Investigation on Digital Fountain Codes over Erasure Channels and Additive White Gaussian Noise Channels

A dissertation presented to
the faculty of
the Russ College of Engineering and Technology of Ohio University

In partial fulfillment
of the requirements for the degree
Doctor of Philosophy

Weizheng Huang
June 2012

© 2012 Weizheng Huang. All Rights Reserved.
This dissertation titled
Investigation on Digital Fountain Codes over Erasure Channels and Additive White Gaussian Noise Channels

by
WEIZHENG HUANG

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Jeffrey Dill
Professor of Electrical Engineering and Computer Science

Dennis Irwin
Dean, Russ College of Engineering and Technology
ABSTRACT

HUANG, WEIZHENG, Ph.D., June 2012, Electrical Engineering

Investigation on Digital Fountain Codes over Erasure Channels and Additive White Gaussian Noise Channels

Director of Dissertation: Jeffrey Dill

As newly invented packet erasure codes, digital fountain codes (LT codes and Raptor codes) under iterative message passing decoding can work very efficiently in computer networks for large scale data distribution (e.g., greater than $6.4 \times 10^4$ bits) without knowledge of the states of individual lossy channels, regardless of the propagation modes. Some researchers have moved further and found that fountain codes can achieve near capacity performance over AWGN channels. However, little literature on the research of a fountain code’s decoding overhead had been obtained, especially for short and moderate-length data (e.g., smaller than $1 \times 10^4$ bits). We are interested in the overheads of fountain codes of different kinds or designs because a harsh communication condition can hurt some decoding schemes by limiting the number of received encoded symbols. Moreover, we have only found in literature studies of Raptor codes whose pre-codes are rate 0.98 left-regular, right-Poisson LDPC codes, but performance with other types of pre-codes is unknown. In this dissertation, we review conventional fountain codes and describe two system models for packet erasure fountain codes and bit error correcting fountain codes under message passing decoding over binary erasure channels or AWGN channels. We assess Raptor codes with different kinds of pre-codes, introduce maximum likelihood decoding to both LT codes and Raptor codes, and propose a hybrid
message passing and fast maximum likelihood decoding algorithm for LT codes. Our simulation results show that error rate and overhead depends on both decoding algorithm and pre-code characteristics. In general, maximum likelihood decoding consumes much less overhead than message passing decoding. Our hybrid algorithm can realize very low overhead with short-length LT codes but still enable fast decoding. LT codes can decrease the fraction of overhead as data length grows but Raptor codes may not. A higher rate pre-code can accomplish better performance of Raptor codes than a lower rate pre-code, and the structure of the pre-code may not matter.

Approved: _____________________________________________________________

Jeffrey Dill

Professor of Electrical Engineering and Computer Science
To Mom, Dad, Wendy, John, Weiwen, Kent and Victor
ACKNOWLEDGMENTS

It has been a long way to come to the end of my Ph.D. program. I would like to take this opportunity to express my gratitude to those professional organizations and persons that have supported, in any ways, my doctoral studies and research.

First of all, I need to thank Ohio University and The School of Electrical Engineering and Computer Science. Six years ago, I received an admission and offer packet from EECS, which led my life from the United Kingdom to the United States so that I stepped onto a higher and broader stage of learning and engineering research.

I was not interested in communication theories at college so that some years later I had to start over at Ohio University to learn in this field. My deep appreciation to Dr. Jeffrey Dill and Dr. David Matolak is always in my heart. In the first week of my study in Athens, Dr. Dill opened a door for me to the area of communications and information. He soon became my academic advisor and later guided me to research in digital fountain codes. I admired his speaking and thinking. Dr. Matolak was my instructor in the classes of stochastic processes, mobile communications and CDMA systems. He has broadened my horizon of statistics, signals and communications, and provided me plenty of helpful guidance on learning, research and teaching.

I am also very grateful to my other committee members, Dr. Shawn Ostermann, Dr. Maarten Uijt de Haag and Dr. Savas Kaya, and my college representatives, Dr. Archil Gulisashvili and Dr. Vardges Melkonian. I will never forget they have taken great efforts for my dissertation proposal and dissertation.
Mr. Timothy Bambeck has been my supervisor of teaching assistantship for more than three years. Without his guidance, support and trust, I would not have been an employable graduate teaching assistant. With this job and the financial aid, I am able to finish my study and dissertation.

Here I want to mention ARCON Corporation which offered me cherished internship in 2010 and later 2011 – 2012. I thank Dr. Aditya Kamal, our chief engineer, who gave me this chance to learn and practice in research in industries. My manager Siva Sivananathan and group leaders took great efforts to training me object oriented programming. Dr. Akhilesh Shrestha taught me a lot of basic knowledge in air surveillance and professional MATLAB simulation.

I will always remember my life in Athens, Ohio. I love this beautiful, quiet and peaceful community which is a very good place for me to study and research. At least, I like to thank my lab partners: Dr. Huanlin Li, Kai-Jen Cheng, Dr. Jingtao Zhang, Qiong Wu, Qian Zhang, Kamal Ganti and Irina Anitei.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>3</td>
</tr>
<tr>
<td>Dedication</td>
<td>5</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>6</td>
</tr>
<tr>
<td>List of Tables</td>
<td>11</td>
</tr>
<tr>
<td>List of Figures</td>
<td>12</td>
</tr>
<tr>
<td>List of Abbreviations</td>
<td>15</td>
</tr>
<tr>
<td>Chapter 1: Introduction</td>
<td>17</td>
</tr>
<tr>
<td>1.1 Decoding Error Rate</td>
<td>18</td>
</tr>
<tr>
<td>1.2 Decoding Complexity</td>
<td>19</td>
</tr>
<tr>
<td>1.3 Overview of Dissertation</td>
<td>19</td>
</tr>
<tr>
<td>1.4 Dissertation Contributions</td>
<td>20</td>
</tr>
<tr>
<td>Chapter 2: Preliminaries and Backgrounds</td>
<td>22</td>
</tr>
<tr>
<td>2.1 Erasure Channels</td>
<td>23</td>
</tr>
<tr>
<td>2.2 Additive White Gaussian Noise Channel</td>
<td>24</td>
</tr>
<tr>
<td>2.3 Rayleigh Fading</td>
<td>26</td>
</tr>
<tr>
<td>2.4 Channel Capacity and Coding Bounds</td>
<td>27</td>
</tr>
<tr>
<td>2.5 Decoding Graph</td>
<td>30</td>
</tr>
<tr>
<td>2.6 Set Diagram</td>
<td>35</td>
</tr>
<tr>
<td>2.7 Multicast and Broadcast</td>
<td>38</td>
</tr>
<tr>
<td>2.8 Three Early Remarkable Erasure Codes</td>
<td>39</td>
</tr>
<tr>
<td>2.9 Summary</td>
<td>41</td>
</tr>
<tr>
<td>Chapter 3: Conventional Fountain Codes</td>
<td>43</td>
</tr>
<tr>
<td>3.1 Luby Transform Codes</td>
<td>44</td>
</tr>
<tr>
<td>3.1.1 Encoding</td>
<td>45</td>
</tr>
<tr>
<td>3.1.2 Decoding with Message Passing Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>3.1.3 Coding Complexity</td>
<td>54</td>
</tr>
<tr>
<td>3.2 Raptor Codes</td>
<td>55</td>
</tr>
<tr>
<td>3.2.1 Encoding</td>
<td>55</td>
</tr>
<tr>
<td>3.2.2 Message Passing Decoding</td>
<td>58</td>
</tr>
<tr>
<td>3.2.3 Coding Complexity</td>
<td>59</td>
</tr>
</tbody>
</table>
3.3 Summary ........................................................................................................... 60

Chapter 4: Two System Models of Fountain Codes with MP Decoding ....................... 61
4.1 System Model of Fountain Codes for Packet Erasure Correction ......................... 62
4.2 System Model of Fountain Codes for Bit Error Correction ................................. 64
4.3 Details of Raptor Encoder .................................................................................. 67
    4.3.1 Straightforward Encoder ............................................................................ 67
    4.3.2 Fast Encoder .............................................................................................. 69
4.4 Details of Raptor Decoder .................................................................................. 73
    4.4.1 Decoding Information ............................................................................... 74
    4.4.2 Local-Iteration Decoder ............................................................................. 76
    4.4.3 Global-Iteration Decoder ........................................................................... 80
    4.4.4 Reliable Bit Selection by the Receiver Pattern ........................................... 82
    4.4.5 Relationship between Hard Decoding and Soft Decoding ......................... 84
4.5 Summary ........................................................................................................... 88

Chapter 5: Studies of LDPC Codes ................................................................................ 90
5.1 Stopping Sets with LDPC Codes of Finite Lengths............................................. 91
5.2 A New Approach to Searching the Smallest Stopping Sets in LDPC Codes........ 93
    5.2.1 Assumptions and Definitions ..................................................................... 93
    5.2.2 The Searching Algorithm .......................................................................... 96
    5.2.3 Complexity and Space ............................................................................. 101
5.3 A Splitting-and-filling Technique to Constructing Good Irregular LDPC Codes .......... 103
    5.3.1 Background ............................................................................................. 103
    5.3.2 The Splitting-and-Filling Technique on Set Diagrams ............................. 105
    5.3.3 Construction of Irregular LDPC Codes with Good Combinatorial Properties ........................................................................................................ 106
5.4 Performances of Our Left-regular LDPC Codes under MP decoding over BEC and AWGNC ............................................................. 111
5.5 LDPC Codes in the IEEE 802.16e Standard ..................................................... 116
    5.5.1 Background ............................................................................................. 116
    5.5.2 Parity-Check Matrix Construction ........................................................... 117
    5.5.3 Fast Encoding .......................................................................................... 119
    5.5.4 Error Performance under Iterative decoding over BEC and AWGNC ...... 119
5.6 Summary ......................................................................................................... 123

Chapter 6: Maximum Likelihood Decoding with Erasures ........................................... 125
6.1 Traditional Method .......................................................................................... 126
LIST OF TABLES

Table 1: LT Code’s Average Recovery Ratio and Standard Deviation ....................... 140
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An $M$-ary erasure channel model</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>Probability density function of received bit values with AWGN</td>
<td>26</td>
</tr>
<tr>
<td>2.3</td>
<td>Bit error probability for BPSK</td>
<td>29</td>
</tr>
<tr>
<td>2.4</td>
<td>A Tanner graph</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Tanner graph of generator matrix $M$</td>
<td>32</td>
</tr>
<tr>
<td>2.6</td>
<td>Another Tanner graph of generator matrix $M$</td>
<td>33</td>
</tr>
<tr>
<td>2.7</td>
<td>Tanner graph of parity-check matrix $M$</td>
<td>34</td>
</tr>
<tr>
<td>2.8</td>
<td>A cycle of size 6 in the Tanner graph of parity-check matrix $M$</td>
<td>35</td>
</tr>
<tr>
<td>2.9</td>
<td>Set diagrams for parity-check matrix $H$</td>
<td>36</td>
</tr>
<tr>
<td>2.10</td>
<td>Set diagrams for parity-check matrix $H_1$</td>
<td>37</td>
</tr>
<tr>
<td>2.11</td>
<td>Examples of multicast and broadcast</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>RSD for dimension $10^4$ with $c = 0.1$ and $\delta = 0.05$</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>Avg. degree of LT encoded symbols with $n = \beta k$, $c = 0.1$ and $\delta = 0.05$</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td>$\beta$ for LT codes with $c = 0.1$ and $\delta = 0.05$</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>An example of parallel hard MP decoding of an LT code</td>
<td>52</td>
</tr>
<tr>
<td>3.5</td>
<td>An example of unsuccessful parallel hard MP decoding of an LT code</td>
<td>53</td>
</tr>
<tr>
<td>3.6</td>
<td>Raptor encoding diagram</td>
<td>56</td>
</tr>
<tr>
<td>3.7</td>
<td>Soliton distribution of a weakened LT code</td>
<td>57</td>
</tr>
<tr>
<td>4.1</td>
<td>System model of packet erasure fountain code with MP decoding</td>
<td>62</td>
</tr>
<tr>
<td>4.2</td>
<td>System model of bit correcting fountain codes with MP decoding</td>
<td>65</td>
</tr>
</tbody>
</table>
Figure 4.3: Straightforward encoding of Raptor codes ................................................ 68
Figure 4.4: Fast encoding of Raptor codes .................................................................. 70
Figure 4.5: H matrix in approximate lower-triangular form for fast encoding .......... 71
Figure 4.6: Raptor local-iteration decoder configuration ............................................. 76
Figure 4.7: Raptor global-iteration decoder configuration ........................................... 81
Figure 4.8: Selection threshold on received bits from AWGNC .................................. 83
Figure 4.9: Hard decoding and soft decoding of an LT code ....................................... 87
Figure 5.1: A stopping set of size 3 in a parity-check matrix ....................................... 92
Figure 5.2: Set Diagrams for parity-check matrix H .................................................... 95
Figure 5.3: Diagram of searching stopping sets ......................................................... 100
Figure 5.4: Partial splitting-and-filling on a variable node set ................................... 106
Figure 5.5: BERs of left-regular LDPC codes under MP decoding over BEC .......... 113
Figure 5.6: FERs of left-regular LDPC codes under MP decoding over BEC .......... 114
Figure 5.7: BERs of left-regular LDPC codes under MP decoding over AWGNC .... 115
Figure 5.8: FERs of left-regular LDPC codes under MP decoding over AWGNC .... 116
Figure 5.9: BERs of 802.16e LDPC codes under MP decoding over BEC .......... 120
Figure 5.10: FERs of 802.16e LDPC codes under MP decoding over BEC .......... 121
Figure 5.11: BERs of 802.16e LDPC codes under MP decoding over AWGNC ...... 122
Figure 5.12: FERs of 802.16e LDPC codes under MP decoding over AWGNC ...... 123
Figure 6.1: Configuration of the submatrix V of B .................................................... 129
Figure 6.2: Encoding matrix A of a nonsystematic Raptor code ................................ 134
Figure 7.1: LT Code (k =1000) with MP Decoding over BEC ..................................... 140
Figure 7.2: LT code \((k = 1000)\) with hybrid MP/fast ML decoding over BEC .......... 141

Figure 7.3: BERs of Raptor codes with local-iteration MP decoding over BEC ........ 142

Figure 7.4: FERs of Raptor codes with local-iteration MP decoding over BEC ....... 143

Figure 7.5: BERs of Raptor codes with global-iteration MP decoding over BEC ...... 144

Figure 7.6: FERs of Raptor codes with global-iteration MP decoding over BEC ...... 145

Figure 7.7: BERs of Raptor codes with ML decoding over BEC .......................... 146

Figure 7.8: FERs of Raptor codes with ML decoding over BEC .......................... 147

Figure 7.9: Average overheads of fountain codes ........................................... 149

Figure 7.10: BERs of Raptor codes with L-iteration MP decoding over AWGNC ..... 150

Figure 7.11: FERs of Raptor codes with L-iteration MP decoding over AWGNC ..... 151

Figure 7.12: BERs of Raptor codes with G-iteration MP decoding over AWGNC .... 152

Figure 7.13: FERs of Raptor codes with G-iteration MP decoding over AWGNC .... 153

Figure 7.14: ERs of Raptor codes with selection threshold 0.6 over AWGNC ........ 154

Figure 7.15: Error rates of Raptor codes over AWGNC with Rayleigh fading ..... 155
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>The 3rd Generation Partnership Project</td>
</tr>
<tr>
<td>ARQ</td>
<td>Automatic repeat request</td>
</tr>
<tr>
<td>AWGN</td>
<td>Additive white Gaussian noise</td>
</tr>
<tr>
<td>AWGNC</td>
<td>Additive white Gaussian noise channel</td>
</tr>
<tr>
<td>BEC</td>
<td>Binary erasure channel</td>
</tr>
<tr>
<td>BER</td>
<td>Bit error rate</td>
</tr>
<tr>
<td>BP</td>
<td>Belief propagation</td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary phase shift keying</td>
</tr>
<tr>
<td>c-node</td>
<td>Check node</td>
</tr>
<tr>
<td>DFR</td>
<td>Decoding failure rate</td>
</tr>
<tr>
<td>FEC</td>
<td>Forward error correction</td>
</tr>
<tr>
<td>FER</td>
<td>Frame error rate</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IPTV</td>
<td>Internet Protocol television</td>
</tr>
<tr>
<td>i.i.d.</td>
<td>independent and identically distributed</td>
</tr>
<tr>
<td>LAN</td>
<td>Local area network</td>
</tr>
<tr>
<td>LDPC</td>
<td>Low-density parity-check</td>
</tr>
<tr>
<td>LLR</td>
<td>Log-likelihood ratio</td>
</tr>
<tr>
<td>LR</td>
<td>Left-regular</td>
</tr>
<tr>
<td>LT</td>
<td>Luby transform</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MBMS</td>
<td>Multimedia Broadcast/Multicast Services</td>
</tr>
<tr>
<td>ML</td>
<td>Maximum likelihood</td>
</tr>
<tr>
<td>MP</td>
<td>Message passing</td>
</tr>
<tr>
<td>PCO</td>
<td>Pre-code only</td>
</tr>
<tr>
<td>pdf</td>
<td>Probability density function</td>
</tr>
<tr>
<td>pmf</td>
<td>Probability mass function</td>
</tr>
<tr>
<td>psd</td>
<td>Power spectral density</td>
</tr>
<tr>
<td>REF</td>
<td>Row echelon form</td>
</tr>
<tr>
<td>RP</td>
<td>Right-Poisson</td>
</tr>
<tr>
<td>RREF</td>
<td>Reduced row echelon form</td>
</tr>
<tr>
<td>RS</td>
<td>Reed-Solomon</td>
</tr>
<tr>
<td>RSD</td>
<td>Robust Soliton distribution</td>
</tr>
<tr>
<td>SF</td>
<td>Splitting-and-filling</td>
</tr>
<tr>
<td>SNR</td>
<td>Signal-to-noise ratio</td>
</tr>
<tr>
<td>s-node</td>
<td>Source node</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of service</td>
</tr>
<tr>
<td>v-node</td>
<td>Variable node</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive or</td>
</tr>
</tbody>
</table>
CHAPTER 1: INTRODUCTION

This dissertation deals with studies of digital fountain codes over some noisy channels: erasure channels, additive white Gaussian noise channels, and Rayleigh fading channels. As new rateless channel codes, fountain codes have been successful as universal erasure correcting codes for broadcast or multicast data. Since multimedia communications and computer networks are getting more and more popular and functional in modern life, we are interested in studying the fundamental properties and behaviors of fountain codes. This dissertation includes our contributions in improving two fountain codes system models for packet recovery and bit error correction, a study of pre-codes of Raptor codes, and proposing a hybrid fast decoding algorithm to reduce the overheads of short and moderate-length LT codes.

We have known that all digital signals can be corrupted by noise introduced by communication channels, no matter wired or wireless, long or short distance. At receivers, the corruption may result in symbol or data frame errors. The errors, of course, impair the quality of service. Sometimes, one or more received and corrupted data frames are discarded if any detected errors included in the frames cannot be corrected. In the past few decades, many channel codes, or forward error correction (FEC) codes, have been invented to help transmitted data restore themselves at receivers. A question with broadcast/multicast (point-to-multipoint) environment came to researchers. Can a single code work with multiple and/or unpredictable data delivery channels without sacrificing bandwidth or power efficiency and without feedback and retransmission? A fountain code can be the solution.
The conventional fountain codes work under important assumptions. All encoded packets accepted by a fountain decoder are assumed to be correct, because the system can use an FEC scheme to correct the detected errors within a packet otherwise the packet will be discarded. The decoder constructs a generator matrix associated with these accepted symbols. The matrix information of each symbol can be delivered to the decoder correctly in many ways. We assume that decoders always know or establish decoding matrices without any errors. Moreover, memory or storage space is assumed sufficient for any scenarios in this work.

The name “fountain” illustrates the magic of fountain codes. With the above assumptions, the code can restore all source data from a sufficient number of encoded symbols randomly collected from a fountain of independent and identical symbols. We see that this “magic” can happen if and only if fountain codes are packet erasure codes. In fact, a fountain code can be a normal linear block code that corrects bit errors. In this case, fountain codes are usually fixed rate codes at a receiver. However, fountain codes can discard less reliable encoded symbols in order to improve decoding performance.

The decoding method or algorithm is very important to the code’s performance. Given a code, the result of a certain decoding method is also influenced by channel model. In this work, only probabilistic decoding methods are considered.

1.1 Decoding Error Rate

Error rate is a common measure of a code’s performance. An error rate can be a symbol error rate which is the fraction of wrong source symbols at the decoder output.
Bit error rate (BER) is the primary symbol error rate in this dissertation because we focus on binary systems only. We use the Monte Carlo simulation method to characterize the error rate of any code or system, given a channel model. Since a code usually encodes a frame or group of multiple symbols which would compose a piece of usable information, we also consider frame error rate (FER). This rate is the frequency that the code fails in restoring source information by frames.

