LT code : decoding algorithm and its applications

Lu, Haifeng

2015

Lu, H. (2015). LT code : decoding algorithm and its applications. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/65451>

<https://doi.org/10.32657/10356/65451>

LT Code: Decoding Algorithm and Its Applications

LU Haifeng

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2015

Abstract

Reliable transmission over error-prone channels is always a problem for network researchers. To control errors in transmissions, the technique Forward Error Correction (FEC) was proposed in 1940s. Among different coding schemes, Reed-Solomon and Low Density Parity Check codes are the most successful ones. Although these codes can achieve channel capacity, the performance relies on knowledge of channel condition since the coding rate is determined beforehand. To overcome this, fountain codes were proposed. For a given number of source symbols, fountain codes can generate potentially limitless encoded symbols. Luby Transform (LT) codes are the first realizations of fountain codes. The low complexity of both encoding and decoding makes LT codes applicable in many different applications from data transmission to storage system. This thesis is dedicated to improve LT decoding algorithm under certain circumstance and apply LT codes in multimedia transmission and in cloud storage system.

Past research has shown that LT codes can perform well for a large number of source symbols. However, mathematical analysis and simulation results have revealed that the packet overhead for LT decoders can be as large as 100% when the number of source symbols is small. LT-W decoding was proposed to tackle this problem. In the first part of this thesis, we make an observation that LT decoders often fail to recover all the source symbols, while LT encoders have a high probability of producing a full-rank coefficient matrix. Through rigorous analysis and extensive experiments, we show that LT-W decoding can dramatically reduce the overhead for LT codes with small input symbol.

LT codes have been proven successful in content delivery due to their abilities to handle varying channel conditions without much feedback. However, original LT codes are

not unable to provide intermediate outputs when delivering layer-coded media content. Although some methods have been proposed to produce intermediate outputs based on adjusting the distribution of LT codes, they are typically content-dependent and unable to guarantee that a more important layer can always be decoded before the decoding of a less important layer. In the second part of this thesis, we propose a simple joint Unequal Loss Protection (ULP) and LT coding (ULP-LT) scheme for layered media delivery, where different numbers of FEC are allocated to different layers to guarantee the priority and LT codes are used to deal with varying channel conditions. Simulation results show that with a small amount of overhead allocated to ULP, the ULP-LT scheme can produce good intermediate performance while still enjoying the nice features provided by LT codes.

Lastly, we investigate the delay performance of a LT-based cloud storage system. LT codes are popular for storage systems due to their efficient recovery. However, in order to ensure high success decoding of LT codes based storage, retrieval of additional fragments is required, and this requirement could introduce additional delay. We show that multiple stage retrieval of fragments is effective to reduce the file-retrieval delay. We first develop a delay model for various multiple stage retrieval schemes applicable to our considered system. By employing the developed model, we study optimal retrieval schemes given requirements on success decodability. Our numerical results suggest a fundamental tradeoff between the file-retrieval delay and the target probability of successful file decoding, as well as that the file-retrieval delay can be significantly reduced by optimally scheduling packet requests in a multi-stage fashion.

Acknowledgments

In the past four years, there is a saying which has always echoed in my mind. “It’s the Ph.D for yourself,” said by my supervisor, Dr. Cai Jianfei, at the very beginning of my Ph.D life. It was that saying which kept me struggling whenever I felt lost and supported me to finish this dissertation. Taking this opportunity, I would like to express my greatest appreciation to Dr. Cai, for his continuous help throughout my Ph.D life. His words and deeds showed me how to be a researcher and a helpful adult. Also, I would like to thank Dr. Foh Chuan Heng and Dr. Wen Yonggang, my co-supervisors, for their encourages and advices. Without them, I would never go so far.

As others whispering that Ph.Ds are bored, I am so lucky to have some not-so-bored Ph.D fellows here. They made my Ph.D life much more colorful and it is my great honor to have worked with them: Dr. Chen Xiangang, Dr. Duan Lixin, Dr. Duan Qi, Dr. Fang Shuo, Dr. Guo Tiantian, Dr. Gao Shenghua, Dr. Jiang Xing, Mr. Liu Yong, Dr. Teng Xiao, Dr. Wang Danqi, Dr. Wu Min, Dr. Wang Zhengxiang, Dr. Yang Ming, Dr. Yang Shengbo, Dr. Zhong Feng, Dr. Zhang Juyong, Dr. Zhu Linhong and Dr. Zhang Yu. Thank you all!

Last but not least, I would extend my utmost gratitude to my parents. For the last 5 years, their unconditional love and continuous support have made it possible for me to focus on study. This dissertation is for them.

Contents

Abstract	ii
Acknowledgments	iv
List of Figures	viii
List of Tables	xi
 Abbreviations	 xii
 1 Introduction	 1
1.1 Background	1
1.2 Research Problems	4
1.2.1 LT decoding with small number of symbols	4
1.2.2 LT codes with unequal loss protection	5
1.2.3 LT codes in cloud storage system	6
1.3 Summary of Contributions	6
1.4 Thesis Organization	7
 2 Literature Review	 9
2.1 LT Codes	9
2.1.1 LT encoding and decoding	9
2.1.2 The Success Probability of Decoding	13

2.2	LT-W: Improved LT Decoding with Wiedemann Solver	14
2.2.1	Deciding which symbol to borrow	15
2.2.2	Recovering the Borrowed Symbol	16
2.2.3	Non-square Case	18
2.2.4	Analysis of Random Matrix Rank	19
2.3	Other Work on Improving LT Decoding Performance	24
2.4	Work on Providing Unequal Loss Protection to LT Codes	26
2.5	Work on Cloud Storage Systems	29
3	Performance analysis of LT-W	32
3.1	Why LT codes fail	32
3.2	Percentage of Overhead Packets	34
3.3	Number of Borrowed Symbols	36
3.4	LT-W V.S. Inactivation	41
3.5	Running Time	43
3.6	Conclusion	43
4	Joint Unequal Loss Protection and LT Coding for Layer-Coded Media Delivery	45
4.1	Review of ULP	46
4.2	Proposed Joint ULP and LT Coding System	47
4.2.1	Encoding and decoding	48
4.2.2	FEC Allocation	51
4.3	Simulation Results	53
4.4	Conclusion	56

5	Delay-Optimized File Retrieval under LT-Based Cloud Storage	57
5.1	System Model and Problem statement	58
5.1.1	System Architecture	58
5.1.2	Traffic Model	59
5.1.3	Problem Statement	60
5.1.4	LT Decodability	62
5.2	File-Retrieval Delay Analysis	64
5.2.1	Delay Analysis for One Stage Request Scheme	64
5.2.2	Delay Analysis for Multiple Stage Request	66
5.3	Optimal Scheduling for Packet Retrieval	70
5.4	Delay-Decodability Tradeoff	76
5.5	General Traffic Model	77
5.6	Conclusion	79
6	Conclusion and Future Works	80
6.1	Conclusion	80
6.2	Future Directions	82
	References	85

List of Figures

1.1	An example of the layered structure.	6
2.1	An example of LT encoding and decoding procedures for $k = 4$ and $n = 5$	12
2.2	The success probability of LT decoding with respect to the number of received packets.	15
2.3	An example of the degree changing from i to j with the $(q + 1)^{th}$ vector added. Note that the dark region indicates the places of 1s.	21
2.4	The probability of obtaining a full-rank coefficient matrix with respect to the number of received packets.	24
3.1	The KL divergence between the preset and actual degree distributions where exact k packets are received.	33
3.2	The overhead for 95% probability of successful decoding.	35
3.3	The overhead for 98% probability of successful decoding.	36
3.4	The expected number of borrowed symbols with respect to the number of received packets.	40
3.5	Comparison between inactivation decoding and LT-W decoding.	42
3.6	The actual running time with respect to the number of input symbols.	44
4.1	A layered unequal loss protection structure.	46
4.2	The system diagram of proposed UEPLT scheme.	47

