively by calculating new points (degree distributions) until some convergence criterion is met. For brevity, we call the developed algorithm ISG-algorithm, the acronym ISG standing for the initials of the words importance sampling and gradient.

While the estimate $\widehat{R}$ could be directly optimized, the problem is that with finite amount of samples the estimate is not exact, especially when $\boldsymbol{q}$ is far from $\boldsymbol{p}$, i.e., the estimate has large variance. The problem of the amount of samples is the downside of this optimization strategy, especially when the number of optimized parameters is large. First of all the generation of samples is not immediate, and secondly with very large amount the computation and memory requirements are so large that efficient implementation is difficult to make. Nonetheless, with finite sample sizes we still can get reasonable results as we will see.

6.3.2 Gradient and Projection Vectors

The component i of the gradient of the estimate $\widehat{R}(\mathbf{q})$ is:

$$\widehat{g}_i = \frac{\partial \widehat{R}}{\partial q_i} = \frac{1}{m}\sum_{k=1}^{m} R_k n_i^{(k)} \frac{1}{p_i}\left(\frac{q_i}{p_i}\right)^{n_i^{(k)}-1}. \tag{6.7}$$

When this is evaluated at point $\boldsymbol{q} = \boldsymbol{p} \;\Leftrightarrow\; q_i = p_i \;\forall\; i$ we have:

$$\left(\frac{\partial \widehat{R}}{\partial q_i}\right)_{\boldsymbol{q}=\boldsymbol{p}} = \frac{1}{m}\sum_{k=1}^{m} R_k \frac{n_i^{(k)}}{p_i}. \tag{6.8}$$

Next we have to ensure that if we actually take the step suggested by the gradient (6.8), i.e., we calculate $\boldsymbol{p}_{\text{new}} = \boldsymbol{p} + \boldsymbol{g}$, the resulting point $\boldsymbol{p}_{\text{new}}$ is a proper probability distribution. This means that all components $(p_i)_{\text{new}} \in [0, 1]$ and the sum of the components is one.

By projecting gradient (6.8) to hyperplane $\boldsymbol{g} \cdot \boldsymbol{e} = 0$, where $\boldsymbol{e}$ is a vector of ones we guarantee that the sum of the components is equal to one. To take care of the other requirement, we limit the change of each component relatively so that the value does not decrease below zero. This combined with the hyperplane projection guarantees that every component $(p_i)_{\text{new}} \in [0, 1]$. By using these restrictions, we ensure that the gradient points in the right direction in space, where each point corresponds to a probability distribution.

Unit normal vector of the hyperplane is obtained by calculating the gradient of the dot product and dividing this by its length:

$$\boldsymbol{n}_l = \frac{\nabla (\boldsymbol{g} \cdot \boldsymbol{e})}{\| \nabla (\boldsymbol{g} \cdot \boldsymbol{e}) \|} = \frac{\boldsymbol{e}}{\sqrt{\boldsymbol{e} \cdot \boldsymbol{e}}} = \frac{1}{\sqrt{n}} (1, \ldots, 1)^{\mathsf{T}} \tag{6.9}$$

By subtracting the components in the direction of normal vector from the gradient we arrive at the projection:

$$\boldsymbol{g}_{\text{proj}} = \boldsymbol{g} - (\boldsymbol{g} \cdot \boldsymbol{n}_l)\, \boldsymbol{n}_l = \boldsymbol{g} - \frac{1}{n} (\boldsymbol{g} \cdot \boldsymbol{e}) \boldsymbol{e}. \tag{6.10}$$

For component i:

$$(g_i)_{\text{proj}} = g_i - \frac{1}{n} \sum_{i=1}^{n} g_i = \frac{1}{m} \sum_{k=1}^{m} \underbrace{R_k \left(\frac{n_i^{(k)}}{p_i} - \frac{1}{n} \sum_{i=1}^{n} \frac{n_i^{(k)}}{p_i} \right)}_{s_i^{(k)}}. \tag{6.11}$$

The expression $s_i^{(k)}$ can be considered as one sample of the projected gradient. The estimated value of the gradient projection is then the calculated sample mean as given in Equation (6.11). As the estimate (6.6) is calculated from simulation samples, we need a criterion for the amount of samples we want to use for calculating the actual estimate. In our optimization algorithm we use the standard deviation of the projected gradient vector as a measure for the amount of samples to be generated. We need to calculate the sample standard deviation for the projected gradient, and use this value to control the accuracy. By modifying the criterion for the accuracy we can strike at the balance between the accuracy of the calculated distributions and practical running times of the algorithm. The results and equations presented next are fundamental results of statistics, an useful reference is [31].

We generate the samples $s_i^{(k)}$ as one long simulation run, where the sample variance is given by:

$$\sigma_i^2 = E\left[(\mathbf{X} - \mu)^2 \right] = \frac{1}{m} \sum_{i=1}^{m} \left(s_i^{(k)} - (g_i)_{\text{proj}} \right)^2, \tag{6.12}$$

and the sample standard deviation:

$$\sigma_i = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} \left(s_i^{(k)} - (g_i)_{\text{proj}} \right)^2}. \tag{6.13}$$

To calculate the standard error of the mean (6.11), we use the following result:

$$\mathrm{Var}\bar{\mathbf{X}} = \frac{\sigma_i^2}{m}, \tag{6.14}$$

where m is the number of samples and $\bar{\mathbf{X}}$ is the sample mean of the random variable $\mathbf{X}$. Hence, the standard error of the mean value of the samples of the projected gradient is:

$$\sigma_{g_i} = \sqrt{\frac{1}{m(m-1)} \sum_{k=1}^{m} \left[R_k \left(\frac{n_i^{(k)}}{p_i} - \frac{1}{n} \sum_{i=1}^{n} \frac{n_i^{(k)}}{p_i} \right) - (g_i)_{proj} \right]}. \tag{6.15}$$

On the other hand this estimate can be represented in a way more convenient for this algorithm:

$$\sigma_{g_i} = \sqrt{\frac{1}{m(m-1)} \sum_{k=1}^{m} \left(s_i^{(k)} \right)^2 - \frac{1}{m} \left(\sum_{k=1}^{m} s_i^{(k)} \right)^2}. \tag{6.16}$$

This latter form is used when generating new samples. The form (6.16) is faster to use because both sums are easy to calculate as running sums during the execution of the algorithm and thus is more practical than form (6.15) to be used in an actual implementation of the algorithm.

6.3.3 Line Search for Step Length Calculation

It is not the best strategy to use the gradient as a step as it is, i.e., calculate directly $\boldsymbol{p}_{\text{new}} = \boldsymbol{p} + \boldsymbol{g}$ but instead a line search in the direction of the gradient should be performed to find the optimum. This means that we want to find a λ such that

$$f(\lambda) = \widehat{R}\left(\boldsymbol{p} + \lambda \boldsymbol{g}\right), \tag{6.17}$$

is optimized, where $\boldsymbol{p}$ is the starting point and $\boldsymbol{g}$ the calculated (projected) gradient. Thus f represents the one-dimensional function in the direction of the gradient.

Many line search methods for this kind of problem exist, see for example [7]. A bisection search is proposed here for this problem. Earlier we also used a Newton-Rhapson step to do the final guess after the bisection search, but the calculation of the second derivate in addition to the first derivate needed in the bisection search turned out to be unpractical and inefficient when the generated sample sizes and the dimension of the problem is large. It is more efficient to make the accuracy goal of the bisection search smaller instead.

Bisection search uses derivatives to find the optimum (minimum/maximum) of a function of one variable in a specified interval $[a_1, b_1]$. The basic idea is to calculate the derivative of the function in the middle point of the interval. Based on the sign of the result, we define a new interval and continue calculating again the derivative in the middle point. This way the interval becomes smaller at every step and we finally stop when the accuracy is enough for us.

As an example case of how the bisection search works, we look into the maximization

of a differentiable function f:

- If derivative at (middle) point $\lambda : f'(\lambda) = 0$, then point λ is maximum (or minimum).
- If derivative $f'(\lambda) < 0$, then the maximum is left of λ.
- If derivative $f'(\lambda) > 0$, then the maximum is right of λ.