### 1.2 Decoding Complexity

A code can be excellent or optimal in view of symbol or frame error rate but the computation complexity may be too high [58]. Too high complexity might not be tolerated in some applications, e.g., voice communication. Thus, complexity is another merit for a code in our research. Complexity is sometimes replaced by the term *cost*.

We need to designate how to measure complexity. Complexity can be the number of arithmetic operations. Besides conventional operations of addition, subtraction, multiplication and division, familiar operations include position interchange of entries (or entry sets) and the number of decoding iterations. In this work, the density of a matrix is the major measure of complexity because only binary codes are investigated and zero matrix entries are usually assumed not to contribute to complexity.

### 1.3 Overview of Dissertation

In this dissertation, we present some knowledge of digital fountain codes, investigate their effects over noisy channels, study designs of practical fountain codes,
and provide simulation results and analysis for these codes. In Chapter 2, we review some preliminaries and background information that should help the reader understand the fundamentals and applications of fountain codes. This includes some basic models of noisy channels, channel capacity, two representations of coding matrix, two point-to-multipoint data delivery modes, and three remarkable channel codes. Chapter 3 contains some concepts, construction and operations of original fountain codes. In Chapter 4, with many details, we introduce two fountain code system models under belief propagation (BP) decoding. In Chapter 5, we study low-density parity-check codes because the pre-code of a Raptor code is often such code. Chapter 6 describes maximum likelihood decoding algorithm for fountain codes over erasure channels. In Chapter 7, we present our simulation results and analysis of fountain codes. Finally, we make conclusions and suggest future work in Chapter 8.

1.4 Dissertation Contributions

In this section, we list the articles published or in review for the research on which this dissertation is based. Each of these articles is also followed by its citation number in the list of references at the end of this dissertation.

Published Articles


**Submitted Articles**

1. **W. Huang** and J. Dill, “A study of the pre-codes of Raptor codes,” submitted to *MILCOMM*, Nov. 2011, Baltimore, MD, USA. (Reference [61])
CHAPTER 2: PRELIMINARIES AND BACKGROUNDS

Digital communications have gained great development during the past decades. The Internet could be the most successful example of the benefits of sustained investment and commitment to research and development of information infrastructure in the past few decades. In computer network communications, requested data are usually divided into groups or blocks. Every one of these information units is encapsulated into a packet that is assigned a header including routing information and error-correction protocol information. These destination-specific packets are transmitted over the communication network, but some of them can be lost before arriving at the destination. The others reach the receiver and are treated with error detecting and correcting algorithms. If any detected errors cannot be corrected, the corresponding packet can be discarded. All lost or abandoned packets are viewed as erasures. Only the packets that are declared error free are accepted. Conventionally, some “make-up” schemes, like ARQ (automatic repeat request), offer best-effort data delivery service. However, the best-effort service can often no longer satisfy today’s real-time, broadcast or multicast applications with absolute end-to-end quality of service (QoS).

In this chapter, some fundamentals are reviewed because they are important and helpful for the reader to understand the application of digital fountain codes. First of all, the basic mathematical model of communication channel loss patterns are reviewed, since fountain codes were initially erasure correcting codes. Then, the additive white Gaussian noise channel, Rayleigh fading and channel capacity are discussed. Two methods of representing linear codes are introduced: Tanner graph, and set diagram. Next, two
common data delivery modes over computer networks, multicast and broadcast, are briefly described. Finally, we go over three families of notable erasure codes that were invented prior to fountain codes: Reed-Solomon codes, low-density parity-check codes, and Tornado codes.

2.1 Erasure Channels

Erasure channels are so important because many lossy or noisy communication channels can be modeled or simplified to approximate an erasure channel. Elias introduced the concept of erasure channels in 1955 [1]. A simple model of an $M$-ary memoryless erasure channel is shown in Figure 2.1, where $p$ is the erasure probability and $s_i$ and $r_i$ are the input and the output of the channel, respectively. The erasure is a special instance of the output if the symbol is lost or discarded. In this case, each input $s_i$ can be changed into an erasure with a probability $p$. Elias also proved that a rate $R$ code ($R < 1 - p$) could be constructed to transmit data over channels of capacity $1 - p$.

Figure 2.1. An $M$-ary erasure channel model.
The channel input and output $s_i$ and $r_i$ are all data symbols and $s_i = r_i$, i.e., the channel is error free in some sense. If $M = 2$, the channel is called a binary erasure channel (BEC). For data packet delivery services, each packet can be considered as a symbol since the packet payload is exactly a symbol which represents a sequence of data bits. In this dissertation, the two terms packet and symbol are equivalent unless specially stated. Symbols accepted by the receiver or decoder are assumed to be correct though, in reality, there is a small probability that undetected errors occur in an accepted packet. In this work, this possibility is neglected.

An erasure channel can be a bit erasure channel or a packet erasure channel, so a chosen erasure code should be on bit level or packet level accordingly. But many conditions on erasure channels share the same or similar theories and many problems in the Internet can be reduced to a BEC.

### 2.2 Additive White Gaussian Noise Channel

In this work, thermal noise of interest approximates a Gaussian random process. The noise sources on the communication channel can be man-made and natural types [49]. The term *additive white Gaussian noise* (AWGN) is often used in our research. The Gaussian noise channel we deal with is memoryless and the power spectral density (psd) of the noise is assumed to be the same for all frequencies. The psd of AWGN is given by

$$S_n(f) = \frac{N_0}{2},$$

(2.1)
where $N_0$ is the single-sided power spectral density. Therefore, the autocorrelation function of the white Gaussian noise is

$$R_{nn}(\tau) = \frac{N_0}{2} \delta(\tau), \quad (2.2)$$

where $\delta(\tau)$ is the Dirac delta function. Given a transmitted bit $x$ in antipodal form, the received bit can be expressed by

$$y = x + n, \quad (2.3)$$

where $n$ is an AWGN variable. The additive Gaussian noise is assumed to be zero-mean in this work, so the probability density function (pdf) of the received bit can be stated as a conditional pdf

$$f_Y(y|x) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{y-x}{\sigma}\right)^2\right], \quad (2.4)$$

where $\sigma^2 = N_0/2$ is the noise variance. Equation 2.4 is sufficient to describe an additive white Gaussian channel (AWGNC). Suppose $x = \pm 1$ and $\sigma = 0.5$ and the conditional pdfs of the received bit is illustrated in Figure 2.2. $E_b$ is the average received bit energy. At the receiver, $y = 0$ is the detection threshold.
2.3 Rayleigh Fading

Rayleigh fading is a type of frequency-non-selective (flat) multipath fading [50]. Flat fading is one of the common models in terrestrial cellular radio systems. It means that the multipath delay spread is much smaller than the transmitted symbol duration. In the wireless environment, multipath fading is caused by the addition of multiple arrivals of the wanted signal. The arrivals are different by amplitudes and/or phases so the addition can be advantageous or harmful. Multipath fading, sometimes called small scale fading, is nearly independent of propagation distance between antennas. It can be also somehow independent of the carrier frequency. Therefore, in this work, the multipath
fading treats all the frequency components of a signal equally (same magnitude gains) and the choice of frequency in a given band does not affect the nature of fading, unless otherwise stated.

In the Rayleigh condition, the inphase and quadrature components of the received band-pass signal are well-modeled as independent and identically distributed (i.i.d) zero-mean Gaussian random variables. The amplitude $A$ of the received complex envelope is a Rayleigh random process that obeys the distribution

$$f_A(a) = \frac{a}{b} \exp \left( -\frac{a^2}{2b} \right),$$

where $a \geq 0$ and $b$ is the variance of the inphase and quadrature components.

### 2.4 Channel Capacity and Coding Bounds

Shannon [7] proved that the capacity of an additive white Gaussian channel can be expressed as

$$C = W \log_2 \left( 1 + \frac{S}{N} \right),$$

where $W$ is the channel bandwidth, $S$ is the average received signal power and $N$ is the average Gaussian noise power. The relationship introduced in Equation 2.6 is also called Shannon-Hartley theorem or Shannon’s law. $S/N$ is named *average signal power to average noise power ratio*, often simply as *signal-to-noise ratio* (SNR), which “is a useful figure of merit for analog communications” [49]. For digital communications, however, $E_b/N_0$ is usually the merit of performance. $E_b$ is average bit energy and $N_0$ is the psd of the Gaussian noise. In fact, $E_b/N_0$ is a normalized version of SNR:
\[
\frac{E_b}{N_0} = \frac{S}{R_b} \frac{W R_b}{N},
\]

(2.7)

where \( R_b \) is transmission bit rate. For convenience without confusing, \( E_b/N_0 \) is sometimes called SNR.

Over an AWGN channel, with coherent detection of orthogonal signaling or phase signaling, bit errors would occur at the receiver sink. The bit error statistics are inherently dependent on the signaling and the channel. With a particular signaling scheme, the bit error probability is a function of \( E_b/N_0 \). In the error probability plane (BER vs. \( E_b/N_0 \)), the error probability is often depicted by a waterfall-like curve with lower error probability at higher \( E_b/N_0 \).

For binary data transmission, coherently detected *binary phase shift keying* (BPSK) is what we can do the best for BER performance without any error correcting coding. Reference [49] contains the derivation of the bit error probability of coherently detected BPSK. This probability is given by

\[
P_B = \int_{\sqrt{2E_b/N_0}}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{z^2}{2} \right) dz.
\]

(2.8)

Equation 2.8 is often stated as

\[
P_B = Q \left( \sqrt{\frac{2E_b}{N_0}} \right).
\]

(2.9)

The \( Q \) function is called the *complementary error function* or *co-error function*. The plot of Equation 2.8 is illustrated in Figure 2.3.
The value $E_b/N_0 = -1.59$ dB is the Shannon limit. Shannon showed that for any given noisy channel, it is possible to achieve error-free or nearly error-free digital information transfer at a maximum rate through the channel. The method was coding but Shannon did not tell how to make an error correction code. The Shannon limit is the bound that any code can never exceed. However, it is a theoretical limit for codes with infinite length and infinite processing time, which is absolutely impossible to create in practice. Although coherently detected $M$-ary orthogonal signaling can achieve better error performance than BPSK, it still requires $M = +\infty$ to reach the Shannon limit. Therefore, we focus on finite-length code designing for near-capacity performance.
As for a BEC with capacity $1 - p$, where $p$ is bit loss rate, an ideal code should achieve error-free communication at rate no more than $1 - p$. A very low rate code can work well with many loss patterns, but its power efficiency and bandwidth efficiency may not be satisfactory for a low loss rate. Therefore, a rateless code can fit various erasure channels.

### 2.5 Decoding Graph

Matrices could be the most classical and practical representations of linear codes, but graphs have been proved very effective to illustrate and explore the performance of propagation decoding with linear codes. Fountain codes, LDPC codes and turbo codes are all graph codes, since their decoding matrices are usually represented by Tanner graphs conveniently for the low density of the matrices. Because their matrices are usually low-density, these codes are also called sparse-graph codes. The sparseness of the decoding graph ensures low computation complexity.

Tanner [2] rediscovered LDPC codes by proposing Tanner graphs which apply recursive techniques to operate error or erasure correction. For example, a binary matrix is given by

$$M = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix}. \tag{2.10}$$

The Tanner graph of $M$ is illustrated in Figure 2.4.
Each row of the matrix $\mathbf{M}$ corresponds to one of the top row of circles in an ordinal manner and the columns of $\mathbf{M}$ are represented by the bottom row of squares in the same manner. The circles are labeled by $r_i$ where $i = 1, 2, 3, 4$; the squares are given by $c_j$ where $j = 1, 2, \ldots, 7$. Both the circles and the squares are called nodes or vertices. For all linear codes of interest, their Tanner graphs contain exactly two types of vertices with edges connecting the two banks, so these graphs are bipartite graphs. The edge between a pair of $r_i$ and $c_j$ denotes the binary 1 in the position $m_{i,j}$ in the matrix $\mathbf{M}$, and $r_i$ and $c_j$ are called a pair of neighbors. Thus, an edge is a symbol of the intersecting of a row and a column. The edges are the paths on which decoding messages travel. All edges are bidirectional message-passing ways between the two types of nodes. The total number of neighbors of a node is the degree of this node. Apparently, the degree equals the number of edges incident on the node. The Tanner graph in Figure 2.4 is sparse because the fraction of 1’s in $\mathbf{M}$ is less than 0.5 [3]. Equivalently, $\mathbf{M}$ is a low-density matrix. One advantage of sparse-graph codes is that the number of edges can be roughly linear with the codeword length.

Figure 2.4. A Tanner graph.
Generator and parity-check matrices can be represented by Tanner graphs conveniently. If \( \mathbf{M} \) in Equation 2.10 is a generator matrix with its rows corresponding to source symbols, its graph is shown in Figure 2.5 (note that it is the same graph as Figure 2.4 with nodes re-labeled). The circles in this Tanner graph are called source nodes (s-nodes), given by \( s_1, \ldots, s_4 \), and the squares are called variable nodes (v-nodes), given by \( v_1, \ldots, v_7 \). Every s-node connects to one single source symbol, or input symbol, and every v-node connects to one single codeword coordinate. The codeword coordinates are also called the output or encoded symbol. Each encoded symbol is the modulo-2 sum of all its neighbors.

![Tanner graph of generator matrix \( \mathbf{M} \).](image)

One may prefer another Tanner graph for the generator matrix, as illustrated in Figure 2.6. The squares with plus signs are check-nodes (c-nodes). They are connected to the v-nodes (codeword coordinates) \( v_1, \ldots, v_7 \) which are depicted by the small circles. Both the c-node and its corresponding v-node represent a same coordinate. The plus sign on the c-node means bitwise addition operation that is equivalent to exclusive or (XOR) operation. The value of \( v_i \) is still the bitwise sum of the neighboring s-nodes of \( c_i \).
modulo-2 sum of all incoming values on $c_i$ must 0. That is why $c_i$ is called a c-node here.

In this graph, the s-nodes can be considered a type of v-nodes.

For digital fountain codes, the generator matrix can be used to decode. At the received side, selected encoded symbols are used to establish a decoding Tanner graph. If the Tanner graph in Figure 2.5 is used, “received (encoded) symbol” and “v-node” are considered the same object. However, with the Tanner graph in Figure 2.6, the “v-node” becomes a “c-node” but the “received symbol” is not the “c-node”. The “received symbol” is connected to the “c-node” with observed information (or evidence) from the channel. In order to avoid confusion, for both the graphs, we accept that “an s-node sends messages to its neighboring received symbols”.

If $\mathbf{M}$ is a parity-check matrix whose rows correspond to the checks, it can be depicted by Figure 2.7. The green circles in this Tanner graph are c-nodes given by $c_1, \ldots, c_4$, and the red squares are called variable nodes, given by $v_1, \ldots, v_7$. The v-nodes are still the codeword coordinates. Every c-node connects to one single parity-check symbol which is the bitwise sum of all its neighbors. For a valid codeword, all the

Figure 2.6. Another Tanner graph of generator matrix $\mathbf{M}$. 
Checksums should be zeros, i.e., all the checks should be satisfied. For LDPC codes, the parity-check matrix is essential for decoding. At the received side, all encoded symbols with channel information are used to decode.

![Tanner graph of parity-check matrix](image)

*Figure 2.7. Tanner graph of parity-check matrix M.*

Sometimes the v-nodes of an LDPC code are called *left nodes* and the c-nodes are called *right nodes*. Accordingly, with a generator matrix (like Figure 2.5), the s-nodes are called left nodes and the v-nodes are called right nodes. The reason could be that some researchers like to arrange Tanner graphs so that v-nodes are on the left and the c-nodes are on the right.

With an LDPC code, some combinatorial characteristics of the parity-check matrix are important to the code performance. Among these concepts are *cycle* and *stopping set* [19]. A cycle is a closed loop consisting of edges in a Tanner graph. The size, or length, of a cycle is the number of its edges. In Figure 2.8, the highlighted cycle is of size 6. The smallest size of the cycles is called the *girth* of the Tanner graph. Apparently, the girth in Figure 2.8 is 4. Length 4 is also the minimum girth of all Tanner graphs. In a matrix, a length-4 girth represents a rectangle with four 1’s at its four
corners. With LDPC codes, girth 4 is usually prevented in order to achieve good performance.

![Tanner graph diagram](image)

**Figure 2.8.** A cycle of size 6 in the Tanner graph of parity-check matrix $\mathbf{M}$.

The fewer edges exist in a Tanner graph, the sparser the graph is. Low density matrices are very important in designing a code. First of all, sparse codes save computation cost. We define encoding and decoding computation cost by the number of edges. For example, to compute a variable node of degree $i$, $i$ bitwise additions are required, so the cost is $i$. Secondly, very low density matrices can avoid some stopping sets so as to lower error floors. All decoding matrices in this dissertation are binary unless specifically stated.

### 2.6 Set Diagram

We introduced a new representation of the parity-check matrix with LDPC codes: set diagram [30]. A set diagram is composed of two sets of Venn diagrams. One is the variable node set diagram and the other one is the check node set diagram. The variable (check) node set diagram consists of multiple Venn diagrams corresponding to the
variable (check) nodes one-to-one, and an intersection of any two Venn diagrams contains all the common check (variable) node(s) of the two variable (check) nodes.

Give a parity-check matrix

\[
H = \begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}
\]

whose columns correspond to v-nodes and whose rows are c-nodes. The set diagrams of \( H \) are illustrated in Figure 2.9.

![Set diagrams](image)

(a) Variable node set diagram

(b) Check node set diagram

*Figure 2.9. Set diagrams for parity-check matrix \( H \).*

The variable node set diagrams are \( v_1 = \{c_1, c_4\} \), \( v_2 = \{c_2, c_3\} \), \( v_3 = \{c_1, c_3\} \), \( v_4 = \{c_2, c_4\} \), \( v_5 = \{c_1, c_2\} \), and \( v_6 = \{c_3, c_4\} \), and the check node set diagrams are \( c_1 = \{v_1, v_3, v_5\} \), \( c_2 = \{v_2, v_4, v_5\} \), \( c_3 = \{v_2, v_3, v_6\} \), and \( c_4 = \{v_1, v_4, v_6\} \). The nodes pointed to by arrows in a set diagram are also called the *elements* of the set diagram. In the case within Figure 2.9, each set diagram looks like the joining of multiple chain links.
A link loop that closes back on itself is exactly a cycle, and the size of the cycle is twice the number of the links in the loop.

The parity-check matrix $H$ is a *regular* matrix whose row weight and column weight are constants: 3 and 2, respectively. If either of the row weight and column weight is not a constant, the matrix is *irregular*. Another important characteristic of this matrix is that any two rows (columns) share at most one intersecting column (row). This property avoids length-4 cycles in the matrix. In fact, its girth is 6, and in its set diagram any pair of Venn diagrams have no more than one common node. On the other hand, it just needs to contain two variable (check) nodes in the intersection of two ‘chain links’ to produce a length-4 cycle. Here is an example.

Another parity-check matrix is

$$H_1 = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \text{(2.12)}$$

and its set diagrams are shown in Figure 2.10.

**Figure 2.10.** Set diagrams for parity-check matrix $H_1$. 
From the above examples and description, we see that the set diagram representation is actually a set of variable (check) nodes contained by some Venn diagrams corresponding to their neighboring check (variable) nodes in the Tanner graph. Apparently, a set diagram is a set of non-zero weight columns (rows) in parity check matrix. Therefore, it is easy to transform between set diagram, parity-check matrix and Tanner graph. To design an LDPC code, the variable node set diagram is usually selected and the check node set diagram is used to assist in searching for some special structures.

One of the important functions of the set diagram is to visualize operations on an LDPC code or describe the code combinatorial characteristics, such as cycles and stopping sets. By computer searching, the variable (check) node sets can be used to easily find combinatorial characteristics or design the structure of an LDPC matrix.

2.7 Multicast and Broadcast

Multicast and broadcast are common methods of data packet transmission in computer networking. Both the schemes enable data delivery to a group of destinations simultaneously in a single transmission, but the multicast mode only establishes transmission links to a subset of terminals in the network that requires the data and the broadcast method transfers messages to all terminals. Examples of multicast and broadcast are shown in Figure 2.11. These two one-to-many communication techniques are widely used for IPTV (Internet Protocol television), the Internet download, mobile multimedia services, and LAN (local area network), etc. Each transmission link has its
own channel model. In order for bandwidth and power efficiency and to avoid retransmission, we need a universal erasure code. A rateless code can be qualified.

![Multicast and Broadcast Diagram](image)

*Figure 2.11. Examples of multicast and broadcast.*

### 2.8 Three Early Remarkable Erasure Codes

*Reed-Solomon (RS) Codes* [4] [62] are cyclic linear codes over $GF(2^r)$ with polynomial generators. These codes can be used to approximate an ideal code since, with a message of $k$ symbols, the RS codes can recover all the source symbols from $n$ received encoded symbols with $n$ slightly greater than $k$. However, Reed-Solomon codes lose efficiency for large $k$ and $n$, requiring quadratic encoding/decoding time [5] [6].