4.3	The probability of a receiver reaching the full rank state under different numbers of original packets and different proportions of FEC packets. (a) 10 original data packets. (b) 20 original data packets. (c) 30 original data packets. (d) 40 original data packets	50
4.4	The decoding probabilities of different layers. Left: with 5% FEC redundancy. Right: with 10% FEC redundancy.	54
4.5	The average number of decoded layers using different methods.	55
4.6	The average number of decoded layers for heterogeneous users. Left: with 5% FEC redundancy. Right: with 10% FEC redundancy.	56
5.1	System Architecture for a cloud storage system.	58
5.2	Arrival process of LT encoded packets seen at the portal for a file-retrieval request.	60
5.3	Fitting result versus simulation result.	62
5.4	Components of delay of one stage request	64
5.5	Comparison between $\overline{D}_1(n)$ and $\tilde{D}_1(n)$	66
5.6	Delay in one stage request. 1: $\lambda_a = 10, \theta = 25$; 2: $\lambda_a = 30, \theta = 50$; 3: $\lambda_a = 60, \theta = 50$	67
5.7	Components of delay of two stage request	67
5.8	Delay in two stage request when $k = 100$ and $n = 200$	69
5.9	Delay in two stage request when $k = 400$ and $n = 700$	70
5.10	Minimal delay comparison among one-stage request, two-stage request, three-stage request when $k = 100$	71
5.11	Minimal delay comparison among one-stage request, two-stage request, three-stage request when $k = 400$	72
5.12	Comparison between $W(x)$ and its bound.	74
5.13	Comparison of optimal first stage request between simulation and approximation under different k	75

5.14	Optimal first stage request ratio when $k = 100$ and $k = 400$	76
5.15	Tradeoff between decoding probability and delay when $k = 100$ and $k = 400$	77
5.16	Simulation result of MMPP model.	78

List of Tables

1.1	Comparison among different fountain code approximations	4
4.1	Symbol Definitions	53
5.1	Fitting result for α and β	63
5.2	Default values in Simulation	64

Abbreviations

LT codes Luby Transform codes

BCH codes Bose-Chaudhuri-Hocquenghem codes

RS codes Reed-Solomon codes

LDPC codes Low-Density-Parity-Check codes

GOP Group Of Pictures

RTT Round-Trip Time

ULP Unequal Loss Protection

EFW codes Expanding Window Fountain codes

SSD Solid State Driver

DRAM Dynamic Random Access Memory

PLR Packet Loss Probability

PMF Probability Mass Function

MMPP Markov-Modulated Poisson Process

AWGN Additive White Gaussian Noise

BSC Binary Symmetric Channel

Chapter 1

Introduction

1.1 Background

In [1], Shannon predicted that reliable communication could be achieved through adding redundancy to messages. However, he suggested situations for reliable communication, but has not designed any framework for that. Since then, researchers in communication society have put continuous efforts into designing codes which are able to achieve the theoretical capacity bounds.

During the past 60 years, many important codes, including Hamming codes [2], BCH codes [3, 4, 5], RS codes [6], convolutional codes [7] and LDPC codes [8], were proposed and successfully applied in real-life applications. These codes share a common property: fixed-rate which means the coding rate is determined before transmission. These rate-fixed codes work well when channel error rate is known or can be accurately predicted and this rate is stable during transmission. Although these rate-fixed codes do not require feedback, they perform poorly at mismatched channel conditions, i.e. wasting bandwidth at better channel conditions while failing to recover the original data at worse channel conditions, where retransmission is still needed.

Later, Byers *et al.* proposed the concept of fountain codes in [9] to avoid this mismatch

shortcoming. By applying fountain codes, a sender can inject a stream of distinct encoded packets into a network, from which a receiver can reconstruct the source data by collecting any subset of the encoded packets where the total number of bits collected from the packets is equal to that of the source data. In this process, it should not matter which encoded packets are collected. Since encoded packets can be potentially infinite, fountain codes are also referred to rateless codes.

According to [9], an ideal fountain code should exhibit the following properties. Given a message containing k input symbols each with length l for transmission, this code should allow the source to generate an infinite supply of encoded packets of length l . The client can then reconstruct the original message once any k encoded packets are received, and the complexity of this reconstruction process should be linear in k .

While fountain codes have various applications, we mention three here.

- **Multicast:** Fountain codes were introduced in the context of reliable multicast [10], such as for delivery of software packages. By applying fountain codes, the benefits are apparent. As each user experiences independent losses, requesting retransmission from the server may cause feedback implosion in the network. Using fountain codes can eliminate the need for feedback and retransmission. Besides, fountain codes also handle the heterogeneous-user problem. Server can transmit packets at full speed of the channel while different users can drop some packets according to their channel speed. Since every packet from the server is useful, dropping packets in this manner may not hurt the slower users if there is no delay requirement.
- **Video Streaming:** While downloading a complete movie can be straightforward by applying fountain codes, providing streaming video or on-demand video is challenging. The key problem is latency: if the entire video file is encoded in one block, video decoder must wait for the entire block being received and decoded. This will

incur a long delay which is intolerable for most video streaming applications. As indicated in [11], a common approach is breaking the entire file into segments and encoding these segments independently. A user can playback current segment and downloading next segment at the same time. Thus, the delay can be controlled by adjusting the length of segment.

- **Storage:** In storage system, the most important thing is reliability. When some data blocks are damaged, it must have solutions to recover these blocks. The easiest way to achieve reliability is replication. In [12], Weatherspoon *et al.* analyzed two schemes for redundancy: replication and erasure coding, concluding that erasure-coded system can provide higher availability with lower bandwidth and less storage space. Compared with other erasure codes, fountain codes provide more flexibility on data redundancy ratio because of their rateless property. Recently, more and more researches focus on applying fountain codes on storage systems [13, 14, 15, 16, 17].

While fountain codes share such nice properties (e.g. rateless, easy to encode and decode) and have many potential applications, designing fountain codes is challenging. One potential way to realize fountain codes is through the use of an erasure code. An erasure code such as Reed-Solomon (RS) codes [6, 18, 19] can produce sufficiently large number of encoded packets from k input symbols. However, the decoding complexity of RS codes is of quadratic order and the number of encoded packets must be determined prior to encoding [20, 21]. Thus, a RS code does not meet the properties of fountain codes. Another potential candidate to realize fountain codes is to use Tornado codes [21] which are based on Low-Density Parity-Check (LDPC) codes [8]. However, the number of encoded packets, n , needed to reconstruct the source data is slightly larger than k , and the decoding complexity depends on n instead of k .

Luby Transform (LT) codes [10] are the first realization of fountain codes. The symbol length of the code can be arbitrary, from one-bit binary symbols to general l -bit symbols. If an original file can be described by k symbols, then each coded packet can be independently generated, and the content of the original file can be recovered from any $k + O(\sqrt{k} \log^2(k/\delta))$ packets with a probability of $1 - \delta$ involving an average of $O(k \log(k/\delta))$ symbol operations. Following [10], Raptor codes were developed by Shokrollahi [22] as fountain codes with even lower decoding complexities than LT codes. Raptor codes make use of a pre-coder before the typical LT process. All the above is summarized in Table 1.1.

Table 1.1: Comparison among different fountain code approximations

Code	Negatives	Positives
Reed-Solomon	Limited encoded packets	Only k encoded packets needed for decoding
	Quadratic decoding time	
Tornado	Limited encoded packets	Decoding complexity is linear to n
	Need $n > k$ encoded packets for decoding	
LT	Need $n \geq k$ encoded packets for decoding	Unlimited encoded packets
		Decoding complexity is linear to $k \log k$
Raptor	Need $n \geq k$ encoded packets for decoding	Unlimited encoded packets
		Decoding complexity is linear to k

Despite of existing different fountain codes, in this thesis, we focus on LT code and its application.