These three cases cover all possible scenarios and lead to the following algorithm:

1. Let $c_k = \frac{1}{2}(a_k + b_k)$. If $f'(c) = 0$, then c is either maximum or minimum $\rightarrow$ stop.
2. If $f'(c) > 0$, go to step 3, otherwise go to step 4.
3. Change of the left bound $\rightarrow a_{k+1} = c_k, b_{k+1} = b_k$. Go to step 5.
4. Change of the right bound $\rightarrow a_{k+1} = a_k, b_{k+1} = c_k$.
5. If $k = n$ stop, optimum lies in the interval $[a_{k+1}, b_{k+1}]$, otherwise increase k by one and go to step 1.

The execution of the algorithm is controlled through parameter n, which describes how many iterations of the algorithms are executed. Usually we want to define a threshold value l so that the algorithm will stop when the length of the interval $[a_{k+1}, b_{k+1}]$ is less than l. We can calculate the number of steps needed to achieve this:

$$\begin{aligned} \left(\frac{1}{2}\right)^n &\leq \frac{l}{b-a} \\ n \log \frac{1}{2} &\leq \log l - \log(b-a) \\ n &\leq \frac{\log(b-a) - \log l}{\log 2} \\ \Rightarrow n &= \left\lceil \frac{\log(b-a) - \log l}{\log 2} \right\rceil . \end{aligned} \tag{6.18}$$

6.3.4 Possible Problems with the Line Search

The bisection search presented above allows us to control the precision of the step via the interval $[a_1, b_1]$ and the threshold value l. However, there are some points which we should consider when implementing a line search method (which could be something entirely different than the bisection search proposed above):

1. If f has local extremum points, the method may converge to one of these.
2. Method needs an initial interval $[a_1, b_1]$. What is a good candidate for this interval?

Both of these can be usually countered. If the method converges to a local maximum, it does not hurt the global convergence properties of the whole ISG-algorithm, the latter being an iterative algorithm. Convergence to a local maximum then just slows the convergence a little bit. With the generation of new samples, the local optimum is likely to smooth out. If this is not the case, and there really is some kind of local optimum even with infinite amount of samples, then the convergence to a local optimum cannot be avoided using the presented algorithm.

The second point is also not a major problem largely by the same reasoning. We can fix a predetermined interval and use it all the time during the execution of the ISG-algorithm. It might happen that the real extremum is really outside the interval; the line search will then converge to the end point of the interval. Test scenarios have shown that interval $[0, 1]$ is a good candidate, when using a normalized gradient. This interval will be used if not mentioned otherwise.

6.3.5 Modifications for Parameterized Distributions

The idea in Section 6.3.1 can be extended to include parameterized distributions instead of a general one where point probabilities are the parameters.

Let $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ be vectors of n parameters, which define a degree distribution. We define the estimate in (6.6) again using parameterized probability distributions:

$$\widehat{R}(\boldsymbol{\eta}) = \frac{1}{m}\sum_{k=1}^{m} R_k \prod_i \left(\frac{p_i(\boldsymbol{\eta})}{p_i(\boldsymbol{\theta})}\right)^{n_i^{(k)}}, \tag{6.19}$$

where $\boldsymbol{\eta}$ denotes the starting point in parameter space, serving similar function as $\boldsymbol{q}$, in Sections 6.3.1 and 6.3.2. The gradient is now computed by differentiating with respect to parameters η_i, using the chain rule for product differentiation:

$$\widehat{g}_j = \frac{\partial \widehat{R}}{\partial \eta_j} = \frac{1}{m}\sum_{k=1}^{m} R_k \sum_i \frac{n_i^{(k)}}{p_i(\boldsymbol{\theta})} \left(\frac{p_i(\boldsymbol{\eta})}{p_i(\boldsymbol{\theta})}\right)^{n_i^{(k)}-1} \frac{\partial p_i(\boldsymbol{\eta})}{\partial \eta_j} \prod_{l\neq i} \left(\frac{p_l(\boldsymbol{\eta})}{p_l(\boldsymbol{\theta})}\right)^{n_l^{(k)}}. \tag{6.20}$$

When this is evaluated at $\boldsymbol{\eta} = \boldsymbol{\theta}$ we arrive at a much simpler form:

$$\left(\frac{\partial \widehat{R}}{\partial \eta_j}\right)_{\boldsymbol{\eta}=\boldsymbol{\theta}} = \frac{1}{m}\sum_{k=1}^{m} R_k \sum_i \frac{n_i^{(k)}}{p_i(\boldsymbol{\eta})} \frac{\partial p_i(\boldsymbol{\eta})}{\partial \eta_j}. \tag{6.21}$$

This closely resembles the form in (6.8), but in addition the information of the derivative of the parameterized distribution is needed.

Let us consider an example case of the parameterized optimization with the degree distribution:

$$p_i(\eta) = \frac{e^{-\eta i}}{\sum_{i=1}^{n} e^{-\eta i}}. \tag{6.22}$$

We have a valid reason for choosing this geometric distribution. As we will see, the optimized forms for small cases, where the size of the message $n \leq 10$, the degree dis-

tribution resembles the form of (6.22), so this might be a good candidate for optimizing the degree distribution for larger number of blocks, from $n = 10$ to $n = 100$ and even larger.

Differentiating (6.22) with respect to the parameter η yields the gradient:

$$(g_i)_{\text{param}} = \frac{\partial p(\eta)}{\partial \eta} = \frac{e^{-\eta i} \sum_{i=1}^{n} i e^{-\eta i}}{\left(\sum_{i=1}^{n} e^{-\eta i}\right)^2} - \frac{i e^{-\eta i}}{\sum_{i=1}^{n} e^{-\eta i}} \tag{6.23}$$

Substituting this derivative into (6.21), yields:

$$\left(\frac{\partial \widehat{R}}{\partial \eta_j}\right)_{\boldsymbol{\eta}=\boldsymbol{\theta}} = \frac{1}{m} \sum_{k=1}^{m} R_k \sum_i n_i^{(k)} \frac{(g_i)_{\text{param}}}{p_i(\boldsymbol{\eta})}. \tag{6.24}$$

This is a good form from the efficient implementation point of view. The last fraction can be easily calculated beforehand, hence the calculation of the gradient is only the matter of inserting the generated samples into (6.24).

Optimization in this example is a problem in one dimension, and using the ISG-algorithm might be an overkill as line-search methods can be used directly for optimizing the only parameter. However, as more parameters are introduced the optimization becomes more complicated and the calculation of the gradient vector similarly as when optimizing the point distributions becomes necessary. The ISG-algorithm presented in the next section can be used for both point and general parameterized optimization.

A simple algorithm for optimization with regard to the parameter η in (6.22) can be performed as follows:

1. Choose a starting value η_0. Set $i \leftarrow 0$.
2. Generate samples using distribution $p(\eta_i)$.
3. Construct the estimate for the average number of packets $\widehat{R}$.
4. Use bisection search to find the optimal λ minimizing $\widehat{R}(\lambda)$.
5. Use the found optimum $\tilde{\lambda}$ as a new parameter value: $\eta_{i+1} \leftarrow \tilde{\lambda}$.
6. If ending condition is true, then stop. Otherwise go to step 2.

The stopping condition is not specified yet, we propose a stopping condition based on the variance of the estimate (6.6) next in Section 6.3.6. Note that there is no need for similar control as the standard deviation of the projected gradient as discussed in Section 6.3. However, the standard deviation can still be used to provide a threshold for sample generation, i.e., to generate samples until the accuracy condition is met. The other option is to use a fixed amount of samples, but as the algorithm gets closer to the optimum, the number of samples needed for the same accuracy grows up, and the accuracy criterion takes care of generating more samples when the algorithm has already converged near to the optimum. Examples of different parameterized forms and optimization results are presented in Chapter 7.

6.3.6 Stopping Condition

The estimate $\widehat{R}$ is calculated from simulation results as shown in Equation (6.6). When considering the convergence of the algorithm, we take into account the "noise" in the simulation, that is, the fact that this is only an estimate based on some finite amount of simulation results. We can approximate this noise as the standard deviation of the estimate of the average number of the packets needed for decoding $\widehat{R}$. The calculation is performed in the same way as presented in Section 6.3.2 with the projected gradient vector:

$$\sigma_{\widehat{R}} = \sqrt{\frac{1}{m(m-1)} \sum_{i=1}^{m} \left(R_k \prod_i \left(\frac{q_i}{p_i} \right)^{n_i^{(k)}} - \widehat{R} \right)^2}. \tag{6.25}$$

Now, after every iteration of the ISG-algorithm, we calculate the estimate for the average number of packets $\widehat{R}$ needed for a successful decoding. We compare the last two values of $\widehat{R}$ and if their absolute difference is smaller than the standard deviation $\sigma_{\widehat{R}}$, we stop the algorithm.