*Low-density parity-check codes* [17] – [19], also known as Gallager codes, were invented by Gallager and published in 1963. These once forgotten codes can allow data transmission rate close to the Shannon Limit. The performance of a LDPC code is determined by its parity-check matrix and decoding algorithm. Common decoding schemes include the belief propagation algorithm and the maximum likelihood (ML) algorithm. The first scheme passes decoding information iteratively between the v-nodes
and c-nodes along the edges in the Tanner graph of the parity-check matrix. In general, the more iterations the decoding goes, the better performance it realizes. The maximum likelihood decoding scheme is similar to Gaussian elimination algorithm. However, an LDPC code cannot guarantee efficient and reliable transmission over multicast and/or broadcast networks where various or unknown data loss patterns exist or asynchronous data access is required.

_Tornado codes_ [8] [9] are a class of erasure codes, which support error correcting, based on multi-layer irregular sparse graphs. These loss-resilient codes solve some packet loss problems in real-time and multicast environments. The most outstanding property of a Tornado code [9] [10] is that with code rate $R = 1 - p(1 + \varepsilon)$ just below the channel capacity $1 - p$ it can recover $k$ original symbols from $(1 + \varepsilon)k$ randomly collected encoded symbols in time proportional to $n \ln (1/\varepsilon)$ with high probability, where $p$ is the channel loss rate, $\varepsilon > 0$ and $n$ is the encoding length. These fast linear-time codes are however limited by their fixed code rates so they are not universal erasure codes. A very low rate Tornado code may match the worst expected loss rate of the channel, but this solution requires large memory proportional to the encoding size.

Many of today’s network communications require strict bandwidth efficiency, power efficiency, low computation cost and short delay. Big challenges have risen from variable loss rates, asynchronous data access and channel handoff etc. For erasure codes, one looks forward to fast encoding and decoding, reliable transmission, efficient recovery, and loss rate tolerance. None of Reed-Solomon codes, LDPC codes and Tornado codes can meet all these requirements very well, especially in multicast or
broadcast environments. Fortunately, during the past decade, a new solution – digital fountain codes – has been brought forth to solve these problems.

2.9 Summary

The Internet is a good example of packet erasure channels and bit error erasure channels. In the communication networks, original data are contained in individual packets and then they are forwarded to destinations. For several reasons, some of the transmitted packets are randomly lost on the way with non-zero probabilities. Once a packet arrives at the destination, it is examined with error detection. If any detected errors cannot be corrected, this packet would be discarded and/or marked as an erasure. For point-to-point communication, the traditional best-effort service can make up all the lost data if enough transmission time is permitted. However, for today’s point-to-multipoint multimedia services in high demand, the best-effort service no longer satisfies the great need of erasure correction in terms of bandwidth, reliability and delay etc.

Besides erasure channels, the models of AWGN channel and Rayleigh fading have been reviewed. The channel capacity of a noisy channel is the bound that a channel code can do the best.

We study linear erasure codes using Tanner graphs and set diagrams in addition to matrices. Their structures and some properties have been reviewed. The Tanner graph visualizes the operations of encoding and iterative decoding. It is convenient to enable searching code combinatorial properties (like cycles) and operations of linear codes using set diagrams.
Some remarkable erasure codes have been well developed to solve the multicast and broadcast communications over the Internet. Examples of such codes are Reed-Solomon codes, LDPC codes and Tornado codes. None of these codes are universal codes. We need rateless codes to fit the various loss patterns environment. Digital fountain codes fulfill the need. They can restore the original data with small decoding overhead and without the knowledge of the channel state. Some fountain codes have linear encoding and decoding time.
CHAPTER 3: CONVENTIONAL FOUNTAIN CODES

Motivated by reliable data distribution to amounts of autonomous clients, Byers et al. [11] introduced the “digital fountain approach” concept in 1998. Under this new design of multicast and broadcast protocols, a receiver can restore the source data with encoded symbols randomly collected from over a lossy channel. The channel loss pattern can be even unknown and data access is initiated in arbitrary time. Although a Tornado code can somewhat meet this basic idea, it must estimate the channel loss rate in advance and necessarily assigns a fixed code rate $R$ close to the loss rate, which in return limits the number of user nodes that is roughly equal to $1/R$ [10] [12]. Therefore, the characteristics of digital fountains should be: reliable delivery, no retransmission, no feedback, efficient encoding and decoding, data access on demand, and loss rate tolerant.

Here is the big picture of a theoretical digital fountain over a memoryless erasure channel. A requested file is divided into $k$ blocks of same size and each block is encapsulated into a packet. Given these $k$ packets, the fountain encoder can potentially generate a limitless stream of independent and identically distributed encoding packets. These encoding packets are transmitted over the erasure channel, but only a fraction of the packets are received error-free and the others are lost or discarded. The fountain decoder then, with high probability, recovers the source packets from any subset of received encoding packets of number $n$ equal to or larger than $k$. This process is like that we use a cup to randomly collect some drops from a fountain and once having enough drops in cup we can satisfy our thirst.
Fountain codes are rateless codes, since at both the transmitter and receiver sides the code rate is not fixed and can potentially approach zero. However, in practice one just needs to consider truncated fountain codes. According to application requirement, like decoding success rate and computation cost, there may be an assigned minimum number of received packets to meet the requirement. For instance, this minimum number is \( n \), and the decoder recovers data with the rate \( k/n \) truncated fountain code. For an ideal or optimal fountain code, the code rate should be 1. When \( k \) is slightly smaller than \( n \), the fountain code is sub-optimal or near optimal. Today there are two main classes of practical fountain codes: Luby Transform (LT) codes and Raptor codes. Both of them are near optimal for finite data length.

### 3.1 Luby Transform Codes

Invented by Luby, LT codes \[13\] are the first class of practical fountain codes. As a linear code, the LT code can generate a resilient number of i.i.d. encoded symbols. Rebuilding up a generator matrix, the LT decoder can restore the original message on the fly from arbitrarily collected encoded symbols with small decoding overhead. The conventional LT codes are fast and efficient with iterative message passing (MP) decoding algorithm with hard decoding information. They can acclimatize to the Internet multicast and broadcast environments just because of a degree distribution for the output encoded symbols. At the receiver, the neighboring information of each encoded symbol is utilized to construct the Tanner graph of a generator matrix. There are quite a few methods to correctly propagate the output symbol’s index information to the decoder. In
3.1.1 Encoding

An LT encoded symbol of degree $d$ is the bitwise sum of $d$ source input symbols randomly selected from a special degree distribution. The $d$ distinct neighbors of the encoded symbol should be uniformly chosen from all the source symbols. Since the channel is lossy and received symbols are not necessarily in the order of transmission, the degree distribution for the output symbols is the most important part of LT codes. It enables the LT code’s efficient erasure recovery from randomly collected i.i.d. symbols without need of knowing the start of the source message.

It is the Robust Soliton distribution (RSD) [13] that guarantees the near optimal LT code’s universality on all kinds of erasure channels. The RSD is defined as follows:

\begin{equation}
\tau(d) = \begin{cases} 
\frac{r}{(dk)}, & d = 1, \cdots, k/r - 1 \\
\frac{r \ln(r/\delta)}{k}, & d = k/r \\
0, & d = k/r + 1, \cdots, k 
\end{cases},
\end{equation}

where $k$ is the number of input symbols (code dimension), $d$ is the degree of the output symbol, and each source symbol is not neighbored by any encoded symbol with a probability at most $\delta$. The probability mass function (pmf) of the RSD is stated as

\begin{equation}
\mu(d) = [\rho(d) + \tau(d)]/\beta,
\end{equation}

where
\[\rho(d) = \begin{cases} 
\frac{1}{k}, & d = 1 \\
\frac{1}{[d(-1)]}, & d = 2, \ldots, k \\
0, & \text{otherwise} 
\end{cases} \quad (3.3)\]

and

\[\beta = \sum_{d=1}^{k} [\rho(d) + \tau(d)]. \quad (3.4)\]

The Robust Soliton distribution for an LT code of dimension \(10^4\) is shown in Figure 3.1. Luby suggested that the number of received encoded symbols be \(n = \beta k\). The corresponding decoding overhead is then

\[\varepsilon = \frac{n}{k} - 1 = \frac{1}{R} - 1 = \beta - 1 \quad (3.5)\]

The average degree of encoded symbols is plotted in Figure 3.2. The average degree increases logarithmically versus the code dimension.

The overhead decreases as the code dimension \(k\) is getting larger because \(\beta\) is a decreasing function with \(k\), as shown in Figure 3.3, but the sparseness of the generator matrix becomes lower and lower.
Figure 3.1. RSD for dimension $10^4$ with $c = 0.1$ and $\delta = 0.05$. 
Figure 3.2. Average degree of LT encoded symbols with $n = \beta k$, $c = 0.1$ and $\delta = 0.05$. 
The common method of generating a stream of encoded symbols is to create the rows of the generator matrix \( G \) on the fly or in advance. Once a row \( G_i \) is determined, the corresponding encoded symbol \( y_i \) is then the dot-product of the row and the column vector \( s \) for all \( k \) input symbols. The encoded symbol is given by

\[
y_i = G_i s,
\]

where \( i = 1, 2, \ldots \). From this point of view, the output symbols are independent of each other and the order of columns does not matter in the code’s performance. The encoded symbols will be transmitted as needed. When the receiver collects enough encoded symbols, according to its application requirement, it can feed back a message to inform

*Figure 3.3. \( \beta \) for LT codes with \( c = 0.1 \) and \( \delta = 0.05 \).*
the transmitter of reception complete. Once the transmitter has completion messages from all requesting terminals, it can stop sending encoded symbols.

Because the decoder knows all the neighbors of each received encoded symbol, it reproduces all the corresponding columns in the original generator matrix. With these restored columns, the decoder establishes a generator matrix of a truncated LT code. It means that the encoder creates a \( m \times k \) generator matrix \( A \) (\( m \) is as large as needed) and the Hamming weight of each column is determined by the RSD. The output symbols are transmitted over an erasure channel and some of them are erased. At the receiver side, with \( n \) received symbols, the decoder sets up a \( n \times k \) generator matrix \( G \) (\( k \leq n \leq m \)) which is actually made of \( A \) without those rows corresponding to the erased symbols. Apparently, the \( n \) received symbols are a random subset of the \( m \) transmitted symbols.

### 3.1.2 Decoding with Message Passing Algorithm

When releasing the invention of LT codes, Luby introduced a message passing decoding algorithm to these erasure codes. The message passing algorithm is a hard decoding scheme, which can be very fast and efficient. We call ‘LT codes with Robust Soliton distribution under message passing decoding’ conventional LT codes or Luby’s LT codes. There are two types of message passing decoding with the LT code: serial decoding, and parallel decoding.

The serial decoding procedure of LT codes is as follows:

Step 1: Find one v-node of degree 1 and assign its symbol value to its neighboring s-node. The v-node is now said to have been released, no longer useful,
and the neighboring s-node is recovered. At the same time, the edge incident on this v-node is removed, which is equivalent to delete a corresponding row from the generator matrix established by the decoder.

Step 2: The value of the newly restored s-node is XORed to the rest of its neighbors (not including the one just released) and its edges are then removed from the graph, i.e., the column corresponding to the recovered source symbols is deleted from the generator matrix.

Step 3: If all the s-nodes are recovered, the decoding is complete and successful. Otherwise, repeat from Step 1. If there is no degree-1 v-node and the decoding is not complete, flag failure.

The parallel decoding can be implemented as easily as the serial method but runs faster. An example of the parallel message passing decoding of an LT code is given in Figure 3.4. The code dimension is \( k = 4 \) and the number of received packets is \( n = 7 \), so an \((n, k) = (7,4)\) truncated LT code is considered at the receiver. Suppose the generator matrix of the truncated LT code is the \( \mathbf{M} \) given by Equation 2.10. Because a packet is equivalent to a symbol which can be represented by a bit sequence whose size ranges from one bit to multiple bits, for simplicity we set the symbol size to be one bit. Suppose the received encoded symbols are \([1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1]\). It takes four iterations to restore the four source symbols \([0 \ 0 \ 1 \ 1]\), in the serial manner but it needs only two iterations to finish the parallel decoding. One can verify the decoding result with \( [0 \ 0 \ 1 \ 1] \mathbf{M} = [1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1] \).
If the passed messages are trapped by a stopping set, the decoding cannot be continued even though the initial generator matrix at the decoder has full rank. Here is an example. Assume that the receiver constructs a generator matrix of full rank 4, given by

\[
G = \begin{bmatrix}
1 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{bmatrix},
\]

(3.7)
and the received encoded symbols are \([1 \ 0 \ 1 \ 0 \ 0 \ 0]\). The decoding procedure is shown in Figure 3.5. As the first iteration is done, the decoding is terminated since there is no v-node of degree 1 for the second iteration. It is said that the decoding is halted by a stopping set composed of \(v_1, v_3, v_4, v_5, v_6\) and \(v_7\). All these unreleased v-nodes are of degree 2 or more.

![Figure 3.5](image-url)

(a) The first iteration

(b) Decoding stops and fails

*Figure 3.5*. An example of unsuccessful parallel hard MP decoding of an LT code.
Recall Equations 3.1 – 3.4 and Figure 3.1. One “miracle” of the Robust Soliton distribution is that it usually ensures more than one degree-1 encoded symbol in each decoding iteration and that a degree-2 symbol drops to degree 1 with very high probability when an iteration is done. In this dissertation, message passing decoding with LT codes is meant to be the parallel scheme unless otherwise stated.

### 3.1.3 Coding Complexity

In order to investigate the complexity of LT codes, we first review an inspiring game presented by MacKay [14]. This is a balls-and-bins experiment. Ideally, every encoded symbol is neighbored by one single source symbol. Randomly throw $n$ identical balls to $k$ equal bins. If one ball is thrown, every bin has a probability $1/k$ to be hit. Under these conditions, given $n$ balls thrown, the probability that a certain bin has no hit is $(1 - 1/k)^n$. If $k$ is large, we approximate this probability by $(1 - 1/k)^n \approx e^{-n/k}$ via $\lim_{x \to \infty} (1 + 1/x)^x = e$. In order to hit all $k$ bins with probability at least $1 - \delta$ ($0 < \delta < 1$), we should have $(1 - 1/k)^n < \delta$, and hence $n > k\ln(k/\delta)$ for large $k$. This lower bound will help analyze the degree distribution.

Although the Robust Soliton distribution facilitates practical LT codes, the coding cost remains a problem that the computation complexity is in order of $\ln(k/\delta)$ on average. With $n = k\beta$ encoded symbols, the average degree of the encoded symbol is

$$D_{e,\text{avg}} = \sum_{d=1}^{k} d\mu(d)$$

$$= \sum_{d=1}^{k} d[\rho(d) + \tau(d)]/\beta. \quad (3.8)$$

So we obtain
\[
D_{e,avg} \leq \sum_{d=1}^{k} d[\rho(d) + \tau(d)].
\] (3.9)

The average degree of the source symbol is

\[
D_{s,avg} = D_{e,avg} n/k \\
= \sum_{d=1}^{k} d[\rho(d) + \tau(d)] \\
= 1/k + \sum_{d=2}^{k} d/d(d-1) + \sum_{d=1}^{k/r-1} d r/(dk) + (k/r)(r/k)\ln(r/\delta) \\
\approx H_k + 1 + \ln(r/\delta),
\] (3.10)

where \(H_k\) is the harmonic sum up to \(k\). Luby [13] stated that, for large data length, the computation complexity of LT codes is of \(O(\ln(k/\delta))\) on average.

### 3.2 Raptor Codes

A Raptor code [16] is an extension of an LT code but can achieve linear complexity with message passing decoding. Raptor codes have been adopted by The 3rd Generation Partnership Project (3GPP) [15], a collaboration agreement that was established in December 1998. Within the frame of 3GPP, Raptors codes are used for reliable data delivery in mobile/wireless broadcasting and multicasting.

#### 3.2.1 Encoding

Raptor codes are concatenated codes. A diagram of general Raptor encoding is shown in Figure 3.6. The source data symbols are first pre-coded by an outer code \(C\). The output symbols of the pre-code are called intermediate symbols and they are the input symbols of an inner LT code. The pre-code is a fixed rate erasure code usually with fairly high rate. The pre-code can be multistage, i.e., the pre-code can be a concatenation of
multiple fixed rate erasure codes. The inner LT code is sometimes called a weakened LT code [5].

A Raptor code can be denoted by \( (k, c, \Omega(x)) \), where \( k \) is the number of source symbols and \( \Omega(x) \) is the polynomial of the encoded symbol degree distribution that the inner LT code takes on the intermediate symbols. \( \Omega(x) \) is stated as

\[
\Omega(x) = \sum_d \Omega_d x^d,
\]

where \( d \) is the chosen degree of the LT encoded symbol and \( \Omega_d \) is the probability that the value \( d \) is chosen. The average degree of the encoded symbols is equal to \( \Omega'(1) \), where \( \Omega'(x) \) is the derivative of \( \Omega(x) \) with respect to \( x \).

Shokrollahi [16] introduced the Soliton distribution for the weakened LT code. With \( \varepsilon \) a real number larger than zero and \( D = \lceil 4(1 + \varepsilon)/\varepsilon \rceil \), the polynomial for the Soliton distribution is stated as

\[
\Omega(x) = \frac{1}{1+\mu} \left[ \mu x + \sum_{d=2}^{D} \frac{x^d}{(d-1)d} + \frac{x^{D+1}}{D} \right],
\]

where \( \mu = (\varepsilon/2) + (\varepsilon/2)^2 \).
An example of the Soliton distribution of a weakened is shown in Figure 3.7. In this example, with a rate 0.5 pre-code of block length 2304 and $\varepsilon = 2/3$, the Soliton distribution is determined by Equation 3.12. Compared with the Robust Soliton distribution, the Soliton distribution could have much higher fraction of degree-1 symbols. Although the weakened LT code usually does not restore all the intermediate symbols, it can recover most of the intermediate symbols with a small number of iterations, and the average degree of the encoded symbols is a constant, no matter how large the data length is.

Figure 3.7. Soliton distribution of a weakened LT code.
3.2.2 Message Passing Decoding

The iterative message passing algorithm is run on two matrices: the generator matrix of the inner LT code, and the parity-check matrix of the outer erasure code. The intermediate symbols are not only the v-nodes of the Tanner graph of the pre-code’s parity-check matrix, but also the s-nodes of the LT codes’ generator matrix. The LT decoder passes updated decoding message to the pre-code decoder and hard decision is made on the output of the pre-code decoder is. Before or when the maximum assigned number of iterations is finished, if all the intermediate symbols have obtained nonzero values, the decoding is successful.

There are two types of message passing algorithms for Raptor codes [24]. In the first method, the LT decoder apply message passing algorithm to recover intermediate symbols as many as it can until the LT decoding is halted by a stopping set. Then the values of the recovered intermediate symbols are passed to the v-nodes of the Tanner graph of the pre-code’s parity-check matrix and the outer decoder restores the rest of the intermediate symbols. This method is a local-iteration scheme. The second method is a global-iteration scheme. Each decoding iteration consists of two stages: one iteration first on the LT code and then one iteration on the pre-code. At the end of either stage, the updated values of the intermediate symbols are passed to the other decoder.

The inventor of Raptor codes, Shokrollahi, stated in [28] that the pre-codes for good raptor codes are generally of high rate. The higher the pre-code rate, the fewer intermediate symbols are created. So, with a same Raptor code rate, greater fraction of
intermediate symbols can be restored by a same or similar inner LT code and then the
error performance is accordingly better.

### 3.2.3 Coding Complexity

Shokrollahi [16] proves that, with a carefully designed Soliton distribution $\Omega(x)$
and an appropriate $R_C$ pre-code $C$, Raptor codes achieve linear encoding and decoding and
the space consumption and the overhead are arbitrarily close to the minimum.

A suitable pre-code $C$ should be a linear code of block-length $n$ with such
properties:

$$R_C = \frac{1 + \varepsilon/2}{1 + \varepsilon},$$

and the belief propagation decoder [17] can decode $C$ on a BEC with erasure probability
$\delta = (\varepsilon/4)/(1 + \varepsilon) = (1 - R)/2$ with $O(\ln(1/\varepsilon))$ arithmetic operations on average for
each symbol. According to Equation 3.13, the pre-code rate is bounded by

$$0.5 < R_C < 1,$$

Shokrollahi [16] also stated that a $(k, C, \Omega(x))$ Raptor code has space consumption
$1/R$, overhead $\varepsilon$ (collecting $(1 + \varepsilon)k$ Raptor encoded symbols to decode), and a cost of
$O(n\ln(1/\varepsilon))$ with respect to BP decoding of both the pre-code and the LT code. The
error probability of this decoder is only polynomially small in $k$, rather than exponentially
small, because the error probability of the decoder for the pre-code $C$ has this property.
For a one-stage pre-coded Raptor code, Shokrollahi [16] defines the encoding cost by
$E(C)/k + \Omega'(1)$, where $E(C)$ is the number of arithmetic operations required to generate a codeword in $C$.