1.2 Research Problems

1.2.1 LT decoding with small number of symbols

It has been shown in [22] that LT codes perform very well for large values of k . By optimizing the degree distribution using linear programming, the packet overhead ϵ can

be as small as 3% for $k > 100000$. However, in reality, small numbers of symbols are often encountered. For example, in video streaming applications, k can be the number of frames in a Group Of Pictures (GOP), which is typically in the range of 10 to 100. In data packet transmission, erasure codes have also been proposed for congestion control in the domain of Internet routing [23]. However, to ensure the correctness of Transmission Control Protocol (TCP) operation (e.g., timeout and Round-Trip Time (RTT) estimates), the number of data packets which can be grouped and used as symbols for encoding is rather limited.

Under this configuration of a small or limited number of symbols for coding in LT codes, mathematical analysis and simulation results have revealed that LT codes can perform poorly, where the packet overhead ϵ can easily reach 100%. This renders LT codes impractical for applications where the number of symbols is small. By analyzing the rank of the coefficient matrix produced by LT encoding, we have found that LT encoding produces a full-rank coefficient matrix with a relatively high probability, which indicates the high decoding potential of LT codes even when the number of symbols is small. However, this potential is suppressed by the design of the LT decoding process where it often terminates prematurely even when the received packets already form a full-rank coefficient matrix.

1.2.2 LT codes with unequal loss protection

LT codes perform very well for data delivery. However, when dealing with progressive or layered media content such as JPEG-2000 [24] coded images or H.264/SVC [25] coded videos, LT codes have some limitations. In layered media content, lower layers are always more important than higher layers. As shown in Fig. 1.1, a certain layer i is useless unless all the lower layers from layer 1 to layer $i - 1$ are correctly received. LT codes treat all the source packets equally. It does not provide prioritized delivery for different layers. If LT

codes encode all the layers for transmission, a low-end receiver targeted for layer 1 has to collect the number of packets sufficient to decode all the layers in order to just obtain the original layer 1 data, which results in an unnecessary long delay. In other words, LT codes do not provide intermediate decoding outputs although intermediate layers do exist in layered media content.

Figure 1.1: An example of the layered structure.

1.2.3 LT codes in cloud storage system

While most of the researches of applying LT codes to storage system [13, 14, 15, 16, 17] focused on system reliability and availability, another important problem raised in storage system is data retrieval performance. Unlike those fixed-rate codes which do not need additional packets for decoding, to recover information encoded by LT codes, a storage collector requires additional encoded packets from the storage system to ensure a high probability of successful decoding for practical reasons. These additional encoded packets add delay in storage retrieval. Thus, a balance between the successful decoding probability and storage retrieval delay is important for the overall system performance.

1.3 Summary of Contributions

The major contributions of this thesis include:

- We explain why an LT decoder often fails when the number of input symbols is small, and analyze the complexity of the LT-W algorithm [26] proposed by our

research group. Also, we conduct more experiments with different parameters to confirm the performance of LT-W.

- We propose a joint Unequal Loss Protection and LT coding (ULP-LT) scheme for layered media delivery, where different numbers of Forward Error Correction (FEC) are allocated to different layers to guarantee that a lower layer can always be decoded before decoding its higher layers and LT codes are used to deal with the varying channel conditions. Also, we analyze the feasibility of joint FEC and LT decoding and propose a simple method for FEC allocation.
- We study the tradeoff between the successful decoding probability and storage retrieval delay and introduce a multiple stage storage retrieval scheme where we show that storage retrieval delay can be reduced without sacrificing the performance on decoding probability. We develop a model describing multi-stage request scheme for LT codes based storage system. What's more, we solve the optimization problem for optimal two-stage request scheme and provide closed-form expression for analysis.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In chapter 2, we give a rough literature review about LT encoding and decoding algorithms, unequal loss protection for scalable content and storage systems. In chapter 3, we present LT-W algorithm which integrates Wiedemann solver into LT decoder to improve the performance of LT codes with small number of source symbols and give a rigorous analysis on the complexity of LT-W. In chapter 4, we introduce our ULP-LT scheme which combines FEC and LT codes to provide unequal loss protection for scalable content. In chapter 5, we propose our multi-stage retrieval scheme for LT based cloud storage system and study the tradeoff between

the successful decoding probability and storage retrieval delay. Finally, we summarize this thesis and discuss the directions of our future work.

Chapter 2

Literature Review

In this chapter, we give a literature review about the related works. To begin with, we introduce encoding and decoding algorithms for LT codes together with some works on improving the decoding performance. Next, we review the works on applying LT codes to scalable content. Finally, we review the works on cloud storage systems.

2.1 LT Codes

2.1.1 LT encoding and decoding

In this section, we describe the LT encoding and decoding processes.

To generate an LT encoded packet, a degree d is first randomly determined according to a degree distribution. Next, d different symbols are chosen randomly. Finally, all chosen symbols are combined by bitwise XOR operation to generate the encoded packet. The generated encoded packet can then be placed in the output buffer for transmission to one or more clients. The sender can repeat the same procedure to generate more encoded packets. The sender stops the process when it is confident that the clients have collected enough packets to decode all symbols. A client may also make use of a feedback channel, if available, to explicitly acknowledge the sender to stop the encoding process when all

symbols have been decoded.

In the encoding process, it is shown that the degree distribution can significantly influence the performance of LT codes [10]. The original LT codes [10] make use of the robust Soliton distribution defined as follows. Let $R = \lambda \log(k/\delta)\sqrt{k}$ for some suitable constants $\delta, \lambda > 0$. Define that

$$\tau(i) = \begin{cases} \frac{R}{ik}, & i = 1, \dots, \frac{k}{R} - 1 \\ R \log(\frac{R}{\delta})/k, & i = \frac{k}{R} \\ 0, & i = \frac{k}{R} + 1, \dots, k \end{cases}$$

$$\rho(i) = \begin{cases} \frac{1}{k}, & i = 1 \\ \frac{1}{i(i-1)}, & i = 2, \dots, k. \end{cases}$$

Adding the ideal Soliton distribution $\rho(i)$ to $\tau(i)$, the degree distribution, Ω_d , is obtained by applying normalization, that is

$$\Omega_d = \frac{\rho(d) + \tau(d)}{\beta} \quad (\text{Eq. 2.1})$$

$$\beta = \sum_{i=1}^k \{\rho(i) + \tau(i)\}.$$

In Robust Soliton Distribution Ω_d , the parameter δ controls the failure probability of LT decoding and the expected number of encoded symbols for successfully decoding. The smaller δ is, the lower failure probability is whilst the larger number of encoded symbols needed. More rigorous analysis on this can be found in [10].

Algorithm 1 summarizes a general LT decoding process. The process first identifies received packets with degree one, where each of these packets is a symbol. These symbols are placed into a buffer denoted as a *ripple*. Then, each of these symbols r_j is taken out from the ripple to decode all the other packets with a degree more than one. For a packet c_i , if it contains r_j , an XOR operation is performed between r_j and c_i which results in reducing the degree of c_i by one as the symbol r_j is eliminated from packet c_i . Moreover, after the XOR operation, if the degree of c_i becomes one, then c_i is reduced to a symbol. This newly recovered symbol is then moved into the ripple if it is not found in the ripple.

Algorithm 1 A general LT decoding process

```

1: release all degree-one symbols to the ripple
2: while the ripple is not empty
3:   remove a symbol  $r_j$  from the ripple
4:   for all encoded packets  $c_i$ 
5:     if  $c_i$  includes  $r_j$ 
6:        $c_i := c_i \oplus r_j$ 
7:       reduce degree of  $c_i$  by one
8:       remove  $r_j$  from  $c_i$ 's set of input symbols
9:     end if
10:    if degree of  $c_i$  is 1 &  $c_i$  is not in ripple
11:      put  $c_i$  into ripple
12:    end if
13:  end for
14: end while

```

The process of taking a symbol out from the ripple for decoding other packets and placing newly recovered symbols into the ripple is repeated until the ripple is exhausted. The decoding process fails if there is at least one symbol yet to be discovered when the ripple is empty. Otherwise, the process is said to be successful since all symbols are decoded.