This means that we stop the algorithm when the approximated noise in the simulation is larger than the difference between the last two calculated estimates. The noise overwhelms the difference, accordingly we can conclude that the last two estimates are calculated using degree distributions as near the optimal ones as we can get to with the given number of samples.

6.4 ISG-Algorithm

6.4.1 Overview of the Algorithm

This section will present a general framework of the ISG-algorithm. The actual algorithm consists of several sub-algorithms, most of which are well-known (bisection search, gradient based optimization etc.). The actual idea of the algorithm is simple as discussed earlier, some complexity occurs from the many parameters which needs to be set to control the convergence of the algorithm and the generation of the samples.

In the case of point distribution optimization, the dimension of the problem at hand is the number n of the blocks in the message to be transferred. The vectors manipulated during the execution are thus of length n in the case where point distributions are optimized directly. The optimization then takes place in space where points are degree distributions. Criteria for a point to belong to this space is that all components are positive and all components sum up to one.

With parameterized degree distributions, the number of vector components is the number of parameters used. The space is now the parameter space, each point corresponding to different values of parameters. It might be reasonable to restrict this space somewhat in certain scenarios, for example not allowing negative values for parameters.

The following description summarizes both point and parameterized distribution optimization. The algorithm takes as an input some degree distribution, defined by point

probabilities $p_1, \ldots, p_n$, threshold ε for sample generation, and interval and threshold for bisection search.

1. Use the given probability distribution (either point or parameterized form) as a starting distribution $\boldsymbol{p}$.
2. Generate samples S using the degree distribution $\boldsymbol{p}$. Generate samples until accuracy is less than ε as described in Section 6.3.2.
3. Use the samples S to calculate the gradient $\boldsymbol{g}$. This is, in the case of point distribution, the projection of the gradient vector (6.11) or, when optimizing parameterized distributions, the gradient (6.24).
4. Optional: divide the gradient $\boldsymbol{g}$ by its length, i.e., $\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$.
5. Do a bisection search in the direction of the gradient, as described is Section 6.3.3. This means that we optimize $\widehat{R}\,(\boldsymbol{p} + \lambda \boldsymbol{g})$, either finding a minimum or maximum depending on the goal (Definition 6.1). As a result we have the step length λ.
6. The step towards a better distribution is: $\mathrm{step} \leftarrow \lambda \cdot \boldsymbol{g}$.
7. For point distributions only: limit the change of each component to 50% of the previous value. This ensures that the conditions $\sum p_i = 1$ and $p_i \in [0, 1]$ are met.
8. Take the step in the direction of the gradient: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \mathrm{step}$.
9. Calculate the value of $\widehat{R}$ using (6.6) and the standard deviation $\sigma_{\widehat{R}}$ using (6.25). If the absolute difference between the last two estimates is less than the standard deviation, then stop. Otherwise continue and go back to step 2.

6.4.2 Implementation Issues of the Algorithm

For the goal Max.Pr. an automatic implementation is easy to make. The line search (maximization) behaves well and has a clear maximum to which the algorithm converges.

This is, however, not the case with the goal Min.Avg. The interval for the line search has to be chosen carefully, otherwise the minimization can converge into a non-feasible region. The limited amount of samples causes the form of the estimate $\widehat{R}$ to have some peculiarities. For example, when optimizing the point distribution, there always exists a minimum of zero at point $\boldsymbol{q} = 0$, as can easily be seen from Equation (6.6). This means that in order to generate the results, visual inspection of the line search and proper convergence is advisable. In any case it is easy to check the results of the algorithm (i.e., proposed distributions) to discard the poor intermediate results.

Chapter 7

Results

This chapter presents simulation and optimization results using the methods presented in Chapter 6. Optimization results for a range of block numbers n are presented. The point distribution optimization focuses on the small values of n and some parameterized forms are used to optimize the distributions for larger cases. Also both of the objectives presented in Section 6.2 are considered. This chapter begins with some information on the implementation and hardware used to obtain the results, then results calculated before this work are presented and last the actual results follow.

7.1 Implementation and used hardware

The implementation of the ISG-algorithm is currently done using Mathematica version 5.2 [17]. In some of the results, especially cases where number of blocks n, and the number of generated samples m per iteration, are large, the sample generation was done using routine written in C programming language [18]. The sample generation written in Mathematica and in C are similar in function, the only difference being the running time. The C implementation is several magnitudes faster than the Mathematica version.

The results were mainly calculated using a dual core AMD Opteron 1600 MHz server with 4GB of memory using Debian Linux (kernel 2.6.14). Also the services of CSC [1] were used, some simulations were run on their Sun Fire 25K general purpose servers.

The used hardware and current implementation of the ISG-algorithm allowed us to optimize forms from $n = 3$ to $n = 100$, optimization for larger values of n takes much more time and needs more samples for generation, and is not considered in this work.

7.2 Exact analysis of cases $n = 3$ and $n = 4$

Optimal degree distributions for small cases, where the size of the message is just $n = 3$ or $n = 4$ blocks can be calculated exactly by constructing a proper state space and using Markov Chains. In [45] this was done by first constructing all possible states in these two scenarios, then reducing the set of states and finally calculating the average steps needed for the Markov Chain to converge to an absorbing state. The set of states can be reduced by noticing that many states have similar structure, as they can be obtained from each

other by a permutation of the source packets, and thus they can be combined into one unique representation of those states. This way the number of states can be decreased considerably and the Markov Chain calculations become feasible.

The results of these calculations are presented here, and are later used as comparison for results obtained using the ISG-algorithm. Table 7.1 shows the optimal weights for case $n = 3$ and respectively Table 7.2 for case $n = 4$. Also the mean values of steps needed for decoding are given ($E[\cdot]$), in addition to the probabilities for the success in exactly n steps ($\mathcal{P}$).

Table 7.1: Optimal weights in case $n = 3$ [45]

	Min.Avg.	Max.Pr.
p_1	0.524	0.517
p_2	0.366	0.397
p_3	0.110	0.086
E[steps]	4.046	4.049
$\mathcal{P}_3$	0.451	0.452

Table 7.2: Optimal weights in case $n = 4$ [45]

	Min.Avg.	Max.Pr.
p_1	0.442	0.429
p_2	0.385	0.430
p_3	0.112	0.100
p_4	0.061	0.041
E[steps]	5.580	5.590
$\mathcal{P}_4$	0.314	0.315

We note that the two different objectives provide almost the same statistics (the expectation and the decoding probability in exactly n steps) for $n = 3$ and $n = 4$. It seems to be the case that small variations in different degree probabilities does not change the expecation or the probability much.

7.3 Optimizing the Point Probabilities

The ISG-algorithm can be directly used to optimize the degree distribution starting from an arbitrary distribution. However, when n increases, this method becomes computationally too intensive, as the point probabilities are the parameters and thus there are n parameters[1] in total. Nevertheless, some results for small values of n can be generated with direct optimization of the point probabilities.

[1] Actually the optimization can be made with regard to $n - 1$ parameters because the sum of the probabilities has to be one, an additional restriction.

7.3.1 Maximizing the Decoding Probability

For $n = 3, \ldots 10$, the ISG-algorithm works well with point distributions when maximizing the decoding probability at most n steps (objective Max.Pr. in Definition 6.1). The ISG-algorithm was run with hard limit of 2000000 samples per iteration, with sample generation threshold $\varepsilon = 0.005$, meaning 0.5% accuracy in the projected gradient vector. This threshold turned out to be near the balance, where the generation stops before hitting the hard limit of 2000000 samples. The line search was performed in the interval $[0, 1]$, and the projected gradient was normalized before the line search (step 4 in Section 6.4.1). Line search was stopped when the result was at maximum 0.0001 from the optimum. Maximum number of iterations was set to 15, in order to stop the algorithm if the stopping condition was not met before.