### 3.3 Summary

Conventional digital fountain codes with message passing decoding are universal erasure codes. LT codes, the first class of practical fountain codes, can achieve near capacity performance on arbitrary erasure channels. This advantage is more notable for large code dimension, because the average decoding overhead is smaller. However, the code complexity increases logarithmically as the code length increases. Because all the source symbols have to be covered and decoding stopping sets must be avoided, the average degree of the encoded symbol grows logarithmically.

Raptor codes are improved versions of LT codes with constant encoding and decoding complexity. In a Raptor code, both the pre-code and the weakened LT code are very sparse graph codes. The polynomial of the encoded symbol degree distribution ensures linear coding time. The weakened LT code usually restores most of the intermediate symbols with a small number of decoding iterations and the pre-code corrects the other intermediate symbols as erasures.
CHAPTER 4: TWO SYSTEM MODELS OF FOUNTAIN CODES WITH MP DECODING

The conventional LT codes and Raptor codes are packet erasure codes. Once a transmitted packet is received, some error detecting and/or correcting operations are run within the payload of the packet. A fountain code may not do anything for the error correction within a single packet. For LT codes, the accepted packets comprise a codeword and a corresponding generator matrix is established by the decoder. The message passing decoding algorithm is applied on the Tanner graph of the generator matrix to find out all the source symbols.

However, researchers have found that fountain codes with message passing decoding can also correct individual symbol errors introduced over noisy channels, like bit errors within a frame or packet payload. In this sense, just like other linear error correcting codes, a fountain code is able to correct bit errors resulting from Gaussian noise channels, symmetric channels and fading channels etc. In practice, we can use interleaving to approximate memoryless channels.

Stockhammer et al. [20] proposed a system model for packet erasure LT codes. Sivasubramanian and Leib [21] designed a model for bit correcting Raptor codes. In this chapter, revising their models, we introduce two system models with structure unification to describe digital fountain codes on the packet level and the bit level. They are actually models of Raptor codes, since LT codes are special cases of Raptor codes.
4.1 System Model of Fountain Codes for Packet Erasure Correction

Taking the system designs in [20]–[22] as references, this dissertation presents a source-to-multi-destination system of fountain codes with message passing decoding for packet erasure correction. The fountain-encoded packets are sent from a single source to multiple asynchronous or synchronous user terminals on erasure channels with arbitrary loss patterns. Acknowledgement feedback is permitted so that data transmission can be terminated as soon as the provider knows there is no more request. In case of no feedback, the maximum broadcast or multicast duration can be assigned. Communication loss or handoffs occur at random time. All synchronizations are perfect. This system is illustrated in Figure 4.1.

Consider a source file \( a \). This file is divided into some groups, or frames, and the sizes of these groups may not be equal. Each group is further split into multiple blocks and the sizes of these blocks are the same. Additional non-information bits may be
engrafted into one or more blocks to make equal sizes. In some cases, the original file is
directly partitioned into blocks, i.e., one file is one group. Every block is an information
symbol. In a single group, all the \( k \) symbols \( \mathbf{u} = (u_1, \ldots, u_k) \) are input into the fountain
decoder once a user requests the file. For a Raptor code, the encoder (decoder) consists of
one pre-code encoder (decoder) and one LT encoder (decoder). The source symbol group
\( \mathbf{u} \) is pre-encoded into a sequence of intermediate symbols \( \mathbf{i} = (i_1, \ldots, i_N) \) of fixed length
\( N \). For an LT code only, the encoder and decoder of the pre-code are not needed, i.e.,
\( \mathbf{u} = \mathbf{i} \) and the LT decoder produces \( \mathbf{m}_v \). Similarly, for pre-code only (PCO), the encoder
and decoder of the LT code are removed and this model is a system for linear block
codes. The intermediate symbols are LT encoded to an unbounded stream \( \mathbf{s} = (s_1, s_2, \ldots) \). The fountain output symbols \( \mathbf{s} \) are encapsulated into packets \( \mathbf{t} \) (of the same
block length as \( \mathbf{s} \)) and transmitted over an erasure channel. A requester accesses the data
at an arbitrary time and collects output packets without need of the start of the source file.

In this system, the channel erasure pattern represents the packet loss model. Before collected by the receiver, the transmitted packets are multiplied by the erasure
pattern \( \mathbf{b} = (b_1, b_2, \ldots) \) with \( b_i \in \{0,1\} \). If \( b_i = 0 \), the transmitted packet \( t_i \) is erased. The
loss reason can be that buffer overflow occurs in a router at some point on the channel or
that, at the receiver, detected errors in the packet cannot be corrected. If \( b_i = 1 \), the
packet \( t_i \) is declared to be error free and will be accepted by the receiver. The pattern
\( \mathbf{b} = (b_1, b_2, \ldots) \) is also of the same length as \( \mathbf{t} \).

Accepted packets are forwarded to the fountain decoder. The index information is
withdrawn from the header of every packet and the decoder uses this information to
restore the generator matrix of the truncated weakened LT code and the parity-check matrix of the pre-code. The payloads of the packets are extracted and, viewed as received symbols $r$, they are connected to their corresponding c-nodes in the Tanner graph of the LT generator matrix. With the local-iteration decoding scheme, the LT decoder outputs intermediate symbol values $m_{v,i}$ to the pre-code decoder after the LT decoding finishes. The pre-code decoder takes over the messages and corrects the rest of the erasures among the intermediate symbols. The pre-code decoder never feeds back any messages to the LT decoder. With the global-iteration decoding scheme, in every single decoding round, the LT decoder first runs one iteration and forwards updated intermediate symbols values $m_{v,i}$ to pre-code decoder, and then the pre-code decoder executes one iteration and feeds back updated intermediate symbols $m_{v,i}$ to the LT decoder. With either of the schemes, if the decoding fails, the receiver may collect more packets to continue to decode. After the fountain decoding succeeds, the recovered symbols are unified to restore the original file.

4.2 System Model of Fountain Codes for Bit Error Correction

We discussed a source-to-multi-destination system of fountain codes with global message passing decoding for bit transmission over and AWGN channel [22]. This system allows synchronous or asynchronous data access. Acknowledgement feedback is still permitted and all synchronizations are perfect. Handoffs and communication interruptions occur at arbitrary time. This system model is modified to fit both local and global-iteration decoding methods, as illustrated in Figure 4.2.
This system is general for linear block codes, LT codes and Raptor codes with message passing decoding on binary erasure, AWGN or flat fading channels. Consider a source file $\mathbf{a}$. This file is split into multiple groups, or frames. Every single group $\mathbf{u} = (u_1, \ldots, u_k)$ is input to the fountain encoder. The fountain encoder and decoder are the same as those in the fountain code system shown in Figure 4.1. The only difference is that the decoding messages in the first system can be hard information and the decoding information in the second system is usually soft. In fact, the first system is a special case of the second one. For an LT code only, the pre-code encoder and decoder are removed: $\mathbf{u} = \mathbf{I}$ and the LT decoder outputs $\mathbf{m}_v$. For PCO, it is a system model of linear block codes. The fountain output bits $\mathbf{s} = (s_1, s_2, \cdots)$ will be transmitted in a modulated form.
\( \mathbf{t} = (s_1, s_2, \cdots) \). The data can be accessed asynchronously and interruption of reception is tolerated.

In this model, the channel introduces gain and/or additive white Gaussian noise to all transmitted bits. The channel gain is a non-negative real vector \( \mathbf{h} = (h_1, h_2, \cdots) \) and its elements are in ordinal manner the gains of the bits in \( \mathbf{t} \). The gain vector can be a constant or approximate any flat fading models. Interleaving can be applied to make the gains memoryless. We assume the channel gain \( h_i \) is constant over the entire slot of the \( i \)-th transmitted bit. The Gaussian noise \( \mathbf{g} \) has zero mean and two-sided power spectral density (PSD) \( N_0/2 \). At the receiver, the output of the matched filter is sampled at the end of the bit duration. At this point, the Gaussian noise sample \( g_i \) is still zero mean and its variance is \( N_0/2 \). Before input to the fountain decoder, these samples are ‘selected’ by the receiver pattern \( \mathbf{b} = (b_1, b_2, \cdots) \) with \( b_i \in \{0,1\} \) that stands by discarding, communication loss or bit acceptance, which depends on the particular channel model. In the case of PCO, \( \mathbf{b} \) is usually an all ones sequence.

Accepted bits are forwarded to the fountain decoder. The decoder has already known the decoding matrices: the generator matrix of the truncated weakened LT code and the parity-check matrix of the pre-code. After setting up the Tanner graphs of the decoding matrices, the decoder connects all accepted bits to their corresponding c-nodes in the Tanner graph of the LT generator matrix. Note that the c-nodes of the erased bits are not in the graph. The message passing decoding can be local-iteration or global-iteration. With the global-iteration scheme, at the end of every decoding round, the decoder can make hard decision on the intermediate bit values \( \mathbf{m}_v \). But with local-
iteration scheme, hard decision is not made in the first decoding stage: LT decoding. If the decision is a codeword of the pre-code, the iterative decoding finishes with a success flag and decision is forwarded to the unifier to restore the original file.

4.3 Details of Raptor Encoder

In order to illustrate the Raptor encoder, assume the pre-code is a one-stage LDPC code. For LDPC codes, there are two common encoding methods. One is straightforward encoding by the generator matrix and the other is fast encoding by the parity-check matrix. Thus, two types of efficient and practical Raptor encoders are available. These generator matrices and parity-check matrices are also decoding matrices.

4.3.1 Straightforward Encoder

The first type of Raptor encoder is depicted in form of Tanner graph in Figure 4.3.
Figure 4.3. Straightforward encoding of Raptor codes.

The straightforward encoding can be executed if the generator matrix of the pre-code, $G_{LDPC}$, is available, since the generator matrix of the outer LT code, $G_{LT}$, can be constructed on the fly. Usually the parity-check matrix of a designed LDPC code, $H_{LDPC}$, is already known, so its generator matrix can be computed. Given a $(N - k) \times N$ $H_{LDPC}$ ($k < N$), calculate $H_{LDPC_{RREF}}$, the reduced row echelon form (RREF) of $H_{LDPC}$. With column permutations on $H_{LDPC_{RREF}}$, $H_{SYS}$, the systematic form of $H_{LDPC_{RREF}}$, is obtained, whose structure can be stated as

$$H_{SYS} = [I_{N-k} \ X],$$

(4.1)

where $I_{N-k}$ is an $(N - k) \times (N - k)$ identity matrix. Therefore, a generator matrix corresponding to $H_{SYS}$ is given by
\[ G_{SYS} = [X^T \ I_k]. \] (4.2)

If \( H_{SYS} = [X \ I_{N-k}] \), the generator matrix can be found by
\[ G_{SYS} = [I_k \ X^T]. \] (4.3)

After inverse column permutations, \( G_{LDPC} \), a generator matrix of the pre-code, is obtained.

From \( H_{LDPC} \) to \( G_{LDPC} \), most of the complexity could result from computing \( H_{LDPC\_RREF} \). A classic algorithm is Gaussian elimination and its cost is of \( O(N^3) \). The complexity is \( O(N^2) \) to encode a source block by multiplying it by \( G_{LDPC} \).

4.3.2 Fast Encoder

The second type of Raptor encoder is shown in Figure 4.4. Fast encoding on low-density parity-check matrices is feasible and it is very useful for Raptor encoding. In this subsection, we describe two fast encoding methods with parity-check matrix.
Figure 4.4. Fast encoding of Raptor codes.

The first fast encoding algorithm was introduced by Richardson and Urbanke [59] that reduces encoding cost to $O(N + g^2)$. The positive integer $g$ is wanted to be as small as possible. This fast encoding algorithm is general for all LDPC codes and it is well introduced in [25] [26] [59], as follows. A given very sparse $(N - k) \times N$ parity-check matrix $H$ is rearranged into an approximate lower-triangular form by row and column permutations, as shown in Figure 4.5. $T$ is a lower-triangular matrix that has the all 1’s main diagonal and all the entries above the main diagonal are 0’s.
That is,

$$H = \begin{bmatrix} A & B & T \\ C & D & E \end{bmatrix}. \quad (4.4)$$

Since the original $H$ is very sparse, the permutated $H$ is still very sparse so it makes possible that the encoding cost is approximately linear. If this parity-check matrix has full rank $N - k$, the code’s dimension is $k$. Suppose $H$ is a binary matrix and all computation is modulo-2 arithmetic. Given any source block $u$ (a row vector), the encoded sequence $v$ is in a systematic form

$$v = [u \ p_1^T \ p_2^T]. \quad (4.5)$$

The parity portions $p_1$ and $p_2$ are developed with the following rules:

1. Find the upper syndrome of $u$,

$$z_A = Au^T. \quad (4.6)$$

2. Compute a sequence of parity-check bits, $p_2^A$, which forces the upper syndrome to 0.

$$p_2^A = T^{-1}z_A. \quad (4.7)$$

3. Find the lower syndrome of the vector $[u \ 0 \ p_2^A]$,
\[ z_B = Cu^T + Ep^A. \]  \hspace{1cm} (4.8)

4. Define

\[ F \equiv ET^{-1}B + D. \]  \hspace{1cm} (4.9)

Then find the first parity portion

\[ p_1 = F^{-1}z_B. \]  \hspace{1cm} (4.10)

5. Calculate the new upper syndrome

\[ z_C = z_A + Bp_1. \]  \hspace{1cm} (4.11)

6. Finally the other set of parity-check bits, \( p_2^A \), can be obtained, such that the upper syndrome is zero,

\[ p_2 = T^{-1}z_C. \]  \hspace{1cm} (4.12)

Almost all the six steps can be done in linear time. \( p_2^A \) in Step 2 and \( p_2 \) in Step 6 are found in linear time by back-substitution. In Step 4, the computation of \( F \) is of \( O(g^3) \) but this is done once only, prior to encoding any source blocks. The complexity of Equation 4.12 is of \( O(g^2) \). That is why \( g \) is wanted to be as small as possible. For example, the low-density parity-check matrices designed in the IEEE 802.16e standard are exactly in the form of Equation 4.4 and \( g = 1 \).

The second fast encoding algorithm is applicable if a parity-check matrix has a staircase structure. When such a parity-check matrix is very sparse, systematic encoding is workable and the parity bits can be computed in linear time [26]. An example of right-hand side staircase low-density parity-check matrix is illustrated in Equation 4.13. The left submatrix A of \( H \) is very sparse. The rest part of \( H \) is actually a lower-triangular matrix that has all ones just underneath the main diagonal.
Given a source block \( \mathbf{u} \), the first \( k \) encoded bits will be \( \mathbf{u} \). The \( N - k \) parity bits \( \mathbf{p} \) can be computed through an accumulator as follows.

\[
\begin{align*}
    p_1 &= \sum_{i=1}^{k} H_{1,i} u_i \\
    p_2 &= p_1 + \sum_{i=1}^{k} H_{2,i} u_i \\
    p_3 &= p_2 + \sum_{i=1}^{k} H_{3,i} u_i \\
    &\vdots \\
    p_{N-k} &= p_{N-k-1} + \sum_{i=1}^{k} H_{N-k,i} u_i.
\end{align*}
\]

\[(4.14)\]

### 4.4 Details of Raptor Decoder

This Raptor decoder configuration is for both the fountain system models introduced in Sections 4.1 and 4.2. No matter the fountain code is applied on packet or bit level, the message passing algorithm is the same. The Raptor decoder configuration is also general for the local-iteration and global-iteration decoding schemes. Sivasubramanian and Leib [24] presented the algorithms for the local-iteration decoder. Sivasubramanian and Leib [21] [24] and Huang et al. [22] described a general global-iteration decoder configuration for LDPC codes, LT codes and Raptor codes, and they explained the execution of the Raptor decoder. These authors’ contributions are all on the bit correcting fountain codes. In this section, their fountain decoder configuration and the
algorithms are reviewed. The Raptor decoder will be considered in more general for both packet erasure correction and the bit error correction.

### 4.4.1 Decoding Information

With the two fountain decoders previously discussed, the message passing decoding algorithm is an application of the classic belief propagation technique. The propagation paths of the decoding messages and the update of the messages can be illustrated on the Tanner graphs of the decoding matrices. The propagated decoding information is log-likelihood ratio (LLR) [60], a kind of soft information. The message passing algorithm has been well studied for the decoding of LDPC codes [18] [19] [23] and for the decoding of LT codes [5] [13] [14].

Without losing generality, the code of interest is supposed to be binary. The binary 0 and 1 are assumed to be equally likely. In the model of packet erasure fountain codes, the packet payload can be viewed as one single bit. The decoding messages in this model is the log-likelihood ratio of a binary random variable $X \in \{0,1\}$ in $GF(2)$, stated as

$$L_X(x) = \ln \frac{P_X(x=0)}{P_X(x=1)}, \quad (4.15)$$

where $P_X(x)$ is the probability that $X$ takes on the value $x$. Apparently, within the system models described in this chapter, $P(s_i = 0) = P(s_i = 1)$. Note that $s_i$ is now equivalent to a fountain encoded bit. According to the derivation in [18], the observed log-likelihood ratio from an erasure channel for a received bit $r_i$ is given by
where \( e \) means an erasure. During the entire packet decoding procedure, the value of the decoding message remains among \(+\infty, -\infty\) and 0. For the LT code only case, hard decoding can replace the soft decoding, because erasures do not take part in the decoding and binary 0 and 1 represent the soft information \(+\infty\) and \(-\infty\).

In the model of bit correcting fountain codes, the transmitted bits are in form of BPSK, so the decoding information is the log-likelihood ratio of a binary random variable \( X \in \{\pm 1\} \) in \( GF(2) \) with +1 standing by binary 1 and –1 by binary 0, given by

\[
L_X(x) = \ln \frac{p_X(x=-1)}{p_X(x=+1)}. \tag{4.17}
\]

According to the derivation in [18], the observed log-likelihood ratio from an AWGN channel for the received bit \( r_i \) is given by

\[
L(r_i) = -4r_i/N_0
= -4(h_is_i + n_i)/N_0, \tag{4.18}
\]

where \( r_i \) is the sampled output of the receiver matched filter, \( n_i \) is the additive noise and \( N_0 \) is the single-sided power spectral density of the white Gaussian noise.

For error-free transmission, in the system model of packet erasure fountain codes, the receiver always has \( r_i = 1 \) \((r_i = 0)\) for \( s_i = 1 \) \((s_i = 0)\); in the system model of bit correcting fountain codes, \( r_i = 1 \) \((r_i = -1)\) for \( s_i = 1 \) \((s_i = 0)\).
4.4.2 Local-Iteration Decoder

A local-iteration decoder for binary Raptor codes is shown in Figure 4.6. This decoder configuration works for both packet erasure correcting and bit error correcting fountain codes.

![Raptor local-iteration decoder configuration.](image)

Since at the receiver side the fountain code is a truncated version, i.e., it is a fixed rate code at a time, suppose currently $n$ Raptor encoded bits have been collected at a user terminal to recover $k$ source bits. The corresponding decoder configuration is represented
by Figure 4.6. The pre-code is already known by the decoder, so the Tanner graph of the pre-code’s parity-check matrix $\mathbf{H}$ is usually set up in advance. The Tanner graph of the inner weakened LT code’s generator matrix $\mathbf{G}$ can be built on the fly. The received and accepted output bits $r_1, \cdots, r_n$ are connected one-to-one to the c-nodes of the LT Tanner graph. The rate $k/N$ pre-code $C$ is an LDPC code. The two columns of v-nodes $v_1, \cdots, v_N$ and $I_1, \cdots, I_N$ are the same set of intermediate bits separately for the two Tanner graphs.

With a packet erasure fountain code, $v_i$ hands in its final decoding message (some LLR) $m_i$ to $I_i$ once the LT decoding is halted by a stopping set. However, for a Raptor code with soft decoding over an AWGN channel, $v_i$ does not pass $m_i$ to $I_i$ until the maximum assigned number of LT decoding iterations are finished. The LDPC decoder takes the decoding messages from the LT decoder as observed LLRs. $c_1, \cdots, c_{N-k}$ are the c-nodes of the Tanner graph of the pre-code’s parity-check matrix. The LDPC decoding is terminated when a valid codeword results from hard decision on $I_1, \cdots, I_N$ or the maximum assigned number of LDPC decoding iterations are exhausted.

Sivasubramanian and Leib [24] introduced a concise and complete algorithm of local-iteration Raptor decoding. In the Raptor decoder, we denote a decoding message by $m$ and the decoding iteration by the superscript $l$. In the inner LT code, denote an accepted encoded bit by $r$ and its LLR value by $L$. The LLR message sent from a v-node $v$ (an intermediate bit) to an encoded bit $r$ (a c-node) is represented by $m_{v,r}$, and $m_{r,v}$ is the decoding message sent from a c-node to a v-node $v$. The message passed from the LT decoder to the LDPC decoder is denoted by $m_i$ with $1 \leq i \leq N$. In the pre-code decoder,
\( m_{l,c} \) is the message sent from the v-node \( I \) (the intermediate bit) to the c-node \( c \), and \( m_{c,l} \) is the message sent from the c-node \( c \) to the v-node \( l \).