To further illustrate LT encoding and decoding process, we present a toy example in Fig 2.1. In this figure, Panel (a) demonstrates an example of LT encoding procedure with 4 binary source symbols (*i.e.* $k = 4$), denoted by $S_1, S_2, \dots, S_4$, and 5 LT encoded packets (*i.e.* $N = 5$), denoted by $P_1, P_2, \dots, P_5$ with value 10110. In this example, P_2 has degree equal to 3 and it is generated by mixing S_1, S_3 and S_4 . In LT decoding process, at the first iteration the only encoded packets with degree equal to 1 is P_3 (refer to Panel (a)). Thus, the source symbol S_2 which is connected to P_3 is set to 1 (refer to Panel (b)). After S_2 is recovered, the encoded packets which have connection with S_2 are extracted by the value of S_2 and the connections are removed (refer to Panel (c)). After this step, a new symbol S_4 can be recovered and the same decoding procedure is executed iteratively until no more new symbol can be found.

Figure 2.1: An example of LT encoding and decoding procedures for $k = 4$ and $n = 5$.

In this example, all the symbols are recovered (refer to Panel (h)) and we say LT decoding is successful. If we only have encoded packets $P_1, P_2, \dots, P_4$, the decoding process would fail at Panel (b) since there is no packet with degree 1. However, the coding matrix formed by P_1, P_2, P_3, P_4

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

is full rank which means the remaining symbols can be recovered by other methods such as Gaussian elimination [27]. However, LT decoding algorithm just suppresses this possibility and reports a decoding failure.

2.1.2 The Success Probability of Decoding

From section 2.1.1, we know that LT decoding may fail when the obtained encoded packets are not sufficient. In this section, we introduce the analysis on success probability of decoding presented in [28]. In the model, a three dimensional state is used to describe the evolution of the LT decoding process. Each state is described by three parameters, namely (i) the number r of symbols with degree one (i.e. symbols in the ripple), (ii) the number c of symbols with degree two and above (i.e. the number of symbols in the cloud), and (iii) the number u of unrecovered symbols. Let $P(r, c, u)$ denote the probability that the LT decoder is in state (r, c, u) . The decoder is said to fail if it has reached a state where $u > 0$ and $r = 0$. Given the degree distribution Ω_d , k symbols and n received encoded packets, the probability of successful decoding can be expressed as

$$S_{k,n}^{LT} = 1 - \sum_{0 < u \leq k, r=0} P(r, c, u). \quad (\text{Eq. 2.2})$$

Karp *et al.* [28] has derived the state probability for the LT decoding process as shown in (Eq. 2.3). The equation describes the transition behavior from states $(r+1+s-t, c+$

t, u) where $t-s \leq r$ and $s, t \geq 0$ to state $(r, c, u-1)$, and $\Psi((r+1+s-t, c+t, u); (r, c, u-1))$ is the transition probability. This transition represents the decoding process of taking out a symbol from the ripple that reduces t cloud packets to degree one, among which s symbols are duplicated. Note that p_u is the releasing probability, i.e., the probability of releasing a degree-one symbol from the cloud packets given the number u of unrecovered symbols.

$$P(r, c, u-1) = \sum_{\substack{s, t \geq 0 \\ t-s \leq r}} P(r+1+s-t, c+t, u) \Psi((r+1+s-t, c+t, u); (r, c, u-1)), \quad (\text{Eq. 2.3})$$

$$\Psi((r+1+s-t, c+t, u); (r, c, u-1)) = \binom{c+t}{t} p_u^t (1-p_u)^c \binom{r+s-t}{s} (1/u)^s (1-1/u)^{r-t},$$

$$p_u = \frac{\sum_d \Omega_d \cdot d(d-1) \cdot \frac{(u-1)(k-u) \dots (k-u-d+3)}{k(k-1) \dots (k-d+1)}}{1 - \sum_d \Omega_d \cdot \frac{(k-u) \dots (k-u-d+1)}{k(k-1) \dots (k-d+1)} - \sum_d \Omega_d \cdot d \cdot \frac{u(k-u) \dots (k-u-d+2)}{k(k-1) \dots (k-d+1)}}.$$

With the recurrence equation in (Eq. 2.3), the success probability of decoding with respect to the number of received packets can be determined. Fig. 2.2 shows the success probability of decoding for the cases of $k = 30$ and $k = 60$, under $\lambda = 0.2$ and $\delta = 0.05, 0.1$, respectively, where λ and δ are the configuration parameters for the degree distribution defined in (Eq. 2.1). Note that the configuration parameters used in the illustration are selected with no particular preference. It is clear from Fig. 2.2 that the packet overhead, $\epsilon = (n-k)/k$, can be as large as 100% for a 95% probability of recovering all symbols.

2.2 LT-W: Improved LT Decoding with Wiedemann Solver

By observing that the coefficient matrix is full rank even when LT decoder reports failure, in [26], Wiedemann solver is introduced by Lu *et al.* from our research group, to *borrow* a

 2.2.a: $k = 30$

 2.2.b: $k = 60$

Figure 2.2: The success probability of LT decoding with respect to the number of received packets.

symbol and restart the LT decoding process. In this mechanism, when the ripple becomes empty and the coefficient matrix of the encoded packets is full rank, a particular symbols is borrowed by using the Wiedemann's method, and this borrowed symbol is then placed into the ripple for the LT decoding process to continue. In other words, when LT decoding process stalls, Wiedemann's method is used to recover a symbol. This procedure is repeated until the LT decoding process terminates with a success. The aforementioned mechanism is named LT-W decoding.

As this work is closely related to the work presented in Chapter 3, for the consideration of completeness, we shall introduce more details in the following.

2.2.1 Deciding which symbol to borrow

The following criterion is used to decide which symbol should be borrowed. To decode K symbols with N packets¹, given the above analogy between a packet and a linear equation, let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_N$ denote the coefficient vectors with the entries in $GF(2)$ at

¹In this section, uppercase letters are used to denote these quantities that will be evolved during the decoding process. Initially, $K = k$ and $N = n$. When more symbols are recovered, both K and N will decrease.

the intermediate decoding process when the ripple is empty; and let b denote the index of the borrowed symbol, which is defined as:

$$b = \arg \max_j \left(V(j) | \mathbf{V} := \sum_{i=1}^N \mathbf{v}_i \right). \quad (\text{Eq. 2.4})$$

The above sum operation is in the real number field, and $V(j)$ gives the value at index j of the vector $\mathbf{V}$. The basic idea here is to choose the symbol that is carried by most packets. Incorporating the borrowed symbols into the LT decoding, LT-W decoding is shown in Algorithm 2.

Algorithm 2 LT-W Decoding

```

1: rank( $\mathbf{M}$ ):= $k$ , ripple:  $r$ , borrowed symbol buffer:  $B$ 
2: while not all input symbols are recovered
3:   release all degree one symbols to the ripple
4:   if ripple is empty
5:     compute the index of borrowed symbol  $b$  as in (Eq. 2.4)
6:     put index  $b$  into buffer  $B$ 
7:     recover the borrowed symbol
8:     release the borrowed symbol into ripple
9:   end if
10:  remove a symbol  $r_j$  from the ripple  $r$ 
11:  for all encoded packets  $c_i$ 
12:    if  $c_i$  includes  $r_j$ 
13:       $c_i := c_i \oplus r_j$ 
14:      update  $\mathbf{v}_i(j)$ :=0 and reduce degree of  $c_i$  by one
15:    end if
16:  end for
17: end while

```

2.2.2 Recovering the Borrowed Symbol

One of the critical designs in LT-W decoding is the recovery of a borrowed symbol. A suitable method that can recover only a single symbol at a low computational cost is needed. Common techniques such as Gaussian elimination are inadequate as they are

designed to recover all the symbols and often incur high computational complexity. The following method is used in LT-W decoding.