The optimized distributions were the evaluated by 100000 runs of the LT process. Averages of these runs are presented in Table 7.4. Results are also compared to uniform distribution, which was used as the starting distribution for the ISG-algorithm.

A plot of overhead percentages is presented in Figure 7.2, standard deviations of the overheads are included as error bars in Figure 7.3. In Table 7.3 the optimized values of different parameters are presented, these are additionally plotted in Figure 7.1. In the range $n = 3, \ldots, 10$, we see that the distributions follow a geometric form. Figure 7.4 shows the overhead percentages for both optimized distributions, and for the uniform distribution, i.e., the starting distribution. We see that our algorithm greatly enhances the uniform distribution, which produces really bad results in terms of overhead.

The results show that the achieved overhead in the range $n = 3, \ldots 10$ is not very good with LT codes, the overhead percentage being between 35% and 50%. If one wants to operate with such small block numbers, other erasure codes should probably be used.

Table 7.3: Optimized weights for degree distributions for $n = 3, \ldots, 10$

n	3	4	5	6	7	8	9	10
p_1	0.516	0.429	0.407	0.368	0.335	0.311	0.287	0.269
p_2	0.397	0.430	0.341	0.303	0.282	0.267	0.250	0.239
p_3	0.087	0.100	0.183	0.187	0.185	0.180	0.177	0.173
p_4		0.041	0.068	0.101	0.112	0.117	0.120	0.119
p_5			0.001	0.041	0.061	0.071	0.078	0.081
p_6				0.000	0.025	0.038	0.050	0.056
p_7					0.000	0.015	0.027	0.034
p_8						0.000	0.011	0.020
p_9							0.000	0.009
p_{10}								0.000

It is worth noting, that the generated distributions for cases $n = 3$ and $n = 4$ match the optimal calculated distributions for maximization case presented in Table 7.1 and Table 7.2 (goal Max.Pr.). These results confirm that our importance sampling based method works correctly and, on the other hand, that the ISG-algorithm works correctly.

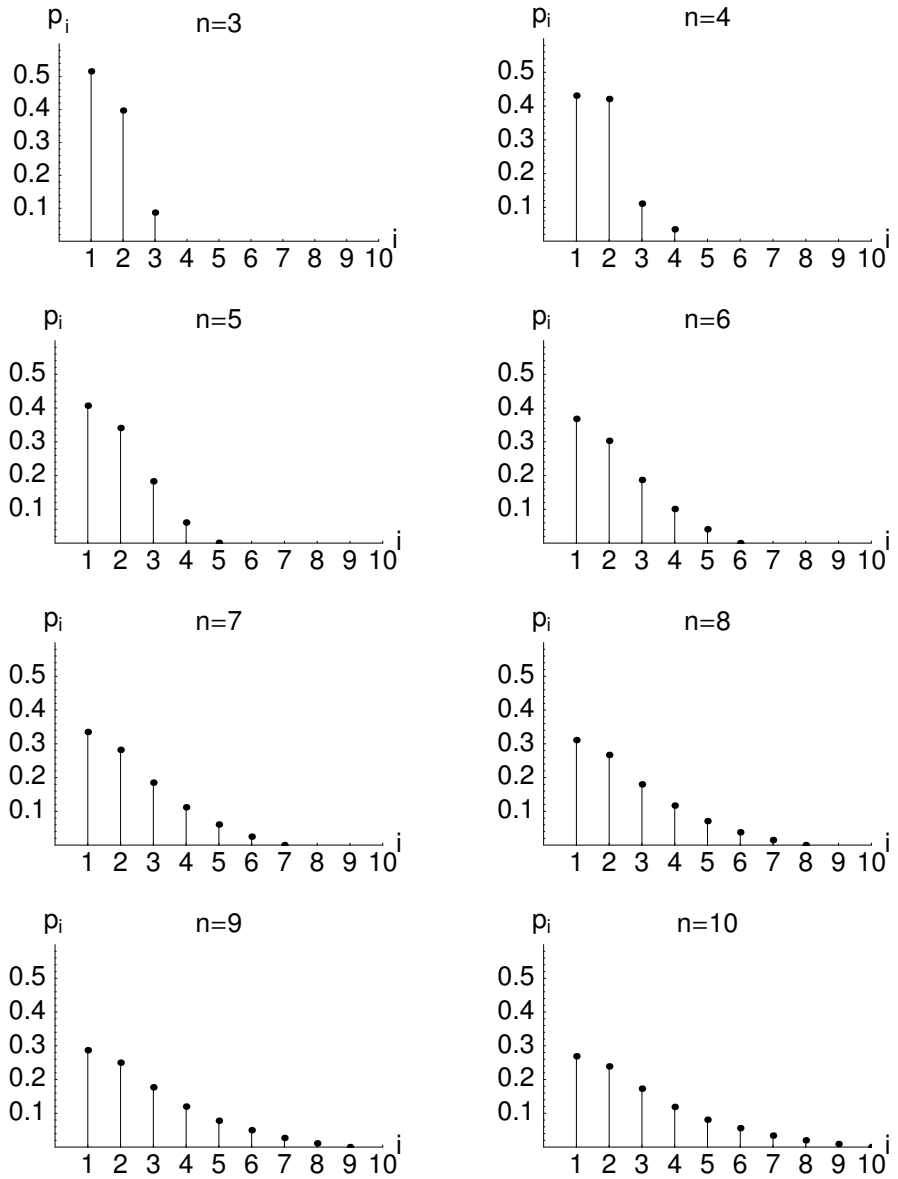

Figure 7.1: Plots of distributions resulting from point optimization for $n = 3, \ldots, 10$. The probability of decoding in at most n steps is maximized.

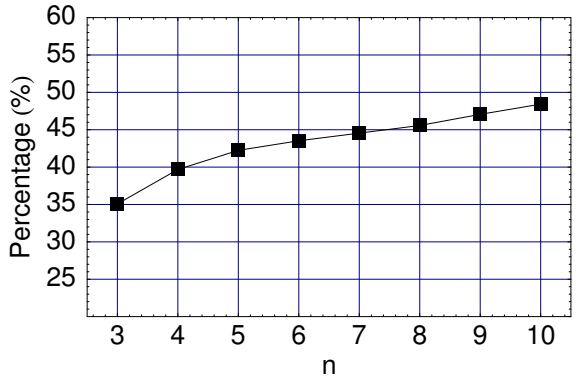

Figure 7.2: Relative overheads with optimized point distributions when the probability of decoding is maximized at most n steps for $3 \leq n \leq 10$. LT process was run 100000 for each optimized distribution, the plotted results are average overheads of packets needed for decoding.

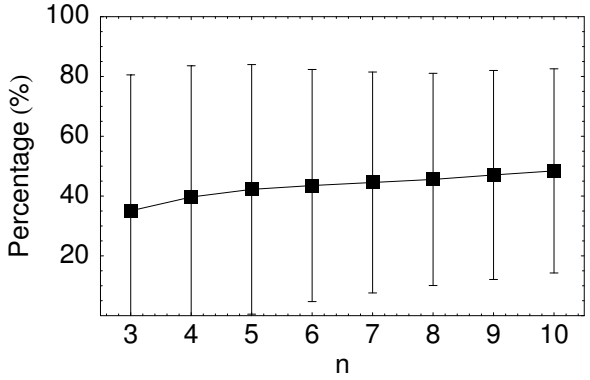

Figure 7.3: Relative overheads with error bars representing standard deviations for $n = 3, \ldots, 10$.

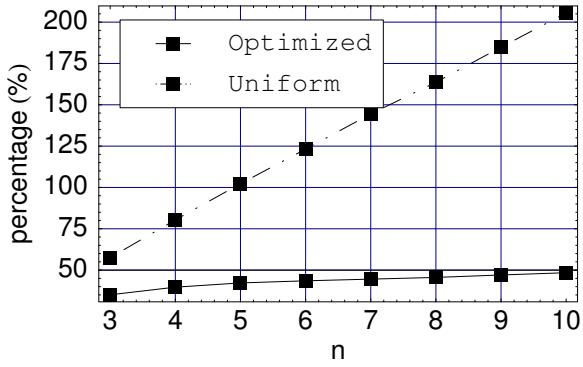

Figure 7.4: Combined results with optimized point distribution and the uniform distribution, which was used as a starting point. The algorithm has succeeded in optimization by reducing the overhead percentages.