The local-iteration decoding starts with the LT decoder, sending the messages from the v-nodes to the c-nodes and then sending the messages back to the v-nodes. At the beginning of iteration 0, all messages are initiated to zeros, except the observed log-likelihood ratios on the accepted encoded bits. At every iteration, v-nodes send messages to c-nodes and then c-nodes feedback updated messages to v-nodes. The ongoing LLR information is updated as follows [12]:

\[
m^{(l)}_{v,r} = \begin{cases} 0, & l = 0 \\ \sum_{r' \neq r} m^{(l-1)}_{r',v}, & l > 0 \end{cases}
\]

(4.19)

and

\[
m^{(l)}_{r,v} = 2 \text{tanh}^{-1} \left[ \text{tanh}(L/2) \prod_{v' \neq v} \text{tanh} \left( m^{(l)}_{v',r} / 2 \right) \right],
\]

(4.20)

where \( \text{tanh}(\cdot) \) and \( \text{tanh}^{-1}(\cdot) \) are the hyperbolic tangent function and the inverse hyperbolic tangent function, \( m^{(l-1)}_{r',v} \) is the LLR message sent to an intermediate bit at the previous iteration from all its neighboring received bits except \( r \), and \( m^{(l)}_{v',r} \) is the LLR message sent to a received bit from all its neighboring intermediate bits except \( v \). After the LT decoding is terminated by a stopping set or the assigned number \( p \) of LT iterations are finished, the final LLR message of an intermediate bit is stated as

\[
m_i = \sum_r m^{(p)}_{r,v},
\]

(4.21)

where \( r \) is the set of received bits neighboring the \( i \)-th intermediate bit.

Taking \( m_i \) as observed LLR information from the channel, the LDPC decoder works from its iteration 0. Just as the LT decoder, all messages are initiated to zeros.
except the incoming $m_i$ from the LT decoder. At every iteration, the v-nodes send messages first and then receive updated information from the c-nodes. The LDPC decoder runs with the following update rules [23]:

$$m_{l,c}^{(l)} = \begin{cases} m_i, & l = 0 \\
\sum_{c' \neq c} m_{c',l}^{(l-1)} + m_i, & l > 0 \end{cases}$$

(4.22)

and

$$m_{c,l}^{(l)} = 2\tanh^{-1}\left[\prod_{l' \neq l} \tanh\left(m_{l',c}/2\right)\right].$$

(4.23)

where $m_{c',l}^{(l-1)}$ is the LLR information sent to an intermediate bit at the previous iteration from all its neighboring c-nodes except $c$, and $m_{l',c}^{(l)}$ is the LLR information sent to a c-node from all its neighboring intermediate bits except $l$. At the end of each pre-code iteration, the decoder can make hard decision on the intermediate bits. With the sign function $\text{sgn}(\cdot)$, the hard decision on the these v-nodes at iteration $l$ is given by a column vector

$$w_i = \left[-\text{sgn}\left(\sum_c m_{c,l}^{(l)} + m_i\right) + 1\right]/2.$$  

(4.24)

Note that some bits of $w$ may be 0.5’s, because the current LLR values of these intermediate bits are still 0’s. Hence, the decoding still needs to proceed.

When all the entries of $w$ are 0’s or 1’s, the decision outcome can be verified through

$$Hw = b.$$  

(4.25)

With the packet correcting fountain code, since the log-likelihood ratio messages are $\pm \infty$ or 0, $w$ is certainly correct if $w_i \in \{0,1\}$. With a soft decoding fountain code, if $b = 0$, $w$ is a codeword and the decoding of the pre-code succeeds and ends; if $b \neq 0$, the pre-code
decoding needs to continue until the iteration threshold $q$ is reached. If the decoder cannot output a codeword at ending iteration $q$, it may declare that the decoding has failed and collect additional encoded bits to continue decoding.

4.4.3 Global-Iteration Decoder

A global-iteration decoder for binary Raptor codes is shown in Figure 4.7. This decoder configuration works for both packet erasure correcting and bit error correcting fountain codes. The global-iteration decoder is almost the same as the local-iteration decoder. The only difference is that, with the alternating global-iteration scheme, the LT decoder and the pre-code decoder hand over their updated LLR messages of the intermediate bit to each other at the end of every one of their own iterations. These exchanged messages are also viewed as observed LLR’s for the decoders.
Sivasubramanian and Leib [21] [24] and Huang et al. [22] presented the details of global-iteration decoding of Raptor codes. Again, consider a rate $k/n$ Raptor code at an arbitrary terminal with a rate $k/N$ LDPC pre-code, as shown in Figure 4.7. The mathematical denotations are the same as those used with the local-iteration, except that the message sent from $v_i$ to $I_i$ is denoted by $m_{v_i}^l$ and the message fed back from $I_i$ to $v_i$ by $m_{I_i}^l$. At the start, all messages are initiated to zeros, except the observed channel information on the collected encoded bits. The message update rules are given as follows:
\[ m_{v,r}^{(l)} = \begin{cases} 0, & l = 0 \\ \sum_{r' \neq r} m_{r',v}^{(l-1)} + m_{l,v}^{(l-1)}, & l > 0 \end{cases}, \]  
(4.26)

\[ m_{r,v}^{(l)} = 2 \tanh^{-1} \left[ \tanh(L/2) \prod_{v' \neq v} \tanh(m_{v',r}^{(l)}/2) \right], \]  
(4.27)

\[ m_{v,l}^{(l)} = \sum_r m_{r,v}^{(l)}, \]  
(4.28)

\[ m_{c,l}^{(l)} = \begin{cases} m_{v,l}^{(l)}, & l = 0 \\ \sum_{c' \neq c} m_{c',l}^{(l-1)} + m_{v,l}^{(l)}, & l > 0 \end{cases}, \]  
(4.29)

\[ m_{c,l}^{(l)} = 2 \tanh^{-1} \left[ \prod_{l' \neq l} \tanh(m_{l',c}^{(l)}/2) \right], \]  
(4.30)

and

\[ m_{i,v}^{(l)} = \sum_c m_{c,i}^{(l)}. \]  
(4.31)

In this looping process, the decoder can make hard decision on the intermediate bits at the end of the pre-code decoder:

\[ w_l = \left[ -\sgn \left( \sum_c m_{c,l}^{(l)} + m_{v,l}^{(l)} \right) + 1 \right]/2. \]  
(4.32)

Again, some entries of \( w \) may be 0.5's, because the corresponding intermediate bits have not acquired non-zero LLR values. Therefore, the decoding needs to continue until the maximum number of iterations are exhausted. The decision output can be still verified through Equation 4.25.

### 4.4.4 Reliable Bit Selection by the Receiver Pattern

Over an AGWN channel, when received bits are sufficient, the fountain decoder can discard some less reliable bits such that it can reduce decoding iterations and save space but improve performance. A threshold [57] is set to select more reliable bits for the LT or Raptor decoder. ‘More reliable’ means that the absolute value of the bit’s log-
likelihood ratio is larger. Recall the pdf of received bit in BPSK signaling with AWGN (Figure 2.2). A selecting threshold is shown in Figure 4.8. The absolute value of the threshold is usually between 0 and 1. The reason that fountain codes can choose encoded bits to decode is that the LT decoding is executed in a generator matrix, rather than a parity-check matrix. The codeword coordinates (variable nodes) are independent in a generator matrix but they are often dependent in a parity-check matrix.

Figure 4.8. Selection threshold on received bits from AWGNC.
4.4.5 Relationship between Hard Decoding and Soft Decoding

With erasure correction, hard decoding and soft decoding are same in nature. With fountain codes, hard decoding is usually used for erasure correction, but on an AWGN channel soft decoding exhibits much better performance than hard decoding for bit error correction.

Choose the LT decoder to illustrate the relationship. Recall the conventional LT code with message passing decoding. With the parallel decoding scheme, all degree-1 encoded symbols send out their then-values and restore their neighboring source symbols at the first half of every iteration. Note that this hard decoding iteration (introduced in Section 3.1.2) is different from the soft LT iteration described in Sections 4.4.2 and 4.4.3. With soft decoding, because the v-nodes $v_1, \ldots, v_N$ initially have zero LLR values only, they send zero messages (absolute uncertainties) to their neighbors $r$ and thus this first half iteration is trivial, as illustrated by Equations 4.19 and 4.26. In the second half of the first iteration, only degree-1 received bits can send non-zero messages to the v-nodes. One can verify this through Equations 4.20 and 4.27. Therefore, with hard decoding, the first iteration begins with the second half of the 0th iteration of the soft decoding. One can also see that according to Equations 4.20 and 4.27 an erasure always forces its outgoing message to zero. That is why the LT code only collects unerased symbols to set up a decoding generator matrix. The iterative propagation operation makes more and more edges acquire non-zero LLR message passing so that more and more intermediate symbols obtain updated non-zero LLR values iteration after iteration. For the erasure channel model, it would be much more convenient to apply hard decoding, since the LLR
messages propagated in the decoding graph are only \( \pm \infty \), absolute certainties. Figure 4.9 shows an LT decoding example that the path of bit value propagation with hard decoding is the same as the route of newly non-zero LLR message flow with soft decoding:

![Diagram](image-url)
Figure 4.9. Hard decoding and soft decoding of an LT code.
In every subfigure of Figure 4.9, the circles of the Tanner graph are source nodes, the squares are c-nodes corresponding to the received symbols, and the small circles at the bottom are v-nodes representing the received bits. The received bit sequence is [0 1 1 0]. The left figure column illustrates the hard decoding process and the right figure column is the soft decoding process. The hard decoding propagates bit values {0,1}. The soft decoding passes LLR messages \{0, ±∞\}, which are marked by the corresponding edge arrows. With soft decoding, the edges between s-nodes and c-nodes are zero-LLR-passing at the first half of iteration 0 but, at every upcoming half iteration, one or more new edges achieve non-zero-LLR-passing, no matter in which direction the message is passed. Apparently, the path of hard decision propagation (the blue arrows in the left figure column) is the same as the order of the newly non-zero-LLR-passing edges. With soft decoding, all the edges between s-nodes and c-nodes are non-zero-LLR-passing in both directions from the second half of iteration 3. No matter how many more iterations the decoder runs, the message in either direction on every edge remains unchanged.

### 4.5 Summary

Fountain codes with message passing decoding can work for many channel models, including packet or symbol erasure channels, AWGN channels and fading channels. Two system models of fountain codes (Raptor codes specifically) have been introduced in this chapter for packet erasure correction and bit error correction,
respectively. Raptor codes can enable fast encoding over straightforward encoding with some special design of the parity-check matrix of the pre-code. This can be done by transforming the parity-check matrix into an approximate lower-triangular form or a staircase form by row and column operations. For Raptor codes, two decoding algorithms have been described: local-iteration decoder and global-iteration decoder. Either of these decoders works for both the system models of Raptor codes. These two models are equivalent in the nature of decoding with channel information. Over an erasure channel, the fountain code decodes on a pool of error-free encoded packets or symbols so that hard decoding can be applied and it is more convenient and efficient in software. However, for bit error correction over AWGN and fading channels, the performance of hard decoding could be very poor because errors of hard decision will propagated through the whole process of decoding like a disastrous “rolling snowball”. Therefore, the fountain decoder should use soft decoding with AWGN and fading channels.
CHAPTER 5: STUDIES OF LDPC CODES

It is common to choose an LDPC code as the pre-code of a Raptor code [12] [16] [21] [22] [24]. However, a good LDPC code is not necessarily a good outer pre-code for a Raptor code, i.e., the Raptor code might not be as good as the LDPC code in terms of error rate, complexity and space consumption, or a worse LDPC code may make a Raptor work better. We know that under iterative message passing decoding, many LDPC codes and fixed rate LT codes may suffer error floors at high erasure rates or high signal-to-noise ratios. However, a Raptor code can substantially lower the error floor, compared with its own pre-coding LDPC code. One may ask, “Since the Raptor code with an ordinary pre-coding LDPC code can have such good error performance, why do we need to investigate good pre-codes?” Our answer lies in universality, complexity, and decoding overhead. One may be also interested in the aspects of the pre-code that affect the Raptor code’s performance. For instance, the performance or decoding speed can be influenced by the structure of the pre-code’s parity-check matrix or the rate of the pre-code. A very sparse parity-check matrix would be a very good choice for an LDPC code. The sparseness helps not only decrease the coding complexity and the number of stopping sets, but also possibly increase the sizes of stopping sets. It is relatively easier to control the density of a matrix because it can be determined by the distribution of row (column) weights, but it is impossible to completely remove stopping sets from a finite length practical LDPC code.

This chapter includes our algorithm that expands searching stopping sets from short to all-length LDPC codes with arbitrary structures, and our new splitting-and-filling
method to creating good irregular LDPC codes of arbitrary lengths or rates. By investigating stopping sets, we have designed simple short-length ($< 10^4$) irregular LDPC codes with low error floors.

5.1 Stopping Sets with LDPC Codes of Finite Lengths

Richardson and Urbanke [31] have proven that finite length LDPC codes can achieve excellent near capacity performance at low computation cost under iterative message passing decoding. However, an LDPC code of finite lengths, especially small dimensions, inevitably suffers an error floor [32], which is observable or non-observable [33]. In [34], Di et al. studied the average error rates of LDPC codes over binary erasure channels under message passing decoding and have found that nonempty stopping sets are the major cause of the error floor performance with LDPC codes. The nonempty stopping set is also a combinatorial property of their parity-check matrices.

A stopping set within the parity-check matrix of an LDPC code is different from a stopping set existing in the generator matrix of an LT code with hard decoding. With conventional LT codes over binary erasure channels, a stopping set is a sub-graph of the generator matrix that results from the vanishing graph at an ongoing hard decoding (edges are removed as decoding goes on). In other words, the stopping set is initially not recognized and it appears if the hard decoding is terminated without recovering all the source nodes. However, the Tanner graph of the parity-check matrix remains unchanged with the iterative decoding in the LDPC code. Shokrollahi [19] uses the Tanner graph to define a stopping set in the parity-check matrix of an LDPC code: a group of message
(variable) nodes compose a stopping set if and only if none of their neighboring check nodes have degree one, and the number of these message nodes is the size of this stopping set. Usually, a stopping set in form of Tanner graph contains both variable nodes and check nodes, as shown in Figure 5.1.

\[ \text{Figure 5.1. A stopping set of size 3 in a parity-check matrix.} \]

A nonempty stopping set is a stopping set of size larger than 0. As we know, an empty set has all the characteristics of a normal stopping set based on the stopping set definition. However, an empty set is meaningless for our research. Therefore, we usually exclude the empty set and use a strict term, "nonempty" stopping set. In LDPC codes, a stopping set is generally considered as a nonempty stopping set if not particularly described. The nonempty smallest stopping set especially plays a very important role for an LDPC code’s the error floor performance over binary erasure channels [33]. The size of a nonempty smallest stopping set is called stopping distance, or stopping number [35] [36]. The stopping distance is approximately linear to the block length of the LDPC code [35], no matter it is a regular code or an irregular code with given degree distribution pair.
Searching the smallest stopping set in an LDPC code has attracted our interest, which would be helpful for designing and analyzing a specific good code. Some literature [37]–[39] presents helpful investigations on the relationship between the performance of the LDPC code ensembles and the stopping set distributions, but the authors did not propose any specific approach for searching the stopping distance for arbitrary LDPC codes. The reason could be the inherent NP-hardness of this searching. Wang et al. [40] made further contribution by proposing an efficient algorithm of searching stopping sets and trapping sets for any arbitrary very short-length (≤ 500) LDPC code by analyzing its upper bound. However, this algorithm is not practical enough because many code lengths in applications are greater than 500. Other studies [41]–[43] introduced some algorithms of searching or analyzing stopping sets for an individual code using the combinatorial or algebraic method, but these approaches are all based on certain limited assumptions or applications so they are not suitable for general LDPC code designing. This has led us to research on a new approach to searching stopping distance and enumerating all the smallest stopping sets for any LDPC code with finite length by means of a tree-like searching algorithm based on set diagrams.

5.2 A New Approach to Searching the Smallest Stopping Sets in LDPC Codes

5.2.1 Assumptions and Definitions

First, make an important assumption: the girth of the LDPC code is at least 6. This is because we need to avoid girth 4 for the parity-check matrix of an LDPC code.
We used set diagrams to define a cycle and a stopping set in [30], which is helpful to investigate the searching of stopping sets.

**Cycle:** In a variable (check) node set diagram, a cycle is an enclosed loop formed by a group of sets, among which any pair of sets must share and only share one common element.

**Stopping set:** In a variable node set diagram, \( d \) \((d > 1)\) variable node sets compose a stopping set of size \( d \) if any element contained in the union of these \( d \) variable node sets is an element of the intersection of at least two of the \( d \) variable node sets. \( d \) is the stopping distance or stopping number of this set diagram.

Here is an example from [30]. A parity-check matrix of block length 9 is given by

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1
\end{bmatrix}
\]

\(c_1, c_2, c_3, c_4, c_5\).  

(5.1)

Clearly, Equation 5.1 defines a regular LDPC code (the \( v \)-nodes are of degree 2 and the degree of the \( c \)-nodes is 4). The set diagrams of this code are illustrated in Figure 5.2.
In this LDPC code, with the assumption and the definition of cycle, we see that a cycle of length $2d$ is exactly $d$ different linking variable (check) node sets containing $d$ different elements and each element is shared by four variable node sets (two check node sets), so it is sufficient to form a cycle with the $d$ variable (check) nodes. The definition of stopping set indicates that any element in the union of the $d$ variable node sets is contained by at least two of these $d$ variable node sets. In other words, there are $d$ different variable node sets and at least $d$ different elements each of which is shared by any two of the variable node sets in a stopping set with size $d$. Note that, only when all these $d$ different variable nodes are of degree 2, the number of different common elements is $d$. We can make the same conclusion as [38]: there must be at least one cycle in a stopping set. Therefore, avoiding girth 4 guarantees that the smallest stopping set size is not smaller than 3.
Next make another assumption that in an LDPC code the degrees of all check nodes are no less than 2. The definition of stopping set can explain the dominant effect of the stopping set on the LDPC code’s error floor performance with iterative message passing decoding when data are transmitted over a binary erasure channel [33]. Denote a nonempty stopping set by $S$. At the receiver over a BEC, we suppose that all the variable nodes contained in $S$ are a subset of the erased variable nodes. The decoder will stop its iterative decoding procedure when all transmitted bits are restored or the maximum number of decoding iterations are finished (no matter the decoding succeeds or not). Because all elements (check nodes) of the variable sets in $S$ are covered by at least two variable node sets, all the decoding messages they receive from their neighboring variable nodes are uncertainties and the values of these elements can never be identified. Therefore, the returning messages from the check nodes will be always uncertain and the variable nodes in $S$ cannot be recovered so the error floor will appear. Reference [34] provides the complete proof of the failure of iterative decoder on the stopping sets.

5.2.2 The Searching Algorithm

Based on the two definitions and the two assumptions, we introduced a new algorithm [30] of enumerating the smallest stopping sets and finding the stopping distance for any individual LDPC code with stopping distance less than 8. This algorithm conducts tree-searching in the set diagram of the parity-check matrix of an LDPC. The searching diagram is rooted on a single variable node and evolves from variable node to variable node connected by edges. Each edge is labeled with a group of check nodes
which are connected to a possible stopping set. The path (searching branch) represents a stopping set if it is a successful path. In the searching diagram, a variable node in searching step \( j \) is called a state at level \( j \), so the root \((0^{th} \text{ level})\) of the diagram is the initial state that is the start of searching. Just for convenience, the terms state \( (s_j) \), variable node \((v_i)\), and vertex are used interchangeably for this algorithm (where \( i \) is the variable node index). Similar to a trellis diagram, this searching diagram is driven by two functions: the output function and the searching state transition function.

The output function is stated as

\[
O_j = \begin{cases} 
  f_j(s_j), & j = 0 \\
  f_j(s_j, I_{j-1}), & j > 0
\end{cases}
\]  

where \( O_j \) includes \( I_j \) that is the searching path so far at level \( j \). And the searching state transition function is stated as

\[
s_j = \begin{cases} 
  v_i, & j = 0 \\
  g_j(s_{j-1}, I_{j-1}), & j > 0
\end{cases}
\]  

The output \( O_j \) is the union of \( I_j \) and the neighbors of vertex \( i \) \((v_i)\). Among the inputs of the output function, \( s_j \) is a given variable node at the current level \( j \) and \( I_{j-1} \) is the combination of check nodes produced at the previous level. \( I_{j-1} \) consists of one special check node and multiple general check nodes. The special check node included in \( I_{j-1} \) is the check node that is used to derive a current state \( s_j \) and the others are general check nodes. At level \( j \), if a check node is covered by both \( I_j \) and the neighbor set of the state \( s_j \), this check node needs to be boxed. So a boxed check node has been visited twice or more, i.e., it has at least two neighbors.
In order to reduce the computation complexity, repeated searching paths should be marked with termination in the searching diagram. The rules of stopping a searching path are as follows:

*Rule 1:* If a vertex in a state \( x \) is the same as another vertex in a previously computed state \( y \) and these two vertices have resulted in the same searching path, the path traveling from the initial vertex to this vertex is duplicate or invalid so use the failure flag \( F \) to terminate this searching branch. \( x \) and \( y \) are unnecessarily at the same searching level.