Let $\mathbf{M}$ denote the coefficient matrix. $\mathbf{M}$ is of size $N \times K$ for decoding K symbols with N packets. For a clearer exposition of the proposed idea, the case of $N = K$ is discussed firstly. The case of $N > K$ will be discussed later. The task of decoding is essentially to solve the following linear system

$$\mathbf{M}\mathbf{x} = \mathbf{y} \quad (\text{Eq. 2.5})$$

where $\mathbf{x}$ is a $K \times l$ vector holding the symbols, $\mathbf{y}$ is a $N \times l$ vector holding the received packets, and l is the length of a symbol. The matrix $\mathbf{M}$ is defined over $GF(2)$. Since the length of a symbol (i.e. the length of a packet) is typically a large value, the demand for computing power is generally high if the entire message set $\mathbf{x}$ needs to be solved for.

The idea of borrowing a symbol for LT decoding process does not, however, require solving the entire message set. This allows us to consider a solver that deals with solving only for a single symbol at a much lower computational cost. Since $\mathbf{M}$ is of rank K , a vector over $GF(2)^K$ can be constructed by linear combinations of a proper collection of the row vectors of $\mathbf{M}$. In order to solve for a symbol located at index i , the idea is to find the right combination to yield the following $K \times 1$ unit vector $\mathbf{u}_i$, where the unique one is located at the index i . Mathematically, the problem can be expressed as solving for $\mathbf{x}'$ given

$$\mathbf{M}_t \mathbf{x}' = \mathbf{u}_i \quad (\text{Eq. 2.6})$$

where $\mathbf{M}_t = \mathbf{M}^T$ and $\mathbf{x}'$ is the right combination. The borrowed symbol can then be easily obtained by the inner product of $(\mathbf{x}', \mathbf{y})$ which is defined as $\sum_{j=1}^N x'_j y_j$. Note that the computational complexity required to solve (Eq. 2.6) is much lower than that of (Eq. 2.5) as $\mathbf{M}_t, \mathbf{x}', \mathbf{u}_i$ are all defined in $GF(2)$ and only bit operations are needed.

The method for solving (Eq. 2.6) is mainly based on the Wiedemann's method [29] due to its low complexity. Firstly, the vector $\mathbf{u}_i$ is used to generate the so-called Krylov sequence

$$\mathbf{u}_i, \mathbf{M}_t \mathbf{u}_i, \mathbf{M}_t^2 \mathbf{u}_i, \dots, \mathbf{M}_t^K \mathbf{u}_i, \dots$$

Let S be the space spanned by this sequence, where $\mathbf{M}_t$ is a nonsingular matrix. Let M_s denote the operator $\mathbf{M}_t$ restricted to S , and define $f(z)$ as the minimal polynomial of M_s , where the coefficient of $f(z)$ is in $GF(2)$. Let $d = \deg(f)$ and $f[i]$ denote the coefficient of z^i in $f(z)$. Provided that $f(z)$ can be found, the solution to (Eq. 2.6) can be quickly obtained by

$$\mathbf{x}' = \sum_{i=1}^d f[i] \mathbf{M}_t^{i-1} \mathbf{u}_i. \quad (\text{Eq. 2.7})$$

The algorithm to solve for the borrowed symbol (see Algorithm 3 [29]) is introduced in the following. The first few steps presented in Algorithm 3 are self-explanatory. For Step 5, given a polynomial $g(z)$ of degree d and a sequence $\{s_i\}_{i=0}^l$, let

$$h(z) = \sum_{i=0}^l s_i z^i$$

and by definition the result of applying $g(z)$ to the sequence $\{s_i\}_{i=0}^l$ is

$$\{(g(z^{-1})h(z))[i]\}_{i=0}^{l-d}.$$

By apply the Berlekamp Massey algorithm [30, 31] in Step 6, find the minimal polynomial can be found. The readers can refer to chapter 6 in [32] for a practical algorithm particularly designed for sequences defined in $GF(2)$.

2.2.3 Non-square Case

In practice, it is likely that more than K packets have to be received in order to ensure complete recovery of all the symbols. As a result, the coefficient matrix $\mathbf{M}$ will be

Algorithm 3 Solving the borrowed symbol

- 1: compute $\mathbf{M}_t^i \mathbf{u}_i$ for $i = 0, 1, \dots, 2K - 1$
 - 2: set $c = 0$ and $g_0(z) = 1$
 - 3: set $\mathbf{u}_{c+1}$ to be the $(c + 1)^{th}$ unit vector
 - 4: extract from the results of step 1 the sequence $(\mathbf{u}_{c+1}, \mathbf{M}_t^i \mathbf{u}_i)_{i=0}^{2K-1}$
 - 5: apply $g_c(z)$ to this sequence
 - 6: set f_{c+1} to be the minimal polynomial of the sequence produced in step 5
 - 7: update $g_{c+1} = f_{c+1}g_s$
 - 8: set $c = c + 1$; if $\deg(g_c) < K$ and $c < K$, go to step 3
 - 9: use $f = g_c$ and compute $\mathbf{x}'$ as in (Eq. 2.7)
 - 10: the borrowed symbol is $(\mathbf{x}', \mathbf{y})$
-

non-square, i.e., the matrix is of size $N \times K$ with rank equal to K and $N > K$. A straightforward method to obtain a square matrix is to randomly select a sub-square matrix from the non-square matrix until a full rank matrix is found. However, this is very inefficient, especially when the non-square matrix is far from being a square one.

The approach is to find a $N \times K$ matrix $\mathbf{M}_c$ such that $\mathbf{M}_t \mathbf{M}_c$ will be of full rank. One way to obtain $\mathbf{M}_c$ is to randomly set an entry of row i in $\mathbf{M}_c$ to one with probability $p_i = \min\left(\frac{1}{2}, 2 \log \frac{N}{K+1-i}\right)$. According to [30], with the probability of at least 12.5%, the $K \times K$ matrix $\mathbf{M}_t \mathbf{M}_c$ is full rank. In this way, with a few rounds of trials, $\mathbf{M}_c$ can be found.

Now, introducing $\mathbf{M}_c$ into the operation, (Eq. 2.6) becomes

$$\mathbf{M}_t \mathbf{M}_c \mathbf{x}' = \mathbf{u}_i. \quad (\text{Eq. 2.8})$$

Note that here $\mathbf{M}_c \mathbf{x}'$ represents the appropriate combinations of the rows in $\mathbf{M}$ to yield the unit vector $\mathbf{u}_i$, and hence the borrowed symbol can be decoded accordingly.

2.2.4 Analysis of Random Matrix Rank

It is understood that the success probability of LT-W decoding algorithm depends on the probability that the coefficient matrix $\mathbf{M}$ formed by the received packets is of full

rank. In other words, as long as $\mathbf{M}$ can reach full rank, the LT-W decoding algorithm can guarantee a success in decoding. Studying the rank in a random matrix directly gives insight to the performance of LT-W decoding algorithm.

Let $\mathbf{v}_i$ be the i^{th} coefficient vector (i.e. the i^{th} row vector of $\mathbf{M}$). Precisely, considering k symbols and n received packets, $\mathbf{M} = (\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n)^T$, where $\mathbf{v}_i$ is of size $k \times 1$ and $\mathbf{M}$ is of size $n \times k$. The probability that $\mathbf{v}_i$ gives a particular bit pattern, $\mathbf{v} \in GF(2)^k$ of degree d is

$$\Pr\{\mathbf{v}_i = \mathbf{v}\} = \frac{\Omega_d}{\binom{k}{d}}, \quad \mathbf{v} \in GF(2)^k, d = \deg(\mathbf{v}) \quad (\text{Eq. 2.9})$$

where Ω_d is given in (Eq. 2.1) that describes the probability of choosing degree d to encode a packet.