Table 7.4: Simulation results with different distributions for cases $n = 3, \ldots, 10$.

	Averages from 100000 simulations with each distribution			
n	Uniform dist.	Opt. dist.	Overhead packets	Overhead %
3	4.71	4.05	1.05	35.1
4	7.15	5.59	1.59	39.7
5	10.09	7.11	2.11	42.2
6	13.30	8.61	2.61	43.5
7	17.00	10.12	3.12	44.5
8	21.10	11.64	3.64	45.6
9	25.57	13.23	4.23	47.1
10	30.52	14.84	4.84	48.42

7.4 Optimizing Parameterized Distributions

7.4.1 Parameterized Form $e^{-\eta i}$

The parameterized form

$$p_i = \frac{e^{-\eta i}}{\sum_i e^{-\eta i}} \tag{7.1}$$

is a simple parameterized form and optimization results for this form are quite easy and fast to generate. We again consider the two different goals introduced in Definition 6.1 in their own sections.

7.4.2 Maximizing the Decoding Probability

There are some issues when optimizing for maximum decoding probability. The probability for the decoding to succeed in exactly n steps approaches zero as n grows. In point distribution optimization, this is not a major problem, although in the cases where n is near 10, major part of the simulator results produce coefficient $R_k = 0$ in (6.6), thus rendering large part of the simulation results useless.

For the parameterized form (7.1) results were calculated for cases $n = 10, 20, 30$ and 40, where probability of decoding is maximized for exactly n steps. Algorithm was ran with hard limit of 1000000 samples per round, 10 rounds at maximum. Generation threshold ε was set at $\varepsilon = 0.1$. Line search interval was $[0, 0.05]$ with stopping threshold of 0.0001.

The optimization results for these cases are presented in Table 7.5. There is no point to continue this for larger values of n, as even for $n = 40$ only approximately 0.01% of the simulation results generate successful decoding in 40 steps, when the simulations are started from parameter value $\eta = 0.33$. This would mean that even with large amount of samples, say 10^7, only 1000 samples would give successful decoding, resulting in high inefficiency of the ISG-algorithm.

For maximization of large values of n we can relax the requirement of decoding in

Table 7.5: Useful samples when maximizing the decoding probability in exactly n steps.

n	η	useful samples
10	0.476	4.6%
20	0.387	0.44%
30	0.350	0.06%
40	0.33	0.01%

exactly n steps to $n + k$ steps, where k can be anything $k \geq 1$. Of course, the larger the value of k, the more samples will take part in forming the estimate of $\widehat{R}$, thus giving more accurate optimization results. The situation however is not analogous to maximizing the probability of decoding in at most n steps, as the algorithm maximizes for the conditions asked for, that is, decoding in at most $n + k$ steps.

Because of this inefficiency when maximizing the decoding probability, we will focus on the minimization of the average number of packets needed for decoding from this point on.

7.4.3 Minimizing the Average Number of Needed Packets

The optimization was performed using ISG-algorithm with generation threshold $\varepsilon = 0.1$, with hard limit of 1000000 samples. The bisection search was performed on interval $[0, 0.1]$ with threshold 0.001, which translates to 10 iterations in (6.3.3). The resulting parameters for $n = 10, 20, \ldots, 100$ are presented in Figure 7.5. Note that the results needed some manual purification, as in some cases the automatic minimization algorithm optimized the parameter values outside the feasible range. The values which produced the lowest overheads were chosen.

Using the parameters plotted in Figure 7.5, we did 10000 simulations of the the LT process and the resulting averages of packets needed for decoding are presented in Figure 7.6. In Figure 7.7 we have also introduced the standard deviations of the simulations into the plot. This is otherwise the same as Figure 7.5. The results show that with increasing n, the overheads shrink a little, as do the standard deviations. Still overhead of nearly 40% is probably not satisfactory for applications.

In Figure 7.8 we have plotted the transfer time model presented in Section 5.2. The data size here is $M = 10^5$ bits and header length $H = 20$ bits. The figure shows that with these parameters the optimal block size to be used with this degree distribution, assuming the optimized values, is $n = 70$. Our model, however, is just a simple example, and many other factors should be taken into account in real world situations. Still, the results show that different scenarios have to be thought out to enable efficient transmission.

7.4.4 Parameterized Form $e^{-\eta_1 i} + \eta_2 e^{-\eta_3 i}$

The parameterized form

$$p_i = e^{-\eta_1 i} + \eta_2 e^{-\eta_3 i}, \tag{7.2}$$

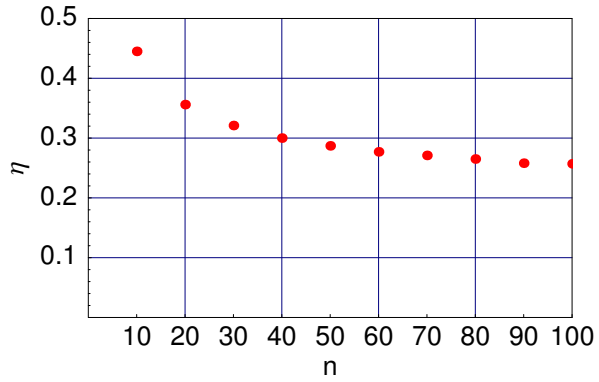

Figure 7.5: Optimized parameter values for form (7.1).

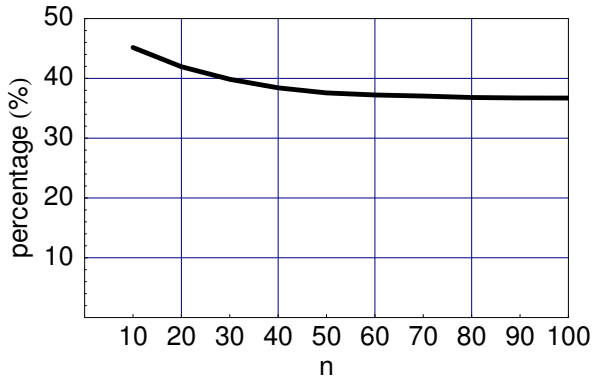

Figure 7.6: The percentages of overhead packets when optimization is done to minimize the average number of packets needed for decoding for form (7.1).

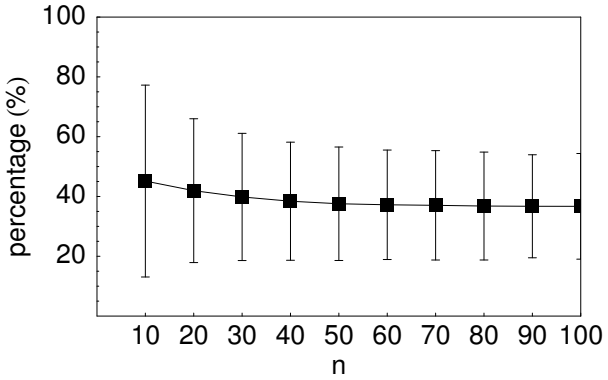

Figure 7.7: Standard deviations of simulations shown with error bars for form (7.1). Note that the error bars shrink with n growing.

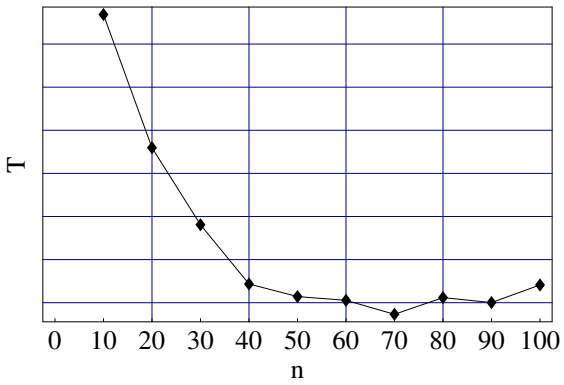

Figure 7.8: The transfer time with data size 10^5 bits and header length 20 bits

with normalization, is an enhanced form of (7.1). The added term should allow the fine-tuning of the form of the distribution, and if possible, generate better results especially when n is near 100.

However, while the ISG-algorithm works nicely with the added number of parameters, we did not find the results any better than with the optimized case with one parameter. One problem might be existing local minima, which were found by using different sets of starting parameters. Table 7.6 lists some results for the case $n = 100$. We see that many of the tested cases produce similar results. Also, it seems that parameter η_2 does not change much, if at all, during the optimization. The best results achieved, with overhead percent around 37% match the results generated with only one parameter. This means that in order to produce better distributions, the form of the parameterized distribution should be changed.