*Rule 2:* If all the check nodes in \( I_j \) output by a vertex are boxed, the searching from the initial vertex to this vertex is successful and this set of variable nodes comprises a stopping set. This successful searching branch is marked with a \( T \) flag as the final state and simultaneously all the other branches including any of the vertices in the successful branch are set as useless, also marked by the failure flag \( F \) and terminated.

With the searching state transition function, a current state \( s_{j-1} \) and a special check node generate the next state \( s_j \) according to the check node set diagram. The next state \( s_j \) is actually any of the neighbors of the special check node except the current state \( s_{j-1} \).

As an example, in Figure 5.3 is shown the searching of stopping sets covering the variable node \( v_1 \) in the LDPC code represented by the parity-check matrix given in Equation 5.1. The process starts from \( v_1 \) which has two neighbors (\( c_1 \) and \( c_2 \)) and hence derives two searching branches at level 0. On both the branches, “\( c_1, c_2 \)” is the resulted check node group. On the upper branch, \( c_1 \) is the special check node marked with a hat, i.e., the searched vertices at next level will be among the neighbors of \( c_1 \), and then \( c_2 \) is a
general check node. On the other hand, $c_2$ is the special check node and $c_1$ is a general check node on the lower branch. The upper branch goes on to the next states, the beginning of level 1, which are the neighbors of $c_1$ except $v_1$: $v_2$, $v_3$ and $v_4$. For the output function on $v_2$, the inputs are $v_2$ ($s_1$) and “$c_1$, $c_2$” ($I_0$), so the output $O_1$ will be the union of $I_0$ and all the neighbor of $v_2$, that is, $I_1 = \{c_1, c_2, c_3\}$. Because now $c_1$ has been covered by $I_0$ and the neighbors of $v_2$, $c_1$ is framed with a green box. At the same time, $c_3$ becomes the special check node to derive searching at the next level. Hence, $v_5$, $v_8$ and $v_9$ are the states (vertices) at level 2. For $v_5$, its output function produces $I_2 = \{c_1, c_2, c_3\}$ and thus all these check nodes are boxed. At this point, a stopping set has been found and it is the union of $v_1$, $v_2$ and $v_5$ which is also the successful searching path “$v_1$, $v_2$, $v_5$”. For $v_8$, the output includes $I_2 = \{c_1, c_2, c_3, c_4\}$, but only $c_1$ and $c_3$ are boxed and $c_4$ is the special check node to continue the searching. Similarly, $v_9$ outputs $I_2 = \{c_1, c_2, c_3, c_5\}$ and it has not existed in any of the previously computed paths, so it still needs to continue searching because only $c_1$ and $c_3$ are boxed. Go back to level 1. Both $v_3$ and $v_4$ derive new searching paths but only $c_1$ is boxed, so these two branches go on to evolve new states for the next level. At level 2, $v_6$ and $v_8$ output different valid paths. However, $v_8$, $v_9$ and $v_{10}$ all appear twice at level 2, so the latter path of each one fails and will be terminated. Similarly, for the lower branch at level 0, all its deduced branches are not successful because the states at level 1 have already existed in the evolution of the upper branch.
Figure 5.3. Diagram of searching stopping sets.
For enumerating the smallest stopping sets only, with every single searching tree, the searching process will not continue to the next level if any stopping set has been found at the current level. Once all the variable nodes are applied with this searching algorithm to finish searching, all the different smallest stopping sets can be found so the stopping distance of a given code is confirmed. For the above example LDPC code, all the smallest stopping sets are \{v_1, v_2, v_5\}, \{v_1, v_3, v_6\}, \{v_1, v_4, v_7\}, \{v_2, v_3, v_8\}, \{v_2, v_4, v_9\} and \{v_3, v_4, v_{10}\}. These stopping sets were enumerated in 0.328 seconds on a 2.2-GHz duo T7500 CPU IBM computer with 2GB RAM [30]. In our simulation, the algorithm was applied to two randomly generated LDPC codes of block lengths 1024 and 2304. It spent 8380.328 seconds and 29615.21 seconds, respectively, to enumerate all the smallest stopping sets of size 5, respectively. In [30] we provide a C++ implementation of this searching algorithm at: [http://oak.cats.ohiou.edu/~hl119803/news.php](http://oak.cats.ohiou.edu/~hl119803/news.php).

5.2.3 Complexity and Space

We analyzed the computational complexity in [30], which is even suitable for irregular LDPC codes that have check nodes of degree 1. With this algorithm, totally \(n\) searching trees are created to enumerate all the smallest stopping sets, where \(n\) is the number of variable node in an LDPC code of interest. On each tree, the computation complexity is mainly caused by the number of successful branches and the size of the shortest paths. The number of valid branches can be estimated with the variable node degree distribution and the check node degree distribution. In fact, the number of valid branches can be bound by \([c_{avg} - 1](v_{avg} - 1)^{s_d}\), \(c_{avg}\) is the average degree of check
node, $v_{avg}$ is the average degree of variable node and $s_d$ is the stopping distance of the LDPC code. For either regular or irregular LDPC codes of finite length $n$, Orlitsky et al. proved that $s_d$ approximates to $\log(n)$. Give the degree distribution of variable node

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

(5.4)

and the degree distribution of check node

$$\rho(x) = \sum_{i=1}^{n-k} \rho_i x^i,$$

(5.5)

where $\lambda_i$ ($\rho_i$) is the probability that a variable (check) node takes degree $i$ and $k$ is the code dimension. The average degree of variable node and the average degree of check node can be obtained as follows

$$v_{avg} = \lambda'(x)$$

$$= \sum_{i=1}^{n} i \lambda_i x^{i-1}$$

(5.6)

and

$$c_{avg} = \rho'(x)$$

$$= \sum_{i=1}^{n-k} i \rho_i x^{i-1}.$$ $$= \sum_{i=1}^{n} i \rho_i x^{i-1}.$$ (5.7)

We also have

$$v_{avg} = (1 - R) c_{avg},$$

(5.8)

where $R$ is the code rate.

The time complexity of searching all the smallest stopping sets can be on the order of $[(c_{avg} - 1)(v_{avg} - 1)]^{s_d} \cdot n$ and the worst case could be $[(c_{avg} - 1)(v_{avg} - 1)]^{\log(n)} \cdot n$. At this point, we are interested in the dominant cause of the complexity. According to the worst case, the time cost mainly depends on the stopping distance and degrees of variable and check nodes in the LDPC code, but not directly on the density of
parity check matrix. In some applications, $c_{avg}$ is around 6 and $v_{avg}$ is about 3. Therefore, if the stopping distance is no greater than 8, the enumerating process is manageable and the searching complexity is much more favorable than the brute-force searching which can be exhaustive. Although our algorithm is limited by the stopping distance of size 8, it is much more feasible with any finite code lengths than Wang’s algorithm [40] and the complexity can be further reduced if Rule 1 and Rule 2 are applied on the searching tree.

Finally, space consumption needs to be addressed since enumerating the smallest stopping sets requires storing the smallest stopping sets and the branches found in the current state. Because the number of the smallest stopping sets is usually much smaller than the number of the branches in the current state, the consumed storage for the valid branches of each variable node dominates the required space for the searching process. Thus, it is suitable to use $O\left(\left(v_{avg} - 1\right)\left(c_{avg} - 1\right)^{v_{avg} - 1}\right)$ to describe the growth of space consumption.

5.3 A Splitting-and-filling Technique to Constructing Good Irregular LDPC Codes

5.3.1 Background

This subsection describes our motivation to inventing a new method of constructing good irregular LDPC codes. With density evolution, Richardson et al. [44] showed that, with carefully chosen v-node and c-node degree distribution pairs, randomly constructed irregular LDPC codes could achieve better near-capacity performance at rates close to the channel capacity than regular LDPC codes if decoded using the iterative
message passing algorithm. However, MacKay et al. [45] found that regular LDPC codes outweigh the irregular codes in error floor performance (at moderate or high signal-to-noise ratios). The major cause of higher error floors can be the unfavorable combinatorial characteristics, such as the smallest cycles, stopping sets or trapping sets [31] [44] [45].

Many researchers have proposed strategies and methods to improve the error floor performance of irregular LDPC codes. These ideas include avoiding small girths in codes, but a large girth is only a sufficient but not necessary condition for the low error floor performance of an LDPC code, because not all small cycles degrade the performance of the iterative decoder equally [46]. Small stopping sets, especially the smallest stopping sets, have been proven to be the main cause of high error floor of an LDPC code [34]. Over a binary erasure channel, if some stopping sets are filled with lost symbols, the code would produce an error floor. Richardson [33] raised the concept of trapping set extended the investigation from the binary erasure channel to memoryless channels. Tian et al. [38] made a further step with detailed explanation on the relationship between cycle sets, stopping sets and the small distances of LDPC codes, and they also proposed a direct algorithm of generating irregular LDPC codes with low error floors. However, this method requires complicated computer searching techniques. Therefore, we have looked for a direct searching strategy of constructing good irregular LDPC codes.

We presented in [46] a new method of generating an irregular LDPC code with low error floor based on set diagrams and Euclidean geometry. Euclidean geometry was not new in constructing irregular LDPC codes. Under the help of decomposition and
mask techniques, Xu et al. [47] designed irregular LDPC codes with given degree distribution pairs based on Euclidean geometry, which have girths of at least six. But our method is good to generate an irregular LDPC code of arbitrary length and code rate, and it can purposely eliminate some bad cycles or stopping sets of the LDPC code, ensuring girth of at least six, and therefore improve its error floor performance. Most importantly, our method has no need to operate row permutation on the parity-check matrix which is necessary with the masking technique in [47].

5.3.2 The Splitting-and-Filling Technique on Set Diagrams

Row splitting and column splitting techniques were introduced by Kou et al. [48] to improve the error performance of an LDPC code. Their method, based on the matrix representation, can lower the density of the parity-check matrix and break some potentially bad cycles. With rotational or/and random splitting, an irregular parity-check matrix can be constructed by applying the row splitting or column splitting technique on a regular or irregular matrix, but it is still difficult to generate an irregular code with a given degree distribution pair. Therefore, we proposed a more functional and flexible technique, splitting-and-filling (SF), base on set diagrams.

The basic idea of our splitting-and-filling method [46] is as follows: with a variable (check) set diagram, first split a variable (check) node set $S$ into a desired number of variable (check) node sets, and then fill the newly generated variable (check) node sets with the elements of the original set $S$ to make them obey a given degree distribution and favorite combinatorial properties. In the filling stage, it is unnecessary to
use up all the elements in the original node set $S$. If all these elements are assigned to the new node sets, the procedure is called full splitting-and-filling. Otherwise, it is called partial splitting-and-filling. The result of full splitting-and-filling is analogous to that of the splitting method in [48]. An example of partial splitting-and-filling is illustrated in Figure 5.4. The variable node set $v_i$ is split into two variable node sets, $v_{i,1}$ and $v_{i,2}$, but the element $c_g$ is removed from the filling.

![Diagram showing partial splitting-and-filling on a variable node set.](image)

**Figure 5.4.** Partial splitting-and-filling on a variable node set.

### 5.3.3 Construction of Irregular LDPC Codes with Good Combinatorial Properties

For applying our splitting-and-filling technique, it first needs to build up a raw set diagram based on Euclidean geometries. Then, with a wisely designed degree distribution pair, an irregular LDPC code is obtained after applying the splitting-and-filling technique to the raw set diagram with some strategies. By purposely filling in the generating
process, a specific and good combinatorial characteristic of the code can be generated. In this subsection, we introduce the method of generating a raw set diagram based Euclidean geometries and then refining the raw set diagram to create a good irregular LDPC code.

Given a code block length $n$ and a degree distribution pair, the number of nonzero entries in the irregular parity-check matrix can be calculated. Recall the polynomials for the variable node and check node distributions, Equations 5.4 and 5.5. Reference [44] introduced an equation to compute the number of 1’s in the parity-check matrix:

$$E = n \cdot \frac{1}{\int_0^1 \lambda(x)dx} = n \cdot \frac{1}{\int_0^1 \lambda(x)dx}.$$  

(5.9)

Here we present an approach, similar to the one in [48], to establishing a raw set diagram. With this approach, it is necessary to know parameters $E$ and $n$ for choosing a proper $m$-dimension Euclidean Geometry over $GF(2^s)$, denoted by $EG(m, 2^s)$. An $m$-dimension Euclidean Geometry over $GF(2^s)$ can be represented by the $2^{ms}$ $m$-tuples

$$V = (b_0, b_1, \cdots, b_{m-1}),$$  

(5.10)

where $b_0, b_1, \cdots, b_{m-1}$ are elements of $GF(2^s)$. Since an $m$-tuple is a point in the Euclidian geometry $EG(m, 2^s)$, a ‘straight’ line that passes through a point $V_0$ can be expressed by

$$\{V_0 + \beta V : \beta \in GF(2^s)\},$$  

(5.11)

where $V_0$ and $V$ are linearly independent in $EG(m, 2^s)$ and $V \neq 0$. Clearly, any two different lines share at most one common point and any two points are connected by one single line. There are

$$a = \frac{2^{ms}-1}{2^s-1}.$$  

(5.12)
lines that pass through each point and each line has \(2^s\) points in \(EG(m, 2^s)\). Now the variable node and check node set diagrams can be mapped into \(EG(m, 2^s)\). Let us view each point as a variable node set and the lines intersecting at this point as its elements (neighboring check nodes). Similarly, each line is treated as a check node set and the points on the line are its elements. Thus, a raw regular set diagram can be produced. To make the raw set diagram feasible with our splitting-and-filling technique, we chose the parameters \(m\) and \(s\) very carefully. In general, the following relations should be satisfied:

\[
2^s \cdot \frac{2^{ms} - 1}{2^{s-1}} \geq 2^E \tag{5.13}
\]

and

\[
2^{ms} < n, \tag{5.14}
\]

where \(n\) is the code block length.

In the next step, we want to generate an irregular LDPC code with a given degree distribution pair. We have created a raw regular set diagram in the previous step and we can change the raw set diagrams into an irregular LDPC code with a desired degree distribution pair. The three steps of refining a raw set diagram are as follows [46].

In the first step, the number of variable (check) nodes with each assigned degree needs to be computed. Reference [44] provides equations of calculating the number of variable nodes of degree \(i\) and the number of check nodes of degree \(k\):

\[
N_{v,i} = n \cdot \frac{\lambda_i / i}{\int_0^1 \lambda(x)dx} \tag{5.15}
\]

and

\[
N_{c,k} = n \cdot \frac{\rho_k / k}{\int_0^1 \rho(x)dx} \tag{5.16}
\]
In the second step, we apply the partial splitting-and-filling technique to every node set in either of the raw diagrams (variable node set diagram and check node set diagram). With proper reforming of the node sets, it intends to eliminate all the redundant nonzero entries in the corresponding parity-check matrix and to obey the pair of degree distributions. After these operations, a new diagram $S_{G1}$ is generated and all sets in $S_{G1}$ should meet the desired degrees. The third step adopts the full splitting-and-filling operation. In this step, another diagram, $S_{G2}$, corresponding to $S_{G1}$ is refined to make its sets have desired degrees and still keep all the degrees of the sets in $S_{G1}$ unchanged.

**Refine Strategy** [46]: The set diagram to be refined in step 2 should have the smaller spread of the degree distribution distances.

The purpose of the refine strategy is that the code has greater diversity of node degrees so that it is more flexible to apply the full splitting-and-filling operation to $S_{G2}$. For an irregular code, it has been proven that, to improve near capacity performance, the variable nodes have high degrees but check nodes have low degrees as far as possible [8]. However, in every single LDPC code, the total degrees of all the variable nodes are exactly equal to the total degrees of all the check nodes. It is inevitable that some variable nodes are of low degrees or some check nodes have high degrees. In order to keep the balance, Richardson *et al.* [44] designed finite length irregular codes that had some variable nodes of very low degrees, such as degree 2 and degree 3 and resulted in large spread of the variable node degree distances. Therefore, we suggest to first doing partial splitting-and-filling operations on the raw check node set diagram first.
In step 3, due to the large spread of the degree distribution distances, the refining process is flexible, but it is still difficult to realize the refining by directly applying the full splitting-and-filling to $S_{G2}$ at a time. In order to accomplish step 3, full splitting-and-filling are executed for two rounds. In the first round, we should check whether the degree of each node set in $S_{G2}$ can be expressed as the sum of some or all desired degrees. If not, return to step 1 and regenerate $S_{G1}$. If yes, adopt an intermediate degree $d_i$, which is the least common multiple of two smallest degrees. Then the full splitting-and-filling technique is applied to each node set in $S_{G2}$. In the generating process, the sets with higher degrees should be generated earlier. In the second round, it splits the sets with degree $d_i$ and makes all the sets have the right degrees to the best of its ability.

In the final step, a set diagram $S_G$ with a given degree distribution can emerge after we apply the splitting-and-filling technique to each node set of $S_{G2}$. If the raw set diagram has girth of at least 6, the set diagram $S_G$ still has girth of at least 6 because the splitting-and-filling technique does not change the row/column of the variable/check nodes in the parity-check matrix. Once we can enumerate the newly generated sets covered by the specific combinatorial characteristic, according to design requirement, the elements can be assigned to the newly generated sets in the filling process. Therefore, the code has a lower error floor with the combinatorial characteristic of its final set diagram optimized.
5.4 Performances of Our Left-regular LDPC Codes under MP decoding over BEC and AWGNC

Left-regular (LR) LDPC codes are actually irregular codes, but their left nodes (variable nodes) have constant degrees. High rate right-Poisson (RP), left-regular LDPC codes have been reported to be good pre-codes for Raptor codes [12] [21] [24] [28]. In this section, some irregular LDPC codes are generated and we use simulation to verify their performances over the binary erasure channel and the additive white Gaussian noise channel. Our irregular codes are all left-regular LDPC codes because their variable nodes (left nodes) are all of degree 4. The error performances of these left-regular LDPC codes under iterative decoding are presented.

Consider a rate $k/n$ binary LDPC code with code dimension $k$. Every group of $k$ input source bits is one data frame. Multiple LDPC codes were simulated over BEC and AWGNC. Their bit error rates and frame error rates were calculated.

The first left-regular code is denoted by SF-1/2-1152: a rate 1/2 LDPC code with $k = 1152$, generated with our splitting-and-filling technique. The size of its parity-check matrix is 1153-by-2304. The number of checks is not 1152 just because the degree of each left node is even and after bitwise row operations the row echelon form (REF) of the parity-check matrix has one all-zero row on the bottom. In order to make the code have full rank which is equal to its dimension $k$, the number of parity checks needs to be 1153.

The second code is denoted by SF-5/6-1080: a rate 5/6 LDPC code whose dimension is 1080, generated with our splitting-and-filling technique. It has a 217-by-1296 parity-check matrix.
The third code is a rate 1/2 left-regular LDPC with $k = 1152$. The degrees of all left nodes are 4 and the four neighbors of each variable node are randomly and uniformly located. The size of its parity-check matrix is 1153-by-2304. This code is called LR-1/2-1152.

The fourth code is a rate 5/6 left-regular LDPC with $k = 1080$. The degrees of all left nodes are 4 and the four neighbors of each variable node are randomly and uniformly located. The size of its parity-check matrix is 217-by-1296. This code is called LR-1/2-1080.

The error performances of these codes with message passing decoding on BEC and AWGNC are illustrated in Figure 5.5 through Figure 5.8, respectively. Note that the frame error rate is equal to the decoding failure rate (DFR). All the four codes can achieve near-capacity performance over BEC and AWGNC. In the simulation ranges, the codes do not exhibit any error floors. With the same code rate, the randomly generated LDPC code has error performances analogous to those of the code created with our splitting-and-filling technique, for the random LR code also has no girth 4. The reason lies in the sparseness of these left-regular codes. All the four codes are very sparse: 0.347% for the rate 1/2 codes, and 1.8% for the rate 5/6. For left-regular codes, as the numbers of rows and columns increase, the probability that girth 4 appears in the parity-check matrix decrease. That means, it is easy to generate a left-regular LDPC codes with girth no smaller than 6, once the dimension of the parity-check matrix is much larger than the constant left degree. The complexity of producing such a random code can be much lower than using our splitting-and-filling technique on the same size code. The splitting-
and-filling is still advantageous if neither of the left and right degree distributions is regular.