2.2.4.1 Random Matrix Rank when $n = k$

To begin with, the special case of $n = k$, where $\mathbf{M}$ is a $k \times k$ matrix over $GF(2)$ is considered. Its row vectors are linearly dependent if there exists a nonzero vector $(c_1, \dots, c_k) \in GF(2)^k$ that satisfies

$$\sum_{i=1}^k c_i \mathbf{v}_i = \mathbf{0}.$$

If $\mathbf{M}$ is said to have a full rank, any linear combination of coefficient vectors $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k)$ will not produce $\mathbf{0}$. Consider a non-zero vector $\mathbf{c} = (c_1, \dots, c_k) \in GF(2)^k$, with exactly q non-zero coordinates. Without loss of generality, let us assume that the first q coordinates of $\mathbf{c}$ are non-zero. Denote P_q to be the probability that $\sum_{i=1}^q c_i \mathbf{v}_i = \mathbf{0}$. Clearly, it is the same as the probability of the condition that $\sum_{i=1}^{q-1} \mathbf{v}_i = \mathbf{v}_q$ because the operations are defined over $GF(2)$.

Suppose that summing the first q vectors results in a vector with degree i , i.e., a vector that carries an i number of 1s. The conditions that its degree changes to j with

one additional vector added is analyzed. As illustrated in Fig. 2.3, the degree of the $(q+1)^{th}$ vector, $\mathbf{v}_{q+1}$, is computed as

$$\deg(\mathbf{v}_{q+1}) = a + b \quad (\text{Eq. 2.10})$$

where $b = j - i + a$ and $0 \leq a \leq i, 0 \leq b \leq (k - i)$. Thus, the probability that $\mathbf{v}_{q+1}$ of degree $(a + b)$ happens to be in the arrangement as revealed in Fig. 2.3 is $\frac{\binom{i}{a} \cdot \binom{k-i}{b}}{\binom{k}{a+b}}$.

Figure 2.3: An example of the degree changing from i to j with the $(q+1)^{th}$ vector added. Note that the dark region indicates the places of 1s.

Based on the above analysis, the state transition probability P_{ij} can be derived, i.e. the probability that the sum of $(q+1)$ vectors has degree j given that the sum of the first q vectors has degree i ,

$$P_{ij} = \begin{cases} \sum_{\substack{0 \leq a \leq \min(k-j, i) \\ b = j-i+a}} \Omega_{a+b} \cdot \frac{\binom{i}{a} \cdot \binom{k-i}{b}}{\binom{k}{a+b}}, & i < j \\ \sum_{\substack{1 \leq a \leq \min(k-j, i) \\ b = j-i+a}} \Omega_{a+b} \cdot \frac{\binom{i}{a} \cdot \binom{k-i}{b}}{\binom{k}{a+b}}, & i = j \\ \sum_{\substack{i-j \leq a \leq \min(k-j, i) \\ b = j-i+a}} \Omega_{a+b} \cdot \frac{\binom{i}{a} \cdot \binom{k-i}{b}}{\binom{k}{a+b}}, & i > j. \end{cases} \quad (\text{Eq. 2.11})$$

The very last result describes the evolution of degree when an additional vector is added. This allows us to determine the degree distribution of the sum of any number of vectors. Now, define a transition matrix $\mathbf{Tr}$ with dimension $(k+1) \times (k+1)$, and each element $\mathbf{Tr}_{ij} = P_{(i-1)(j-1)}$. Let $\mathbf{\Omega}^q$ denote the degree distribution of the sum of q vectors where $q \geq 1$. Then the following equation can be established

$$\mathbf{Tr}^{q-1} \cdot (\Omega_0, \Omega_1, \dots, \Omega_k)^T = \mathbf{\Omega}^q \quad (\text{Eq. 2.12})$$

where $\mathbf{\Omega}^q$ can be determined numerically.

The next target is finding P_q , i.e., the probability of the condition that

$$\sum_{i=1}^{q-1} \mathbf{v}_i = \mathbf{v}_q. \quad (\text{Eq. 2.13})$$

In order to satisfy (Eq. 2.13), the summation on the LHS must give a vector that has the same number of 1s as that of $\mathbf{v}_q$, and these 1s should be located at the same positions. Knowing that a vector of degree d can have $\binom{k}{d}$ possible forms, the probability for (Eq. 2.13) to hold can be derived as

$$P_q = \sum_{d=0}^k \frac{\Omega_d \cdot \Omega_d^{q-1}}{\binom{k}{d}}. \quad (\text{Eq. 2.14})$$

If $\mathbf{M}$ is of full rank, then any combination of the coefficient vectors should not result in a linear dependency. Thus, the probability of obtaining a full rank can be determined by

$$P_f(k, n) = \prod_{q=2}^k (1 - P_q)^{\binom{k}{q}} \quad (\text{Eq. 2.15})$$

where $k = n$ in this case, and there are $\binom{k}{q}$ vectors with exactly q non-zero coordinates.

2.2.4.2 Random Matrix Rank when $n > k$

This subsection extends the previous analysis of random matrix rank with $n = k$ to the case of $n > k$. Additional handling must be introduced since (Eq. 2.15) is derived

$$\begin{pmatrix} Q(q, 1) \\ Q(q, 2) \\ \vdots \\ Q(q, k-1) \\ Q(q, k) \end{pmatrix} = \begin{pmatrix} 1 - Q_{o1} & & & & \\ Q_{o1} & 1 - Q_{o2} & & & \\ & Q_{o2} & 1 - Q_{o3} & & \\ & & \ddots & \ddots & \\ & & & Q_{o(k-1)} & 1 - Q_{ok} \end{pmatrix}^{(q-1)} \cdot \begin{pmatrix} Q(1, 1) \\ Q(1, 2) \\ \vdots \\ Q(1, k-1) \\ Q(1, k) \end{pmatrix} \quad (\text{Eq. 2.17})$$

based on the condition that for a full rank matrix no linear dependency exists for any combination of the row vectors, which is not true for the case of $n > k$. This is because an $n \times k$ coefficient matrix $\mathbf{M}$ that gives full rank of k consists of k linearly independent row vectors. When $n > k$, $n - k$ row vectors are redundant. The following analysis includes the consideration of possibly redundant row vectors.

Let (q, r) denote the state of the coefficient matrix $\mathbf{M}$ where $\mathbf{M}$ consists of q row vectors with rank r , and $Q(q, r)$ denote the corresponding state probability. The objective is to derive the state probability of $Q(n, k)$. The evolution of the rank of $\mathbf{M}$ can be described by the following recurrent relationship, i.e.,

$$\begin{aligned} Q(n, r) &= (1 - Q_{or})Q(n-1, r) + Q_{o(r-1)}Q(n-1, r-1), \\ 1 < r \leq k, Q(1, 1) &= 1, Q_{ok} = 0, Q(n, r) = 0, \text{ if } n < r \end{aligned} \quad (\text{Eq. 2.16})$$

where Q_{or} is the probability that a random vector is linearly independent to r other linearly independent row vectors. It is easy to see that $Q_{or} = \frac{P_f(r+1, r+1)}{P_f(r, r)}$.

With the establishment of the state recurrent relationship, the system state probabilities can then be determined by solving (Eq. 2.17). Note that $(Q(1, 1), Q(1, 2), \dots, Q(1, k))^T$ is initialized to $(1, 0, 0, \dots, 0)^T$. Since the transition matrix shown in (Eq. 2.17) is of full rank, methods such as the eigen decomposition or companion matrix and the Jordan normal form [33] can be utilized to derive a close form expression for the probability of obtaining a full rank with n encoded packets received with respect to k original packets $P_f(k, n) = Q(k, n)$.