Table 7.6: The three parameter example. Starting from different sets of parameters, optimized parameters were calculated. Averages and standard deviations are calculated from 10000 runs of the LT process.

Start parameters	Optimized parameters	Avg. # of packets	Std
$\{0.4, 0.1, 0.4\}$	$\{0.26, 0.1, 0.38\}$	136.8	19.4
$\{0.4, -0.3, 0.4\}$	$\{0.27, -0.26, 0.42\}$	137.5	17.6
$\{0.4, -1.5, 0.4\}$	$\{0.44, -1.52, 0.24\}$	158.0	17.9
$\{0.9, -2.0, 0.4\}$	$\{0.90, -2.0, 0.29\}$	137.5	19.0
$\{0.4, 1.5, 0.23\}$	$\{0.34, 1.5, 0.23\}$	136.6	17.9

7.5 Forms Based on the Soliton Distributions

Even more effective forms are achieved by considering the Soliton distributions defined in Section 3.3.3. One characteristic of both Soliton distributions is that the probability for degree one symbols is less than the probability for degree two symbols. This should ensure that not too many redundant degree one packets are sent, resulting in more effi-

cient transmission. However, the Ideal Soliton distribution itself performs rather poorly, as we will see, so we give parameters to the first two degree probabilities and define the rest of our distribution to be the Ideal Soliton distribution, i.e.,

$$p_i = \begin{cases} \eta_1, \text{for } i = 1 \\ \eta_2, \text{for } i = 2 \\ \frac{1}{i(i-1)}, \text{for } i = 3, \ldots, n \end{cases} . \tag{7.3}$$

This needs to be normalized to get proper probabilities for each component.

We ran the ISG-algorithm with a similar setup as before for $n = 100$, allowing the generation of 1000000 samples if the threshold of 0.1 is not met before. Bisection search was performed in the interval $[0, 0.05]$ with threshold of 0.0001. The maximum number of iterations was again set to 15. The starting point was set at $\boldsymbol{\eta} = (0.2\ 0.2)$.

The algorithm did not meet the stopping condition, but at last results were near the value $\boldsymbol{\eta}_{\text{opt}} = (0.09\ 0.36)$. 10000 runs of the LT process defined by corresponding degree distribution shows the overhead of around 25%, clearly a better result than with the previous geometric forms.

The question of whether the spike present in Robust Soliton distribution is really necessary leads us to consider a slightly modified form of (7.3). We ran the ISG-algorithm with the same setup as above, but with an extra parameter for the probability of degree 50 packets. We started from $\boldsymbol{\eta} = (0.2\ 0.2\ 0.2)$ and arrived at the optimized parameters $\boldsymbol{\eta}_{\text{opt}} = (0.083\ 0.487\ 0.032)$. With this form the overhead is around 24%, again a slightly better result than with (7.3). Exact statistics are presented in Table 7.7, where we have also included statistics of Ideal and Robust Soliton distributions with different parameters.

Table 7.7: Results calculated from 10000 runs of LT process with the different distributions. Robust Soliton distribution shows good performance compared to the Ideal Soliton distribution, but (7.3) achieves still better performance.

Distribution	Avg # of packets	Std
(7.3)	125.0	13.1
(7.3) with spike at $i = 50$	123.9	9.9
Ideal Soliton	169.5	72
Robust Soliton, $\sigma = 0.5, c = 0.01$	148.5	44.8
Robust Soliton, $\sigma = 0.5, c = 0.03$	134.9	23.9
Robust Soliton, $\sigma = 0.5, c = 0.1$	132.9	13.3

Table 7.7 shows us that while the Robust Soliton distribution performs much better than the Ideal Soliton distribution, our algorithm was able to find even better distributions first of all by simply using the first two probabilities of Ideal Soliton distribution as free parameters, and even better by introducing a spike. The behavior of Ideal Soliton distribution, as explained in Section 3.3.3, is clearly very poor in real situations. With $n = 100$ the first two probabilities of Ideal Soliton distribution are 0.01 and 0.5, respec-

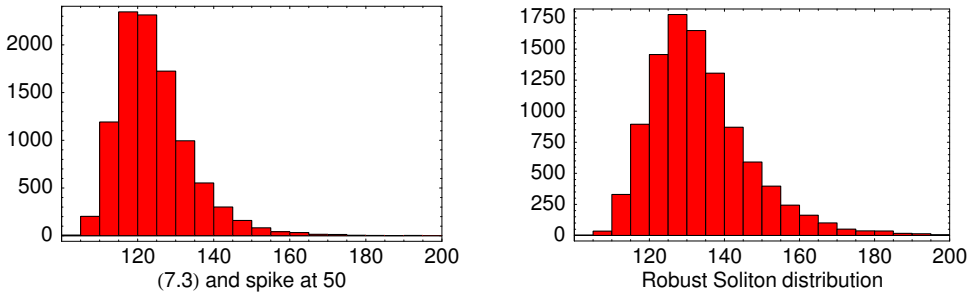

Figure 7.9: Histograms of number of packets needed for successful decoding with our best parameterized form and with the Robust Soliton distribution for $n = 100$ with 10000 simulation runs.

tively, but our results show that the optimized (and normalized) values of 0.1 and 0.38 give much better performance.

It is still unclear, however, if there exists parameters for the Robust Soliton distribution which generate better results for $n = 100$, as our algorithm cannot be used directly to optimize these parameters as the form (3.5) cannot be properly differentiated. A form where the tail distribution ensures that there is enough packets of high degrees could probably eliminate the need for the spike.

Figure 7.9 shows the histograms of 10000 simulations of the LT processes run with our parameterized distribution resembling the Robust Soliton distribution and with the real Robust Soliton distribution. With our distribution the worst case requires nearly always less than 150 packets, with Robust Soliton distribution the tail goes much further, with several hundred cases with over 160 packets. We also note that with our optimized distribution, the average number of packet degrees was 7.7, and with the Robust Soliton distribution this was 8.6. This means that less operations are needed for decoding when using the form (7.3) with spike at $i = 50$ and parameter values $\boldsymbol{\eta}_{\text{opt}} = (0.083\ 0.487\ 0.032)$, resulting in a little better decoding performance.

7.6 Tests With Larger n

As stated before, our implementation of the ISG-algorithm requires some optimization in order to work in reasonable time for n larger than 100. Nevertheless, we run some LT process simulations with $n = 1000$ to see how our optimized distribution for $n = 100$ performs when compared to the Robust Soliton distribution. To our surprise, (7.3) outperforms the Robust Soliton distribution with $n = 1000$, at least with wide range of tested parameters. Some test results are presented in Table 7.8.

The results show that from the tested distributions, best performance was provided by our optimized form with the spike moved to $i = 100$. Even while the form with spike at $i = 50$ provides the same overhead as the form with $i = 100$, the standard deviation is much larger. The values of the parameters used in (7.3) were the same as before, $\boldsymbol{\eta}_{\text{opt}} = (0.083\ 0.487\ 0.032)$, regardless of the location of the spike. Without the spike, we see that the standard deviation is quite high. This could imply that the spike (or at

Table 7.8: Results calculated from 1000 runs of LT process for $n = 1000$.

Distribution	Avg # of packets	Std
(7.3)	1130	84
(7.3) with spike at $i = 50$	1122	60.8
(7.3) with spike at $i = 100$	1121	37
Robust Soliton, $\sigma = 0.5, c = 0.01$	1185	150
Robust Soliton, $\sigma = 0.5, c = 0.03$	1128	65
Robust Soliton, $\sigma = 0.5, c = 0.1$	1177	36
Robust Soliton, $\sigma = 0.9, c = 0.04$	1124	57

least more probability mass at the tail distribution) is needed for a reasonable standard deviation.

The choice of the right parameter values with Robust Soliton distribution seems to be very important and a bad choice leads to poor performance. We did not found parameter combinations which would outperform the form (7.3) with optimized parameter values $\boldsymbol{\eta}_{\text{opt}} = (0.083\ 0.487\ 0.032)$ for neither $n = 100$ nor $n = 1000$. If still better distributions exist, that remains an open question and area for further work.