Figure 5.5. BERs of left-regular LDPC codes under MP decoding over BEC.
Figure 5.6. FERs of left-regular LDPC codes under MP decoding over BEC.
Figure 5.7. BERs of left-regular LDPC codes under MP decoding over AWGNC.
5.5 LDPC Codes in the IEEE 802.16e Standard

5.5.1 Background

Developed from the IEEE 802.16 standard, the 802.16e standard is supposed to meet the expanding demand for mobility services [27]. One typical example is that it provides wireless broadband access to fast automobiles on highway. WiMAX (Worldwide Interoperability for Microwave Access) is a protocol, based on the 802.16 standard, which realizes long-distance data transmission for both fixed and mobile Internet access. The standard uses LDPC codes as error correcting codes. The 802.16e LDPC codes are

Figure 5.8. FERs of left-regular LDPC codes under MP decoding over AWGNC.
free and open to the public. They have been proven to be very good error correcting
codes. The details of these codes are introduced in this section.

**5.5.2 Parity-Check Matrix Construction**

The LDPC codes in the IEEE 802.16e standard are all represented by their own
parity-check matrices. For these parity-check matrices, there are in total six base matrices
so the 802.16e LDPC codes have strict structures and dimensionalities. Suppose the
parity-check matrix $H$ is of size $m \times n$, where $m$ is the number of parity-checks and $n$ is
the number of codeword coordinates. Thus, the code dimension (the number of source
symbols) is $k = n - m$. The parity-check matrix is the combination of multiple $z \times z$
submatrices. Each submatrix is either a permutation of an identity matrix or an all zeroes
matrix so that a base matrix just needs to be a set of numbers that specify individual right
circular shifts for the permutations. The size of the base matrix is $m_b \times n_b$, where
$m = zm_b$ and $n = zn_b$ (so we can define $k = zk_b$). The entries of the base matrix are
real integers no smaller than $-1$. A parity-check matrix is constructed by replacing each
entry of the base matrix with a $z \times z$ submatrix: each $-1$ with an all zeros matrix, and
each non-negative number with a circularly right shifted identity matrix. Therefore, given
the code dimension $k$ (or the block length $n$) and a base matrix, an LDPC code is
determined.

The base matrix is denoted by $H_{bm}$. In [27], the six base matrices are given for
particular code rates and different subtypes:
Rate 1/2:

\[
\begin{bmatrix}
-1 & 27 & -1 & -1 & 22 & 79 & 9 & -1 & -1 & -1 & 12 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 24 & 22 & 81 & -1 & 33 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & 47 & -1 & -1 & -1 & -1 & -1 & 65 & 25 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 39 & -1 & -1 & 84 & -1 & -1 & 41 & 72 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 46 & 40 & -1 & 82 & -1 & -1 & 79 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 95 & 53 & -1 & -1 & -1 & -1 & 14 & 18 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 11 & 73 & -1 & -1 & -1 & 2 & -1 & -1 & 47 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
12 & -1 & -1 & -1 & 83 & 24 & -1 & 43 & -1 & -1 & -1 & 51 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & -1 & -1 & 94 & -1 & 59 & -1 & -1 & 70 & 72 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
-1 & -1 & 7 & 65 & -1 & -1 & -1 & 39 & 49 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
45 & -1 & -1 & -1 & -1 & 66 & -1 & -1 & -1 & 26 & 7 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & -1 & -1 \\
\end{bmatrix}
\]

Rate 2/3 A:

\[
\begin{bmatrix}
3 & 0 & -1 & -1 & 2 & 0 & -1 & 3 & 7 & -1 & 1 & 1 & -1 & -1 & -1 & 1 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & 36 & -1 & -1 & 14 & 10 & -1 & -1 & 18 & 2 & 1 & 3 & 0 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 12 & 2 & -1 & 15 & 40 & -1 & 3 & -1 & 15 & -1 & 2 & 13 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 19 & 24 & -1 & 3 & 0 & 1 & 6 & -1 & 17 & -1 & -1 & 18 & 39 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
20 & -1 & -1 & 6 & -1 & -1 & 10 & 29 & -1 & -1 & 28 & -1 & 14 & -1 & 38 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 10 & -1 & 28 & 20 & -1 & -1 & 8 & -1 & 36 & -1 & 9 & -1 & 21 & 45 & -1 & -1 & 0 & 0 & -1 & -1 \\
\end{bmatrix}
\]

Rate 2/3 B:

\[
\begin{bmatrix}
-1 & 69 & -1 & 88 & -1 & 33 & -1 & 3 & -1 & 16 & -1 & 37 & -1 & 40 & -1 & 48 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
10 & -1 & 86 & -1 & 62 & -1 & 28 & -1 & 85 & -1 & 16 & -1 & 34 & -1 & 73 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & -1 & 32 & -1 & 81 & -1 & 27 & -1 & 88 & -1 & 5 & -1 & 56 & -1 & 37 & -1 & -1 & 0 & 0 & -1 & -1 & -1 & -1 \\
-1 & 30 & -1 & 65 & -1 & 54 & -1 & 14 & -1 & 0 & -1 & 30 & -1 & 74 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & -1 \\
32 & -1 & 0 & -1 & 15 & -1 & 56 & -1 & 85 & -1 & 5 & -1 & 6 & -1 & 52 & -1 & 0 & -1 & -1 & -1 & -1 & 0 & 0 \\
-1 & 0 & -1 & 47 & -1 & 13 & -1 & 61 & -1 & 84 & -1 & 55 & -1 & 78 & -1 & 41 & 95 & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{bmatrix}
\]

Rate 3/4 A:

\[
\begin{bmatrix}
6 & 38 & 3 & 93 & -1 & -1 & -1 & 30 & 70 & -1 & 86 & -1 & 37 & 38 & 4 & 11 & -1 & 46 & 48 & 0 & -1 & -1 & -1 & -1 \\
71 & -1 & 55 & -1 & 12 & 66 & 45 & 79 & -1 & 78 & -1 & -1 & 10 & -1 & 22 & 55 & 70 & 82 & -1 & -1 & 0 & 0 & -1 & -1 & -1 \\
38 & 61 & -1 & 66 & 9 & 73 & 47 & 64 & -1 & 39 & 61 & 43 & -1 & -1 & -1 & -1 & 95 & 32 & 0 & -1 & -1 & 0 & 0 & -1 \\
-1 & -1 & -1 & -1 & 32 & 52 & 55 & 80 & 95 & 22 & 6 & 51 & 24 & 90 & 44 & 20 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\
\end{bmatrix}
\]
5.5.3 Fast Encoding

Apparently, any parity-check matrix of an LDPC code in the IEEE 802.16e standard is in an approximate lower-triangular form without any row or column operations, so fast encoding can be enabled directly. Recall Figure 4.5 and Equation 4.4. For any 802.16e $H$ matrix, one always has $N = n$ and $g = z$. With Equations 4.5 through 4.12, the encoding can be very fast. If $z = 1$, the $H$ matrix is a right-hand side staircase matrix, as is in Equation 4.13, so linear encoding can be achieved with Equation 4.14.

5.5.4 Error Performance under Iterative decoding over BEC and AWGNC

In this subsection, we present simulation results of LDPC codes constructed with the six base matrices in the IEEE 802.16e standard. In our simulation, the maximum block length is chosen: $n = 2304$. The error performances of these codes on BEC and AWGNC are illustrated in Figure 5.9 through Figure 5.12. All the six codes achieve near capacity performance over both BEC and AWGNC. The performances of the rate 1/2 and
5/6 codes are better than our rate 1/2 and 5/6 left-regular codes. No error floors occur in the simulation range. These six codes are very sparse:

<table>
<thead>
<tr>
<th>Rate</th>
<th>1/2</th>
<th>2/3A</th>
<th>2/3B</th>
<th>3/4A</th>
<th>3/4B</th>
<th>5/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparseness</td>
<td>0.275%</td>
<td>0.434%</td>
<td>0.439%</td>
<td>0.615%</td>
<td>0.637%</td>
<td>0.868%</td>
</tr>
</tbody>
</table>

So they can assure low decoding complexity using iterative message passing algorithm.

*Figure 5.9.* BERs of 802.16e LDPC codes under MP decoding over BEC.
Figure 5.10. FERs of 802.16e LDPC codes under MP decoding over BEC.
Figure 5.11. BERs of 802.16e LDPC codes under MP decoding over AWGNC.
In this chapter, discussions are presented on some LDPC codes that are going to the pre-codes of Raptor codes. We avoid girth 4 in LDPC codes, so we have invented a low cost method to checking the girth length of any arbitrary parity-check matrix and invented a splitting-and-filling technique to generating an LDPC codes with girth at least 6, given a degree distribution pair. The splitting-and-filling method is useful for any given code size, but it is unnecessary for all cases. We have found that our left-regular LDPC codes with left degrees randomly and uniformly chosen can avoid length-4 girth with very high probability, once the number of checks is much greater than the constant.

Figure 5.12. FERs of 802.16e LDPC codes under MP decoding over AWGNC.

5.6 Summary
left degree. Such a left-regular code is easy to generate and it is of very low cost. The LDPC codes in the IEEE 802.16e standard are free to the public and they have very good near capacity performances with low error floors.

Among these good LDPC codes, lower code rates usually bring better error performance but it does not mean that the lower rate LDPC codes are the better pre-codes for Raptor codes. The following chapters will show evidences.
CHAPTER 6: MAXIMUM LIKELIHOOD DECODING WITH ERASURES

Maximum likelihood (ML) decoding algorithms have been well used for linear block codes with applications over noisy channels. As we know, the conventional LT codes and Raptor codes execute belief propagation decoding, but we also learned that the maximum likelihood decoding had been applied to Raptor codes [51]–[53] over binary erasure channels. With BEC, maximum likelihood decoding can outperform message passing decoding by reducing decoding overhead, though it usually can cause high computation complexity. In this chapter, studies are introduced on fountain codes under maximum likelihood decoding over the binary erasure channel model.

As rateless erasure correcting codes, fountain codes under message passing decoding can realize efficient and fast data recovery in multicast and broadcast applications if receivers collect enough numbers of reliable data packets. However, the message passing decoding would fail once the decoder cannot own sufficient encoded packets. Some environments can cause such problem: for example, extremely high data erasure rate, loss of communication, and limited data space at receiver. Without enough encoded packets, the belief propagation decoding process would be defeated by a stopping set, but there is still hope for successful data recovery if the code’s decoding matrix has rank equal to the number of source packets or symbols. The help can be maximum likelihood algorithm. We also bring forth the hybrid message passing and maximum likelihood (MP/ML) decoding algorithm for LT codes. In order to raise the decoding speed further, we have adopted a fast maximum likelihood algorithm. Hence, the maximum likelihood algorithm is basically a remedy for the belief propagation
decoding. It will be very useful when the message passing decoding cannot accomplish full restoration of data but the code is still decodable. The purpose of using the maximum likelihood algorithm is to decode with smaller overhead, but it will be very important to make the decoding matrix have rank equal to source symbol length at this low overhead with a high probability.

6.1 Traditional Method

Because with maximum likelihood a transmitted symbol (or packet) is considered error free or as an erasure at the output of an erasure channel, the correct symbol values are used to solve a system of linear equations and find the erased symbols. This method is analogous to Gaussian elimination which reduces the parity check matrix $H$ or the generator matrix $G$ to a row echelon form and recovers the erasures.

With an LDPC code, because

$$H\mathbf{x}' = \mathbf{0},$$  \hspace{1cm} (6.1)

where $\mathbf{x}'$ is a codeword, it solves the following linear equation system to find the unknowns $\bar{x}$:

$$H_{\bar{x}}\bar{x} = H_{\bar{x}}x,$$  \hspace{1cm} (6.2)

where $\bar{x}$ ($x$) is a column of lost (correctly received) symbols and $H_{\bar{x}}$ ($H_x$) is the set of those columns from $H$ corresponding to $\bar{x}$ ($x$). Suppose the size of binary $H$ is $m \times n$, where $n$ is the number of encoded bits, and the number of received bits is $a$. Thus the size of $H_{\bar{x}}$ is $m \times a$ and the size of $H_x$ is $m \times (n - a)$. It requires that the rank of $H_{\bar{x}}$ equals $n - a$ to solve Equation 6.1, so a necessary condition for its solvability is
\[ m \geq n - a. \] \hspace{1cm} (6.3)

Let \( k \) denote the number of source bits. Note that \( n - k \geq m \) because the REF of \( H \) may have all-zero bottom rows. Equation 6.3 can be turned into

\[ n - k \geq n - a \]

\[ a/n \geq k/n \]

\[ 1 - (n - a)/n \geq k/n \]

\[ 1 - p \geq R, \] \hspace{1cm} (6.4)

where \( p \) is channel bit erasure rate and \( R \) is code rate. Recall the capacity of BEC discussed in Chapter 2. Since the rate of an LDPC code is fixed, the channel loss rate needs to be known before choosing an LDPC code. In broadcast or multicast environments, data delivery links are equivalent to individual erasure channels whose loss rates can be very different, changeable or even unpredictable. In order to achieve near-capacity performance, power and bandwidth efficiency, and asynchronous data access, a single rateless code can outperform a single LDPC code.

With an LT code, the maximum likelihood decoding works on the generator matrix \( G \):

\[ Gu = x, \] \hspace{1cm} (6.5)

Where the \( k \times 1 \) vector \( u \) is the set of source symbols, the \( n \times 1 \) vector \( x \) is a subset of received symbols and \( G \) is a generator matrix constructed with the neighbor index information of each element in \( x \). The rank of \( G \) has to equal the number of source symbols for the solvability of Equation 6.5, so at least \( k \) reliable symbols should be
collected by the LT decoder. If the LT code guarantees successful decoding at \( n = k \), it is an ideal code in terms of overhead.

### 6.2 Fast Maximum Likelihood Algorithm

Traditionally, Gaussian elimination is sufficient to solve linear equation systems but it has complexity of \( O(n^3) \), where \( n \) is block length. Kim et al. [51] proposed a fast maximum likelihood algorithm with processing time of approximately \( O(n) \), which is support to maximum likelihood decoding of Raptor codes in the standard in the 3GPP Multimedia Broadcast/Multicast Services (MBMS) [54]. In fact, Kim’s fast algorithm is useful and general to both LT and Raptor codes.

Suppose that \( \mathbf{E} \) is a column of \( n \) received encoded bits and \( \mathbf{G} \) is an \( n \times k \) generator matrix reconstructed at the receiver. A combined decoding matrix \( \mathbf{B} \) can be set up as

\[
\mathbf{B} = [\mathbf{G} \quad \mathbf{E}].
\]  

(6.6)

Now Kim’s algorithm is going to change \( \mathbf{B} \) into an RREF. If \( \mathbf{B} \) has rank \( k \), all the source bits will be in the last (rightmost) column of its RREF. Note that in this last column the source bits would have been permutated because it may have exchanged some columns of \( \mathbf{G} \).

This algorithm is composed of four phases. During the decoding course, the RREF is produced step by step as a submatrix \( \mathbf{V} \) of \( \mathbf{B} \) is shrinking. Meanwhile, the rearranged order of the columns is recorded. \( \mathbf{V} \) is the intersection of all but the first \( i \) rows and all but the first \( i \) columns and the \( u + 1 \) columns of \( \mathbf{B} \), as shown in Figure 6.1. The
remaining $u + 1$ columns of $B$ but the last one form a submatrix $U$. $U_1$ and $U_2$ denote the first $i$ rows and the other $n - i$ rows of $U$, respectively.

Figure 6.1. Configuration of the submatrix $V$ of $B$.

Kim et al. defined the concept of score. The right nodes of $V$ correspond to its rows and the left nodes correspond to its columns. The score of the $j^{th}$ row of $V$ is the sum of the degrees of this right node’s neighbors. At the initial state of the decoding, $i = 0$, $u = 0$ and $V = G$. The process of the decoding is described as follows. In this algorithm, a row or column of $V$ represents the corresponding row or column of $B$, and vice versa.

Phase 1:

Step 1) Pick up one row in $B$ with $r$ 1’s in $V$, where $r > 0$. 
a) If \( r = 1 \), choose this row.

b) If \( r > 1 \), choose any such row with the highest score.

Step 2) Swap the first row of \( V \) and the chosen row. Reorder the columns of \( V \) so that the first entry in the chosen row of \( V \) is 1 and the other 1’s in this row are in the last \( r - 1 \) columns of \( V \).

Step 3) The first row of \( V \) is XORed with the other rows of \( V \) that have a 1 in the first column of \( V \) so that the first column of \( V \) has only one 1 on the top.

Step 4) \( i \) is increased by 1 and \( u \) is increased by \( r - 1 \). A new \( V \) is formed.

Step 5) If \( V \) has 1’s, go to Step 1). If \( V \) has disappeared \( (i + u = k) \), go to Phase 2. If \( V \) still exists but has no 1’s, the decoding fails and stops.

Phase 2:

Apply Gaussian elimination to \( U_2 \) and obtain its reduced row echelon form. If the rank of \( U_2 \) is smaller than \( u \), the decoding fails and stops. If successful, the bottom \( n - k \) rows of \( G \) are all-zero.

Phase 3:

Use \( U_2 \) to flip all the 1’s in \( U_1 \) to 0’s by XOR operation. Now the submatrix \( G \) of \( B \) has become an RREF and all the \( k \) source bits are located in the first \( k \) entries of the last column of \( B \).
Phase 4:

Use the recorded rearranged order of columns to restore the original order of the source bits in the last column of $B$.

In order to better approximate decoding time of $O(n)$, $u$ should be as small as possible because the Gaussian elimination is used in Phase 2. This is the reason that $r = 1$ is preferable in Phase 1. For $r > 1$, it needs to find out a row with the highest score because it is going to eliminate 1’s in $G$ at a time as many as it can. The degree distribution of the generator matrix $G$ greatly affects the complexity of Kim’s algorithm. For Raptor codes in the 3GPP MBMS, it can well approximate linear processing time, since it requires a fixed amount of row and column operations to calculate the score of every right node. For LT codes, it may not be so fast but this algorithm is still faster than Gaussian elimination.

6.3 Hybrid MP and Fast ML Algorithm for LT Codes

Message passing decoding is very fast with LT codes, but it fails with insufficient overhead, especially for short-length LT codes. Combining the conventional message passing algorithm and Kim’s fast maximum likelihood algorithm, we have proposed a hybrid decoding method [55] that enables the fast maximum likelihood stage on unreleased encoded symbols once the iterative message passing decoding is halted such that the decoding will succeed if the generator matrix has full rank. This algorithm is
similar to the hybrid Zyablov iterative decoding/Gaussian elimination scheme on LDPC
codes proposed in [56].

When this hybrid algorithm is applied to LT codes, the decoding can be still on
the fly. Once \( k \) encoded symbols have arrived, the receiver decodes with the message
passing algorithm. If it stops without successful decoding, the receiver keeps collecting
symbols from the channel. However, when it cannot receive any additional symbols, the
decoder switches to the fast maximum likelihood mode and tries to solve the system of
linear equations resulting from the unreleased encoded symbols. Note that the values of
the encoded symbols input to the maximum likelihood decoder could have been changed
due to the message passing decoding.

Basically, the more source symbols are recovered in the first stage, the faster the
decoding is. We use (average) recovery ratio to evaluate how well an LT code under
message passing decoding restores the source symbols from a certain number of channel
symbols. The recovery ratio is defined as the fraction of source symbols which have been
reproduced at the end of the iterative decoding. From the average recovery ratio mean
and its standard deviation, it is easy to see how often maximum likelihood decoding is
needed to take over the belief propagation decoding.

LT codes under message passing decoding can achieve smaller decoding
overhead as its code dimension increases, and the trade-off is the computation load
\( O(\ln(k/\delta)) \) for each source symbol. Short-length LT codes may require large overhead
which is unacceptable in some applications and the supportive maximum likelihood
algorithm helps reduce overhead consumption with negligible or nearly negligible extra
latency. This hybrid algorithm does not affect the universality of LT codes. It is valuable and useful in hard situations without sufficient channel symbols or packets.

6.4 Raptor Codes with Maximum Likelihood Decoding

Conventional Raptor codes have message passing decoding run on two matrices, respectively: the generator matrix of the inner LT code, and the parity-check matrix of the outer LDPC code. If the inner LT code does not restore enough intermediate symbols, the pre-code cannot recover all source symbols, no matter how many decoding iterations have been done on the matrices. But maximum likelihood decoding might be able to succeed. The maximum likelihood algorithm can be applied to the matrices: either one of them, or both of them. However, neither of the methods is apparently advantageous. If maximum likelihood decoding works with only one matrix, the message passing decoding could still be terminated by a stopping set in the other matrix. If the maximum likelihood algorithm is used on the both matrices, respectively, decoding can be slow and may not save considerable overhead. We need simple implementation of the fast maximum likelihood algorithm to arbitrary Raptor codes, which saves sizable overhead.