By setting the number of input symbols to be 30 and 60, the numerical results for the probability of obtaining a full-rank $\mathbf{M}$ with respect to the number of packets received ($\lambda = 0.2, \delta = 0.1$ and $\lambda = 0.2, \delta = 0.05$) is presented in Fig. 2.4. As can be seen from the results presented in Fig. 2.4, the probabilities of obtaining a full-rank $\mathbf{M}$ climb quickly as soon as k packets are received. Comparing with the results shown in Fig. 2.2 for the original LT decoding algorithm, for a given success probability of decoding, LT-W decoding algorithm uses about 30% fewer number of received packets to decode the same number of symbols.

2.4.a: $k = 30$ 2.4.b: $k = 60$

Figure 2.4: The probability of obtaining a full-rank coefficient matrix with respect to the number of received packets.

2.3 Other Work on Improving LT Decoding Performance

The extremely high packet overhead associated with LT codes for a small number k of symbols has also been reported in other studies [34, 35, 36, 37]. In [34], Hyytia *et al.* attempted to optimize the configuration parameters of the degree distribution for LT codes with small numbers of symbols. They proposed two approaches, namely, a Markov chain method and a combination method. However, as noted in the paper, the

methods are not scalable, and can only handle symbols $k \leq 10$ (numerically up to 30 for the combination method). In [35], Liang *et al.* proposed a new degree distribution for encoding but they can only slightly reduce the overhead. In [36], Studholme and Blake defined a completely new degree distribution function and applied the Gaussian elimination method [27] for decoding. Although the packet overhead ϵ is reduced, the decoding complexity has a quadratic order of $O(k^{3/2})$. In [37], Bodine and Cheng analyzed the impact of the parameter selection, e.g. δ and λ in the robust Soliton distribution, on the decoding performance of LT codes and gave a suggestion on how to set these parameters when the number of symbols is small. However, the improvement achieved with such a heuristic scheme was limited.

In [38], Kokalj-Filipovic *et al.* proposed Doped Fountain Coding where a doping mechanism is integrated into the original LT decoder for data collection in circular networks. The idea in [38] is to pull a raw symbol whenever the decoder fails. This scheme enjoys a short delay and a small number of pulled symbols. We recognize that the proposed Doped Fountain Coding has an advantage over other solutions as it avoids transmissions of excessive encoded packets, and its proposed packet doping process ensures information usefulness for decoding and maintains a low delay. However, this solution relies on the availability of a feedback channel.

In [39], the authors proposed inactivation decoding to reduce packet overhead for LT codes with small number of input symbols. The idea of inactivation decoding is introduced through an example in the following. We rewrite the coding matrix at the end of 2.1.1 here:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{pmatrix}.$$

Using LT decoding algorithm, we start to identify a row with degree 1, remove it from other rows, and continue with identifying another row with degree 1. In the above

example, this process can only recover $S_3 = P_3$. We rearrange the equations as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} * \begin{pmatrix} S_2 \\ S_1 \\ S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} P_3 \\ P_1 \\ P_2 \\ P_4 \end{pmatrix}.$$

Now, we inactive S_1 that is the second column can be removed. After that, S_3 and S_4 can be recovered. Once again, we rearrange the equations to reflect the order of symbols recovered and perform the removing operation.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} * \begin{pmatrix} S_2 \\ S_3 \\ S_4 \\ S_1 \end{pmatrix} = \begin{pmatrix} P_3 \\ P_1 \\ P_2 \\ P_4 \end{pmatrix} \implies$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} S_2 \\ S_3 \\ S_4 \\ S_1 \end{pmatrix} = \begin{pmatrix} P_3 \\ P_1 \\ P_2 \oplus P_1 \\ P_4 \oplus P_2 \oplus P_1 \end{pmatrix}.$$

From the above, we can find that $S_1 = P_4 \oplus P_2 \oplus P_1$, $S_2 = P_3$, $S_3 = P_1 \oplus S_1$ and $S_4 = P_2 \oplus P_1 \oplus S_1$. Substituting the values of P_1 to P_2 , we can obtain the value of S_1 to S_4 . Interested reader can refer to [39] for more details.

In Chapter 3, we introduce a decoding method that can reboot LT decoding process whenever the ripple is empty. The idea is similar to inactivation decoding but we use different technique to solve the decoding problem.

2.4 Work on Providing Unequal Loss Protection to LT Codes

From the toy example presented in section 2.1.1, we can see that when a source symbol can be recovered is determined by the received encoded packets. Different sets of encoded packets will give different orders of recovered symbols. This phenomenon does not

affect disseminating content without priority. However, in prioritized content, important symbols always require faster recovering.

Conventionally, Unequal Loss Protection(ULP) can be achieved by joint source channel coding, in which more important contents are encoded with stronger codes [40, 41, 42, 43]. Those work assume that channel state information is available at the transmitter side and keeps stable during transmission. When the channel state is unavailable, joint source channel coding targets the worst-case error rate as the design criterion in choosing the optimal code rates citech2nosratinia2003source. Designing codes targeting the worst channel state will typically waste bandwidth. To overcome this shortcoming, Huo *et al.* proposed bit-level inter-layer forward error correction scheme in [44]. In their design, bits of base layer and bits of enhancement layers are combined together through XOR operation. This allowed the receiver to exploit the implanted of enhancement layers for assisting the base layer's decoding. A short tutorial on this inter-layer FEC scheme can be found in [45]. In fact, such kind of combination borrows the idea from rateless codes.

Rateless codes such as LT codes do not assume any information about the channel and is self-adaptive to state-varying channel through tuning the sending rate. However, LT codes cannot provide ULP for important symbols.

Some methods have been proposed to generate intermediate results for prioritized content delivery using LT codes. In [46], the sender first transmits the original packets followed by parity packets generated by LT codes. The generated parity packets are carefully designed to give important packets higher priority. The performance of this scheme very much depends on the design of the degree distribution used in generating parity packets. Inspired by [46], in [47], Chang *et al.* proposed to tune the coding graph of LT codes so that packets belonging to lower layers can have more decoding paths with short length and thus can be decoded more quickly. In [48], Arslan *et al.*

introduced a systematic degree-dependent selection concept that can provide ULP to rateless codes through assigning weight to different priority contents and encoding those contents based on weight. Zao *et al.* proposed another way to find the optimal degree distribution through evolution strategies to achieve ULP for LT codes in [49]. Although those methods can achieve ULP for different layers, the design of the degree distribution is content-dependent. In other words, whenever there is a new layered content, the degree distribution of LT codes needs to be redesigned. Simon and Michael proposed prioritized LT codes in [50]. In their design, important symbols are encoded with degree 1 and 2 only such that these symbols can be quickly recovered. However, the probability of generating two identical encoded packets is increased. The encode must have a filter to remove these identical packets to avoid bandwidth waste. The designing concept of those above mentioned work is tuning the degree distribution to achieve ULP. Many other work follow this concept [51, 52, 53, 54].

Another designing wisdom for achieving ULP is through pre-coding or two-layer coding. In [55], the authors presented a novel code call Expanding Window Fountain (EWF) code to address the ULP problem. In EWF codes, symbols belong to different coding windows. Encoding in one coding window is independent from encoding in other coding windows. The most important symbols belong to all coding windows while the least important symbols belong to only one coding windows. In this way, the probability of encoded packet containing important symbols is higher. Thus, the important symbols can be recovered earlier than other symbols. This windowed coding concept can be easily extended to other codes to achieve ULP property. In [56], the authors applied windowed coding fashion on random linear codes and discussed the impact of non-overlapped and overlapped window on performance. Alternatively, Ni [57] *et al.* use duplicated window to repeat important content more times and then apply non-overlapped window encoding

to achieve ULP. Chen *et al.* proposed Progressive Rateless Code (PRC) in [58] to protect high priority contents. In their design, different layers of contents are first encoded separately, then different layers of encoded contents are mixed together using FEC for transmitting, which is similar to [C3]. The method proposed in [59] also follows this similar idea.