Chapter 8

Conclusions

This chapter concludes this work, with some discussion of the generated results and suggestions for further research topics.

8.1 Erasure Correcting Methods for Data Transmission

We have presented different existing methods for employing erasure correction on data transmission in end-to-end fashion. The use of erasure correction instead of traditional retransmission based techniques is certainly an interesting application, and quite good efficient methods for this task are available. Both the codes under the digital fountain concept and the LDPC codes have presented good performance in different references. We have discussed some example applications where these codes show their strengths and also considered some issues which arise when employing software erasure correction. We noted that while the different software FEC methods definitely have their uses, an implementation utilizing these methods should be carefully considered to avoid poor performance and conflicts with other possible protocols in the same network.

Our own contribution for this field includes the ISG-algorithm, which was derived using mathematics borrowed from the importance sampling theory. Our algorithm falls into the practical side of LT coding research, by providing a tool which can be used to test out and optimize different degree distributions for finite-length LT codes. We deliberately left out the graph theoretic considerations, and derived the algorithm from an experimental starting point.

8.2 Optimization Results

The optimization results provided in the previous chapter show us that the ISG-algorithm and the importance sampling based method works for the optimization of degree distributions for finite-length LT codes.

The point distribution optimization performs well for the low range of $n = 3, \ldots, 10$, after this, numerical difficulties arise and the optimization is not feasible anymore because of the large number of free parameters. The optimized point distributions in this range do not show any spike, which is one characteristic of the Robust Soliton distri-

bution (Figure 3.6). Also, the probability of degree one symbols is greater than the probability of degree two symbols, which is the case neither with the Soliton nor the Robust Soliton distribution.

For parameterized forms 7.1 and 7.2 we presented results for $n = 10, \ldots, 100$. The packet overhead percentages drop from over 40% to around 37% with this form as n increases from 10 to 100.

The best forms we found for $n = 100$ were derived from Ideal Soliton distribution with two first probabilities parameterized. A spike was also introduced and this way the overhead dropped to around 24%. We also noticed that the optimized distribution works better than the Robust Soliton distribution when $n = 1000$, the packet overhead was around 12%. It is probably possible to generate even better distributions by considering more sophisticated forms with more parameters.

In our simulations, the maximum number of samples generated in one iteration was limited to one or two million samples. Increasing this limit would result in more accurate distributions, but even more important goal is to make the calculations efficient for a large n, as the optimization algorithm running times take a great hit for $n > 100$.

The questions of what is the best form of degree distribution and if the spike used in Robust Soliton distribution and in our example distributions is really needed, remain unanswered. The distributions with the spike did produce the best results, but if a degree distribution with heavier tail is sufficient was not tested.

8.3 Further Research Topics

The area of erasure correcting coding includes many topics which are not answered yet. More practical results of using different coding schemes and larger scale implications of the usage of erasure codes needs to be answered. This would require the research and comparison of the operation of different erasure correcting schemes in real or simulated networks with large amount of participants. One issue is also to find out how different transmission schemes interoperate for example in the Internet.

Efficient degree distributions for LT coding in full range from low n to tens of thousands have not been published, and better results could, without a doubt, be generated. Our optimization method performs quite well at least for $n < 100$, but more could be done. The problem is the large number of samples needed for efficient and accurate optimization with the importance sampling based method, resulting in long simulation times.

A more efficient implementation of the proposed algorithm would probably help, perhaps implementing parts of the calculations in a lower level programming language. The natural continuation of this work would be optimization of the algorithm itself, allowing operation for larger values of n with more samples.

We expect that the basic idea of utilizing importance ratios in the estimate for average number of packets could be further employed in different optimization tasks, one specific application related to this work would be the optimization of graphs used in the LDPC codes. The field of new data transmission schemes is certainly interesting and important in the future. Considering the large number of positive research results found in this field during the last few years, better methods are probably just waiting for to be found.

Bibliography

[1] CSC, the Finnish IT center for science. 7.1

[2] Digital Fountain Inc. *http://www.digitalfountain.com*. 3.3

[3] Information additive group code generator and decoder for communication systems. U.S. Patent #6,320,520. Issued November 20,2001. 5.6

[4] The oceanstore project. *http://oceanstore.cs.berkeley.edu/, accessed 10.3.2006*. 4.3

[5] 3GPP TS 26.346 v6.3.0, 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, Multimedia Broadcast/Multicast Service (MBMS), Protocols and codecs(release 6). Technical report, December 2005. http://www.3gpp.org/ftp/Specs/html-info/26346.htm accessed 5.3.2006. 5.6

[6] B. Adamson, C. Bormann, M. Handley, and J. Macker. Negative-acknowledgment (NACK)-Oriented Reliable Multicast (NORM) Protocol. RFC 3940, IETF, November 2004. http://www.ietf.org/rfc/rfc3940, accessed 5.3.2006. 1

[7] Mokhtar S. Bazaraa, Hanif D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley and Sons, Inc., 2nd edition, 1993. 6.3.3

[8] J. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications, 20(8)*, October 2002. 4.1

[9] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998. 1.1, 3.1.3, 3.3.1, 3.3.2, 3.3.3

[10] Bruce A. Carlson, Paul B. Crilly, and Janet C. Rutledge. *Communication Systems: An Introduction to Signals and Noise in Electrical Communication*. McGraw-Hill, 4th edition, 2002. 2.1

[11] R. L. Collins and J. S. Plank. Assessing the performance of erasure codes in the wide-area. In *DSN-05: International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005. IEEE. 3.1.3

[12] A. Harmin et. al. Generating high weight encoding symbols using a basis. U.S. Patent #6,411,223. Issued June 25, 2002. 5.6

[13] M. Luby et. al. On demand encoding with a window. U.S. Patent #6,486,803. Issued November 26, 2002. 5.6

[14] Robert G. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, Massachusetts Institute of Technology, 1960. 3.2

[15] Robert G. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, 8(1):21–28, January 1962. 3.2

[16] T. Hardjono and B. Weis. The Multicast Group Security Architecture. RFC 3740, IETF, March 2004. http://www.ietf.org/rfc/rfc3740, accessed 5.3.2006. 5.4

[17] Wolfram Research Inc. Mathematica 5.2. http://www.wolfram.com. 7.1

[18] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988. 7.1

[19] Jin Li. The efficient implementation of Reed-Solomon high rate erasure resilient codes. In *2005 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1097–1100, 2005. 3.1.3

[20] M. Luby. Information additive code generator and decoder for communications systems. U.S. Patent #6,307,487. Issued October 23, 2001. 5.6

[21] M. Luby. Information additive code generator and decoder for communications systems. U.S. Patent #6,373,406. Issued April 16, 2002. 5.6

[22] M. Luby. Information additive code generator and decoder for communications systems. U.S. Patent #6,614,366. Issued September 2, 2003. 5.6

[23] M. Luby, J. Gemmell, L. Viciano, L. Rizzo, and J. Crowcroft. Asynchronous Layered Coding (ALC) Protocol Instantiation. RFC 3450, IETF, December 2002. http://www.ietf.org/rfc/rfc3450, accessed 5.3.2006. 2

[24] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer. Raptor Forward Error Correction Scheme for Object Delivery. Internet-draft, IETF, October 2005. http://www.ietf.org/internet-drafts/draft-ietf-rmt-bb-fec-raptor-object-03.txt, accessed 12.3.2006. 4.1.1, 5.6

[25] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. Forward Error Correction (FEC) Building Block. RFC 3452, Internet Engineering Task Force, December 2002. http://www.ietf.org/rfc/rfc3452.txt, accessed 3.2.2006. 4.1.1

[26] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. The Use of Forward Error Correction (FEC) in Reliable Multicast. RFC 3452, Internet Engineering Task Force, December 2002. http://www.ietf.org/rfc/rfc3453.txt, accessed 2.3.2006. 1.1, 4.1.1

[27] Michael Luby. LT Codes. In *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002. 1.2, 3.3.3, 3.3.3, 3.3.3, 3.3.3

[28] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Improved low-density parity-check codes using irregular graphs. *IEEE Transactions on Information Theory*, 47(2):585–598, February 2001. 3.3.1

[29] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *Proceedings of 29th Symposium on Theory of Computing*, pages 150–159, 1997. 3.3.1, 3.3.2