The 3GPP MBMS introduces maximum likelihood decoding on the encoding matrix of the Raptor code. This is an efficient and simple decoding scheme which can turn out a high success probability at low overhead. The construction of binary encoding matrix $A$ given in [54] is described as follows. Let column vector $D$ denote the $k$ source bits. The outer LDPC code has an $s \times k$ generator matrix $G_{LDPC}$. $G_H$ is the $h \times (k + s)$ generator matrix of half symbols. The systematic intermediate symbols $F$ are formed as
\[ F = \begin{bmatrix} D \\ D_s \\ D_h \end{bmatrix}, \quad (6.7) \]

where

\[ D_s = G_{LDPC} D \quad (6.8) \]

and

\[ D_h = G_H \begin{bmatrix} D \\ D_s \end{bmatrix}. \quad (6.9) \]

In the end, the \( n \) encoded Raptor bits \( E \) are obtained by directly encoding the \( k + s + h \) intermediate bits with the generator matrix \( G_{LT} \) of the inner LT code:

\[ E = G_{LT} F. \quad (6.10) \]

The configuration of the encoding matrix \( A \) is shown in Figure 6.2.

\[ \begin{array}{ccc}
  & k & \\
 s & G_{LDPC} & I_s & Z \\
 h & G_H & I_h \\
n & G_{LT} \\
\end{array} \]

*Figure 6.2. Encoding matrix \( A \) of a nonsystematic Raptor code.*
\( \mathbf{I}_s \) and \( \mathbf{I}_h \) are identity matrices and \( \mathbf{Z} \) is an all zero matrix. In a word, the Raptor encoding is simply as

\[
\mathbf{A}\mathbf{F} = \begin{bmatrix} \mathbf{0} \\ \mathbf{E} \end{bmatrix}. \tag{6.11}
\]

With maximum likelihood algorithm, the Raptor decoder uses the encoding matrix and reliable received bits to establish a system of linear equations. By solving these equations, the source bits are restored.

One possible drawback of this Raptor ML decoding is the size of \( \mathbf{A} \) which is much larger than that of any of \( \mathbf{G}_{LDPC}, \mathbf{G}_H \) and \( \mathbf{G}_{LT} \), so in general high rate pre-codes are preferable. Another disadvantage is that the density of \( \mathbf{G}_{LDPC} \) may be much higher than the corresponding parity-check matrix \( \mathbf{H}_{LDPC} \) and it brings about more computation cost.

6.5 Summary

Maximum likelihood decoding is useful to fountain codes, especially when received encoded symbols or packets are very limited. For both LT codes and Raptor codes, maximum likelihood decoding is carried out on the generator matrix or the encoding matrix. Compared to message passing decoding, it saves overhead at the expense of processing time. We recommend applying Kim’s fast maximum likelihood algorithm to remedy the iterative decoding. Once any channel symbols or packets are collected, message passing decoding is usually preferred. In general, with fountain codes, message passing decoding is preferable as long as additional symbols can be received. Our hybrid MP/fast ML algorithm is efficient and practical for LT codes, especially
short-length LT codes. It holds fast decoding speed and guarantees successful decoding at the minimum encoded symbols that establish the full rank of the decoding matrix.
CHAPTER 7: SIMULATION RESULTS OF FOUNTAIN CODES

Simulations of fountain codes have been done with MATLAB. Simulation results are illustrated in this chapter. With these results, analysis and comments are made. Without losing generality, every symbol or packet payload is assumed to be a bit. A block of bits is also called a frame. The local-iteration and global-iteration message passing algorithms are used for all the Raptor codes over BEC or AWGNC. The maximum likelihood decoding is applicable for binary erasure codes only.

7.1 Construction of Fountain Codes

In this section, some parameters are set in order to construct fountain codes for simulation.

Given any dimension $k$ for an LT code, we just need to specify $c = 0.1$ and $\delta = 0.05$ with the Robust Soliton distribution.

For Raptor codes, only one-stage pre-codes are applied. Equations 3.11 – 3.13 are regulations to high rate pre-codes ($0.5 < R_c < 1$), but rate 1/2 pre-codes are also simulated. When $R_c = 0.5$, we set $\varepsilon = 2/3$. Simulated Raptor codes of dimension $k$ are as follows:

1) R-LR4 .98 $k$: pre-code is a rate 0.98 left-degree 4, right-Poisson code.
2) R-LR3 .98 $k$: pre-code is a rate 0.98 left-degree 3, right-Poisson code.
3) R-LR4 5/6 $k$: pre-code is a rate 5/6 left-degree 4, right-Poisson code.
4) R-SF4 5/6 $k$: pre-code is a rate 5/6 left-degree 4 code generated with our splitting-and-filling technique.
5) R-IE 5/6 $k$: pre-code is the rate 5/6 IEEE 802.16e code.
6) R-LR4 1/2 k: pre-code is a rate 1/2 left-degree 4, right-Poisson code.

7) R-SF4 1/2 k: pre-code is a rate 1/2 left-degree 4 code generated with our splitting-
and-filling technique.

8) R-IE 1/2 k: pre-code is the rate 1/2 IEEE 802.16e code.

We have presented investigation on R-LR and R-IE codes in [61]. In this
dissertation, R-SF codes are added.

For each of these Raptor codes, the parity-check matrix of the pre-code is known
prior to data transmission. The encoder converts the original parity-check matrix to a
staircase matrix and applies the fast encoding algorithm (Richardson and Urbanke [59])
to encode. The generator polynomials of the weakened LT codes for the three different
pre-code rates 0.98, 5/6 and 1/2 are given by

\[ \Omega_{0.98}(x) = 0.0208x + 0.4896x^2 + 0.1632x^3 + 0.0816x^4 + 0.0490x^5 \]
\[ + 0.0326x^6 + 0.0233x^7 + 0.0175x^8 + 0.0136x^9 + 0.0109x^{10} \]
\[ + 0.0089x^{11} + 0.0074x^{12} + 0.0063x^{13} + \cdots + 0.0098x^{101}, \quad (7.1) \]

\[ \Omega_{5/6}(x) = 0.2381x + 0.3810x^2 + 0.1270x^3 + 0.0635x^4 + 0.0381x^5 \]
\[ + 0.0254x^6 + 0.0181x^7 + 0.0136x^8 + 0.0106x^9 + 0.0085x^{10} \]
\[ + 0.0069x^{11} + 0.0058x^{12} + 0.0049x^{13} + 0.0586x^{14} \quad (7.2) \]

and

\[ \Omega_{1/2}(x) = 0.3077x + 0.3462x^2 + 0.1154x^3 + 0.0577x^4 + 0.0346x^5 + 0.0231x^6 \]
\[ + 0.0165x^7 + 0.0124x^8 + 0.0096x^9 + 0.0077x^{10} + 0.0692x^{11}. \quad (7.3) \]
7.2 Fountain Codes over Binary Erasure Channel

Figure 7.1 shows the error performance of an LT code of dimension $k = 1000$ with message passing decoding. Besides bit error rate, frame error rate is of interest since FER is the decoding failure error probability. Within 1200 received bits, the FER is 1 or almost 1. Once the decoding overhead is 0.25 or more, the BER and the FER curves fall rapidly. That means, with enough received symbols, the LT code can guarantee finding source symbols with a very high probability. Table 1 illustrates the average recovery ratio of the LT code with message decoding and its standard deviation versus the number of received bits. Within 1260 received bits, the standard deviation increases as the number of received bits increases, i.e., the performance is not stable and the message passing decoding cannot guarantee successful decoding in this range. The decoding failure rate is very low and the performance gets stable at the price of high overhead.
Figure 7.1. LT Code \((k = 1000)\) with MP Decoding over BEC.

Table 1

**LT Code’s Average Recovery Ratio and Standard Deviation**

<table>
<thead>
<tr>
<th>Number of received bits</th>
<th>1000</th>
<th>1050</th>
<th>1100</th>
<th>1150</th>
<th>1200</th>
<th>1220</th>
<th>1240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average recovery ratio</td>
<td>0.099</td>
<td>0.124</td>
<td>0.171</td>
<td>0.209</td>
<td>0.305</td>
<td>0.435</td>
<td>0.497</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.036</td>
<td>0.048</td>
<td>0.065</td>
<td>0.091</td>
<td>0.157</td>
<td>0.197</td>
<td>0.281</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of received bits</th>
<th>1260</th>
<th>1280</th>
<th>1300</th>
<th>1320</th>
<th>1340</th>
<th>1360</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average recovery ratio</td>
<td>0.642</td>
<td>0.786</td>
<td>0.886</td>
<td>0.943</td>
<td>0.972</td>
<td>0.988</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.310</td>
<td>0.296</td>
<td>0.248</td>
<td>0.187</td>
<td>0.137</td>
<td>0.095</td>
</tr>
</tbody>
</table>
Maximum likelihood decoding reduces overhead for this LT code. The hybrid MP/fast ML decoding algorithm is applied to the LT code. Its error rates versus decoding overhead are illustrated in Figure 7.2. The hybrid decoding can accomplish good error performance at very low overhead $\varepsilon = 0.01$. It has error floors over the overhead interval $(0.01, 0.06)$, but the FER falls rapidly again after that because the probability that the rank of the LT generator matrix is equal to $k$ increases steeply as sufficient encoded bits are received. Note that this FER is also the probability that the LT code is not solvable.

Figure 7.2. LT code ($k=1000$) with hybrid MP/fast ML decoding over BEC.
The performances of the Raptor codes under local and global-iteration message passing and fast maximum likelihood decoding are shown in Figure 7.3 – 7.8.

*Figure 7.3.* BERs of Raptor codes with local-iteration MP decoding over BEC.
Figure 7.4. FERs of Raptor codes with local-iteration MP decoding over BEC.
Figure 7.5. BERs of Raptor codes with global-iteration MP decoding over BEC.
Figure 7.6. FERs of Raptor codes with global-iteration MP decoding over BEC.
Figure 7.7. BERs of Raptor codes with ML decoding over BEC.
The simulated error performances show that, for $0.5 < R_c < 1$, higher pre-code rates require less decoding overhead with any of the three decoding methods. Note that the polynomial of the degree distribution of LT encoded symbols for rate 1/2 pre-codes is actually for rate 2/3 pre-codes. Of a single pre-code rate, the error rates of the Raptor codes are roughly close, no matter which pre-codes are used. At the point of this view, the pre-code rate is more important than the pre-code’s own performance. Given a certain rate, a free and sparser pre-code would be preferable. The local-iteration method in general saves more overhead than the global-iteration method. The overhead of

*Figure 7.8. FERs of Raptor codes with ML decoding over BEC.*
maximum likelihood decoding is much smaller, but the price is much higher latency and its error rate curves are not as steep as those with message passing decoding.

Overhead statistics can illustrate a code’s erasure recovery capability. The average overheads of our fountain code are shown in Figure 7.9. The average overhead of the LT code falls steadily versus its code dimension, which tends to vanish for large data length. The LT code with maximum likelihood decoding saves much more overhead. With message passing decoding, it can reduce the overhead by decreasing $\delta$, but it causes logarithmically increasing computation cost. With either local-iteration message passing decoding or fast maximum likelihood decoding, the Raptor codes consume very close average overheads as long as the pre-code rates are the same. For pre-code rates 5/6 and 1/2, the fast maximum likelihood decoding consumes less average overhead. For pre-code rate 0.98, however, the local-iteration message passing decoding needs more overhead than the maximum likelihood decoding, and the overhead difference between the two decoding algorithms decreases as code dimension grows. The Raptor codes of pre-code rate 5/6 and 1/2 have ‘overhead floors’ and the Raptor codes of pre-code rate 1/2 with local-iteration message passing decoding require very high average decoding overhead.
7.3 Fountain Codes over Additive White Gaussian Noise Channel

Experiments indicate that our LT code does not guarantee near capacity performance at low or moderate signal-to-noise ratios. Its performance is not stable. Thus, we focus on Raptor codes in this section. The performances of rate 1/3 Raptor codes under local and global-iteration message passing decoding are shown in Figure 7.10 – 7.13.
Figure 7.10. BERs of Raptor codes with local-iteration MP decoding over AWGNC.
Figure 7.11. FERs of Raptor codes with local-iteration MP decoding over AWGNC.
Figure 7.12. BERs of Raptor codes with global-iteration MP decoding over AWGNC.
Over AWGNC, the rate 0.98 pre-codes no longer significantly outperform the rate 5/6 codes. With global-iteration decoding, the R-LR .98 codes have very similar FER’s at low signal-to-noise ratio as the R-LR 5/6, R-SF 5/6 and R-IE 5/6 codes but the R-LR .98 codes tend to have bit error floors beyond $E_b/N_0 = 2 \text{dB}$. The performances of the R-LR, R-SF and R-IE 5/6 and 1/2 codes are very poor at low signal-to-noise ratio.

The code’s performance can be improved by setting a threshold to select more reliable encoded bits, as described in Section 4.4.4. The receiver is assumed to be able to collect as many encoded bits as needed and then it chooses those received bits whose absolute values are greater than the threshold. We set the threshold to be 0.6 and a

---

**Figure 7.13.** FERs of Raptor codes with global-iteration MP decoding over AWGNC.
received bit without noise is ±1. R-LR4 .98 and R-LR3 .98 codes are simulated and the results are shown in Figure 7.14. With the bits selection threshold, the codes achieve coding gain of approximately 0.8 dB at BER of $10^{-4}$. The performances of these codes are very close.

![Figure 7.14](image-url)

*Figure 7.14. Error rates of Raptor codes with selection threshold 0.6 over AWGNC.*

Finally, simulate R-LR4 .98 and R-LR3 .98 codes over AWGN channels with Rayleigh fading. This channel model is still memoryless. Set the variance of the inphase and quadrature components of the received band-pass signal without AWGN: $b = 1$. The simulation results are shown in Figure 7.15. The difference between these codes’ error
performances can be negligible. Their performances are very poor at low signal-to-noise ratio \((E_b/N_0 < 3 \text{ dB})\) but beyond this point they achieve substantial coding gains.

Figure 7.15. Error rates of Raptor codes over AWGNC with Rayleigh fading.
CHAPTER 8: CONCLUSIONS AND FUTURE WORK

In this dissertation, we have exhibited studies of digital fountain codes, describing code construction, introducing system models, addressing their applications on different memoryless noisy channel models, and discussing the pre-codes of Raptor codes. We investigated good fountain codes in a particular scenario for better error performance, lower computation complexity and/or smaller decoding overhead. In this chapter, we make conclusions of this dissertation and discuss future research work.

8.1 Conclusions

We need universal codes for some applications, especially in form of broadcast or multicast services or communications. Rateless codes are suitable choices because, for data packet delivery over networks, they can recover source packets regardless of channel data loss model. As new rateless codes, digital fountain codes have become our research interest. We developed detailed studies of fountain codes over erasure and AWGN channels. The two fountain code system models illustrate general pictures for LT codes, Raptor codes and LDPC codes. Since our research group has also accomplished new method of constructing irregular LDPC codes with low error floors, we tested some Raptor codes whose pre-codes are chosen from existing good LDPC codes and our simply constructed left-regular LDPC codes. Two classic decoding methods, message passing and maximum likelihood, for fountain codes have been studied. We have proposed a hybrid message passing and fast maximum likelihood decoding algorithm for LT codes, which can ensure successful data restoration with high probability at very low
overhead without sacrificing much decoding speed. Simulation results show that we can select a suitable fountain code for a given channel model with specific demand on data length, error rate, decoding latency or overhead, or with a balance of these demands.

LT codes, the first practical fountain codes, are a very good example to illustrate the name of fountain. In order to make every water drop (encoded symbol or packet) in the fountain reliable and useful, the drops must be i.i.d. so the degree of a drop should be wisely designed. The Robust Soliton distribution can guarantee full data recovery with some decoding overhead. However, this amazing distribution has some weaknesses: high error floors may occur; overhead could be large (small) for short (long) data length; complexity increases logarithmically as code dimension grows. A Raptor code can achieve a lower error floor and constant complexity for each symbol, but it does not necessarily mean that the complexity is low since the Raptor code is a concatenated code.

LT codes and Raptor codes with rapid message passing decoding are originally packet erasure codes but researchers have found that they apply to correct bit errors caused by AWGN. We have presented two system models of fountain codes under message passing decoding in Chapter 4: one for packet erasure correction, and the other for bit error correction. With the belief propagation decoding algorithm, both models are the same in nature. The difference is that for bit error correction, erasure pattern can be used or not, but for packet recovery, unreliable received packets must be discarded.

All Raptor codes of interest have one-stage pre-codes only and all these pre-codes are chosen from LDPC codes. Thus, we have reviewed the LDPC codes in the IEEE 802.16e standard, left-regular right-Poisson LDPC codes and left-regular LDPC codes
generated with our splitting-and-filling technique. With same code rate, an IEEE 802.16e LDPC code has the best error performance on both AWGNC and BEC though higher decoding cost might be required (the density of its parity-check matrix is higher than that of the other two kinds of left-regular LDPC codes).

In spite of possible high latency, maximum likelihood decoding is very useful for packet erasure fountain codes when decoding overhead is very limited. The 3GPP MBMS standard contains the configuration of Raptor codes with two-stage pre-codes and systematic intermediate symbols. These Raptor codes are decoded with maximum likelihood algorithm based on encoding matrices. Due to the large size of the encoding matrix, complexity and space are accordingly larger than LT codes only. Kim et al. improved this algorithm and accomplished approximately linear decoding cost, though it is still slower than message passing algorithm. In order to realize fast decoding of LT codes with small overhead, we have proposed a MP/fast ML algorithm which substantially reduces overhead, especially for small code dimensions. This scheme keeps high speed at the first decoding stage and lets the maximum likelihood algorithm decode with the stopping set (if any). This hybrid method always succeeds once the rank of the generator matrix is equal to the number of source symbols.

We have simulated both LT codes and Raptor codes over binary erasure channels and additive white Gaussian noise channels. With message passing decoding, the LT code can avoid high latency but may require large overhead. The maximum likelihood algorithm not only reduces numerous overhead for all code dimensions, but also keeps overhead decreasing as data length grows. Our LT code is able to achieve near capacity
performance upon BEC, but it works poorly over AWGNC and the performance is not stable. Our Raptor codes all have one-stage pre-codes. All these pre-codes are LDPC codes and their code rates are 0.98, 5/6 and 1/2. They are chosen from the IEEE 802.16e LDPC codes, left-regular right-Poisson LDPC codes and the LDPC codes created by our splitting-and-filling method. The simulation results demonstrate that in general, higher rate pre-codes enforce better error performance and lower complexity. For the same pre-code rate, the performances of the Raptor codes do not vary much with different pre-codes. Over BEC, lower rate pre-codes can bring about much higher decoding overhead because they need much more encoded symbols to restore greater amounts of intermediate symbols. Upon AWGNC, the rate 0.98 pre-codes realize the best performance but they tend to have error floors at moderate signal-to-noise ratios where they are outperformed by the rate 5/6 pre-codes. We can use lower rate Raptor codes to achieve better performance. However, it causes high cost and space consumption. We recommend selecting more reliable received symbols to decode. This can be done by setting acceptance threshold. The simulation result shows that error floors also disappear with this scheme. If Rayleigh fading is considered, the rate 0.98 pre-coding Raptor codes have good performances at moderate SNR but at low SNR, these Raptor codes do not work as well as BPSK without channel coding.

In summary, as long as sufficient received symbols can be used to decode, message passing decoding is our first choice for both LT codes and Raptor codes. With Raptor codes, global-iteration decoding is preferable at high signal-to-noise ratios for less iterations. The IEEE 802.16e LDPC codes may not be very good pre-codes, because the
rates of the 802.16e codes are not very high, and the code rates and the block length
options are very limited. High rate and very sparse pre-codes are usually preferable for
smaller error rates, lower cost, smaller overhead and lower space consumption.

8.2 Future Work

In this ending section, we present some possible future work that might interest
the reader based on the theories and results provided in this dissertation.

The structure of LT codes are much simpler than that of Raptor codes, so very
short length (e.g., < 10^3) LT codes can be easy and cheap to implement. The overall
complexity could be restricted in a certain bound. However, we have not achieved good
results of LT codes over AWGNC. There are two obvious possible ways we can research
this. The first one is the degree distribution for encoded symbols. The Robust Soliton
distribution limits a very small fraction of degree-1 encoded symbols. A suitable degree
distribution can be expected. The second way is based on the decoder. There might be
one or more trapping sets or stopping sets in the generator matrix of the LT code. If we
find out the sets, a revised decoder can further correct more errors and lower the error
floor.

The degree polynomial of the weakened LT code is also worthy of investigation
so that more coding gain can be achieved when using a same pre-code. The simulation
results show that lower pre-code rate makes higher overhead. So we expect a new Soliton
distribution to reduce overhead for higher rate pre-codes.
Many coding schemes are getting very close to Shannon limit but the research in coding theory is not dying because at least we still need to consider complexity. This is very possible for Raptor codes because extremely sparse LDPC codes can be very good pre-codes.

Video-on-demand and large scale data distribution are hot topics in today’s Internet services. Digital fountain codes have been applied to HDTV and multicast delivery of file download. We have only studied some basic and theoretical fundamentals of fountain codes. New applications deserve research.
REFERENCES


[15] The official home page of 3GPP.


[40] Chih-Chun Wang, Sanjeev R. Kulkarni, and H. Vincent Poor, “Exhausting Error-Prone Patterns in LDPC Codes”.


[61] W. Huang and J. Dill, “A study of the pre-codes of Raptor codes,” submitted to MILCOMM, Nov. 2011, Baltimore, MD, USA.