[30] David J. C. Mackay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. 2, 2.3, 2.6, 2.6.1, 3.2.2, 3.3.1

[31] J.S. Milton and Jesse C. Arnold. *Introduction to Probability and Statistics*. McGraw Hill Series in Probability and Statistics. McGraw-Hill Publishing Company, 2nd. edition, 1990. 6.3.2

[32] Michael Mitzenmacher. Digital fountains: A survey and look forward. 5.6

[33] Robert H. Morelos-Zaragoza. *The Art of Error Correcting Coding*. John Wiley & Sons., 2002. 2, 3.1.1, 3.1.1, 3.1.2

[34] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh. FLUTE - File Delivery over Unidirectional Transport. RFC 3926, IETF, October 2004. http://www.ietf.org/rfc/rfc3926, accessed 3.2.2006. 4.1.1

[35] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997. 1.1, 3.1.2

[36] J. S. Plank, A. L. Buchsbaum, R. L. Collins, and M. G. Thomason. Small parity-check erasure codes - exploration and observations. In *DSN-05: International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005. IEEE. 3.2.3, 5.2

[37] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on reed-solomon coding. Technical Report CS-03-504, University of Tennessee, April 2003. 3.1.2

[38] J. S. Plank and M. G. Thomason. On the practical use of LDPC erasure codes for distributed storage applications. Technical Report CS-03-510, University of Tennessee, September 2003. 1.1, 3.2, 4.3, 5.6

[39] Irvin Reed and Gustave Solomon. Polynomial codes over certain finite fields. *SIAM Journal of Applied Mathematics*, 8(2):300–304, 1960. 3.1, 3.1.1, 3.1.2

[40] L. Rizzo. On the feasibility of software FEC. Technical report, Università de Pisa, 1997. Available at http://www.iet.unipi.it/~luigi/softfec.ps, accessed 8.3.2006. 2.4.1

[41] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, April 1997. 1.1, 3.1.3

[42] Reuven Y. Rubinstein and Benjamin Melamed. *Modern Simulation and Modeling*. Wiley Series in Probability and Statistics. John Wiley & Sons Inc., 1998. 6.1

[43] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948. 2.6

[44] Amin Shokrollahi. Raptor codes. Preprint at http://www.inference.phy.cam.ac.uk/mackay/dfountain/RaptorPaper.pdf, cited 1.3.2005. 3.3.4, 5.1

[45] Tuomas Tirronen. Optimal degree distribution for LT codes in small cases. Technical report, Helsinki University of Technology, 2005. 6.3.1, 7.2, 7.1, 7.2

[46] B. Whetten, L. Vicisano, R. Kermode, M. Handley, S. Floyd, and M. Luby. Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer. RFC 3048, IETF, January 2001. http://www.ietf.org/rfc/rfc3048.txt, accessed 2.3.2006. 4.1.1

[47] Stephen B Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice Hall, 1995. 2, 2.4.2, 3.1.1, 3.1.2, A

Appendix A

Finite fields

This appendix explains briefly parts of Galois field theory. More extensive treatment can be found in numerous algebra and coding theory books, the following material is largely adapted from [47].

Definition A.1. *Field $\mathbb{F}$ is a set which satisfies the following field axioms for all $a, b, c \in \mathbb{F}$:*

	Addition	***Multiplication***
Commutativity:	$a + b = b + a$	$a \cdot b = b \cdot a$
Associativity:	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Distributivity:	$a \cdot (b + c) = a \cdot b + a \cdot c$	$(b + c) \cdot a = b \cdot a + c \cdot a$
Identity element:	$\exists!\ e$ *s.th.* $a + e = a$	$\exists!\ z$ *s.th.* $a \cdot z = a$
Inverse element:	$\exists!\ -a$ *s.th.* $a + (-a) = e$	$\exists!\ a^{-1}$ *s.th.* $a \cdot a^{-1} = z\ \forall\ a \neq e$

Examples of familiar fields include rational numbers $\mathbb{Q}$ and real numbers $\mathbb{R}$. Integers $\mathbb{Z}$ on the other hand does not form a field, the only element which has inverse element belonging to $\mathbb{Z}$ is 1.

Definition A.2. *A Galois field (or finite field) is a field with finite number of elements. A Galois field with q elements is denoted $GF(q)$. Galois fields are always unique for a q and exists for all $q = p^m$, where p is a prime number and m a positive integer.*

Galois field $GF(2)$ includes elements $\{0, 1\}$ and addition and multiplication operations are defined in Table A.1.

Table A.1: Addition and multiplication tables in $GF(2)$

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

Similar tables are easy to construct for every Galois field where number of elements is a prime. When $q = p^m$ is a power of prime, then elements of Galois field can be represented as polynomials whose coefficients belong to $GF(p)$.

Definition A.3. *All Galois fields have a primitive element α such that all other elements can be expressed as consecutive powers f primitive element: $\beta = \alpha^i, 0 \leq i \leq q - 2$.*

Definition A.4. *An irreducible polynomial $p(x)$ of order m, whose coefficients belong to $GF(q)$ is a primitive polynomial if the smallest integer n for which $p(x)$ divides $x^n - 1$ is $n = p^m - 1$. Primitive element α is a root of primitive polynomial $p(x)$.*

The roots of primitive polynomial can be used to form a polynomial representation for elements of $GF(p^m)$. For example $p(x) = x^3 + x + 1$ is primitive in $GF(2)$. Elements of $GF(2^3) = GF(8)$ can be represented using this fact. If α is root of $p(x)$, that is $p(\alpha) = \alpha^3 + \alpha + 1 = 0$, then $\alpha^3 = \alpha + 1$. This result can be used to form a polynomial representation for elements of $GF(8)$. For example fourth power of primitive element α, $\alpha^4 = \alpha \cdot \alpha^3 = \alpha \cdot (\alpha + 1) = \alpha^2 + \alpha$. Also a binary vector representation can be formed from polynomial representation by using a basis $\{\alpha^2, \alpha, 1\}$, in fact, every $GF(2^m)$ is isomorphic to linear vector space $\{0, 1\}^m$. Table A.2 lists different representations of elements of $GF(8)$. Similar tables can be constructed for any $GF(p^m)$, where p is prime and m a positive integer.

Table A.2: Elements of $GF(8)$

Element	Polynomial	Vector
0	0	000
1	1	001
α	α	010
α^2	α^2	100
α^3	$1 + \alpha$	011
α^4	$\alpha + \alpha^2$	110
α^5	$1 + \alpha + \alpha^2$	111
α^6	$1 + \alpha^2$	101

Table A.2 can be used to construct multiplication and addition tables of $GF(8)$. For example addition $\alpha^5 + \alpha^4 = (1 + \alpha + \alpha^2) + (\alpha + \alpha^2) = 1$, by using polynomial representations of α^4 and α^5 and coefficients from $GF(2)$.

Appendix B

Proof of Theorem 3.3

Proof. We use proof by induction for $i \in \{2, \ldots, N-1\}$:

1. Basis of induction: with $i = 2$ the result follows from (3.9), i.e.,

$$n_2^{(t)} = \frac{N-t}{2}.$$

2. Induction step: our induction hypothesis is that Theorem 3.3 is true for k, i.e.,

$$n_k^{(t)} = \frac{N-t}{k \cdot (k-1)}.$$

Now we prove the case for $k+1$ using (3.12):

$$\begin{aligned} n_{k+1}^{(t)} &= \frac{N-t}{k+1}\left(n_k^{(t+1)} - n_k^{(t)}\right) + \frac{i}{i+1} n_k^{(t)} \\ \text{by induction hypothesis} &= \frac{N-t}{k+1}\left(\frac{N-t-1}{k(k-1)} - \frac{N-t}{k(k-1)}\right) + \frac{k}{k+1}\frac{N-t}{k(k-1)} \\ &= -\frac{N-t}{(k+1)k(k-1)} + \frac{k(N-t)}{(k+1)k(k-1)} \\ &= \frac{(k-1)(N-t)}{(k+1)\ k\ (k-1)} = \frac{N-t}{k(k+1)}, \end{aligned}$$

which is the same as the statement of Theorem 3.3.

By the induction property of natural numbers, Theorem 3.3 is true for all $i \in \{2, \ldots, N-1\}$.

□