Those above mentioned work make the more important layer (base layer) have a higher decoding probability through different techniques. However, they cannot guarantee that an important layer can always be decoded before other layers (enhancement layers). In practice, the enhancement layers are useless if the base layer is not ready. We address this problem in Chapter 4.

2.5 Work on Cloud Storage Systems

Cloud storage systems provide a scalable online storage solution to end users who require flexible amount of storage space but do not wish to own and maintain storage infrastructure [60, 61]. Comparing with traditional data storage, cloud storage has several advantages. For example, end user can access their data anywhere through Internet without bothering about carrying physical storage medias. Also, different users can collaboratively contribute to the data stored in cloud storage with permission from the data owner. Due to its high popularity in industry, cloud storage systems has been a hot topic [62] in cloud computing community.

Generally, cloud storage systems rely on thousands of storage nodes. A content is often fragmented and distributed into a set of storage nodes. To offer high reliability and availability of storage services, redundancy of contents may be employed. Fragments of contents may be simply replicated and stored in different storage nodes to achieve redundancy. Other than naive replication, Lin *et al.* [63] proposed two QoS-aware data

replication algorithms to reduce storage cost while maintain QoS for the applications. In [12], Weatherspoon *et al.* analyzed two schemes for redundancy: replication and erasure coding, concluding that erasure-coded system can provide higher availability with lower bandwidth and less storage space. Since then, there are plenty of works on designing erasure codes for storage system [64, 65, 66, 67]. An example of a successful deployment of commercial cloud storage system using erasure coding is Symform [68, 69]. In Symform, Reed-Solomon code [6] is used for redundancy generation. Its success shows erasure-coding-based cloud storage systems is practical and comparable with traditional replication-based cloud storage systems. Another research effort focuses on the design of storage system operational optimization based on existing erasure codes to achieve some specific optimal performance matrices. These operational designs may include storage redundancy overhead optimization, storage allocation and repair, and others [70, 71, 72, 73].

While those works focus on the reliability and availability of a storage system, another important problem raised in storage system is data retrieval performance. Traditional distributed file systems like NFS [74] and Sprite [75] are single server based and achieve acceptable retrieval performance through file caching. However, the retrieval performance achieved by client is bounded by the speed of single storage server. Conversely, parallel file systems, such as Vesta [76], Galley [77], PVFS [78] and GPFS [79], allow data spreading across multiple storage nodes and provide parallel access to it. The high transfer rate is reached by various techniques such as I/O scheduling, content prefetching *etc.* More recently, high retrieval performance is achieved by introducing fast storage devices like Solid-State Drivers (SSD) [80] and DRAM [81].

In [17], Cao *et al.* proposed a cloud storage system based on LT codes. This the first work on designing cloud storage system using LT codes. In this work, LT encoded content are properly placed so that the content is available by accessing a determined number of

storage nodes from the whole system. A backup node is used to store all the encoded content such that whenever a storage node fails, it can be recovered by copying the same encoded content from this backup node to new storage node. After this work, researchers begin to investigate different problems of LT-based cloud storage system. In [82], instead of using try-and-error methods, the author proposed a systematic method to generate LT encoded contents for LT-based cloud storage system. Zhang *et al.* mixed replication and LT codes in [83] for better storage efficiency by modifying the design of backup node. By borrowing the idea from AONT-RS [84] system, the authors in [85] proposed AONT-LT system for data protection. In this system, data will be first encoded with all-or-nothing transforming codes [84] and then encoded with LT codes. This scheme can achieve data privacy even if data is dispersed over large scale of distributed storage system. Also, in [86], the authors proposed a new technique for analyzing codes-based cloud storage system reliability to provide quantified comparison between different designs. In [87], the authors designed another LT-based cloud system call ENIGMA. By testing this prototype using real workload, it shows that LT-based cloud system is able to achieve suitable levels of performance, availability, integrity and confidentiality.

While most work focus on system design, few work investigate the retrieval performance for LT-based cloud storage system. As LT decoding is probabilistic, one cannot request a determined number of encoded contents to recover the raw content. In Chapter 5, we introduce a multiple-stage retrieval scheme to achieve lower retrieval delay.

Chapter 3

Performance analysis of LT-W

In this chapter, we present rigorous analysis on LT-W decoding algorithm which is introduced in Section 2.2. We shall first investigate why LT codes fail. Next, we present the complexity analysis on LT-W decoding together with the analysis on the number of borrowed symbols. Finally, we compare the performance between LT-W decoding and inactivation decoding.

3.1 Why LT codes fail

In section 2.1.2, we presented the analysis on success probability of LT decoding and we showed that when the number of symbols are small, the decoding overhead associated with LT decoder may reach 100%. We first investigate the shortcoming of high packet overhead in the LT codes, which is resulted from the LT decoding process. As mentioned in Section 2.1.1, the LT decoding process ends successfully when all symbols are recovered. In an event that the ripple is exhausted before all symbols are recovered, the decoding process stalls resulting in an unsuccessful decoding. Hence, in the process, it is important not to let the symbols in the ripple run out before all symbols are recovered. Since the arrival of new symbols into the ripple depends on the degrees of the received packets, the degree distribution used in the encoding is crucial to the success of the decoding process.

However, based on our observation, when the number of symbols is small, the actual degree distribution differs significantly from the preset distribution used for encoding. To illustrate this point, we measure the KullbackLeibler (KL) divergence [88] for a range of 30 to 100 for the number of symbols under two sets of configuration parameters (see Fig. 3.1). The preset degree distribution is robust Soliton distribution and the actual degree distribution is obtained by averaging 100 rounds of simulation with number of encoded symbols equal to number of input symbols. The large distance between the preset and actual distributions as shown in the figure is a clear evidence of the poor performance of LT codes with a small number of input symbols.

Figure 3.1: The KL divergence between the preset and actual degree distributions where exact k packets are received.

Our further investigation on the LT decoding process reveals that when the LT decoding process terminates and reports a failure, often the undecodable packets can be

decoded to recover all the symbols by other means such as Gaussian elimination. In other words, when viewing a packet as an equation formed by combining linearly a number of variables (or symbols) in $GF(2)$, the set of available equations (or packets) may yield a full rank such that a numerical solver (or decoder) can determine all the variables (or symbols). However, such a numerical solver does not exist in the LT decoding process. Whenever there are undecodable packets, a decoding failure is reported by LT decoder.

LT-W is succeed since it permits the continuation of the LT decoding process when the ripple is exhausted.

In the following sections, we investigate the performance of LT-W decoding algorithm. We focus on three performance measures, namely, the overhead packets required for a specific probability of successful decoding, the number of borrowed symbols involved in a successful decoding and the actual running time for recovering all the symbols.

3.2 Percentage of Overhead Packets

Let $S_{k,n}$ denote the success probability of decoding with n received packets, where $S_{k,n}$ follows $S_{k,n}^{LT}$ for the case of LT decoding or $S_{k,n}^{(LT-W)}$ for that of LT-W decoding. Define $n_{0.95}$ as the least number of packets required to achieve at least 0.95 of successful decoding, i.e.,

$$n_{0.95} = \arg \min_n S_{k,n} \geq 0.95 \quad (\text{Eq. 3.1})$$

and the percentage of overhead packets are calculated by $\frac{n_{0.95}-k}{k}$.

Figure 3.2 depicts the packet overhead comparison between the original LT decoding and the LT-W decoding. Two different sets of configuration parameters are used, and the number of input symbols k ranges from 30 to 100. Note that the configuration parameters are arbitrarily selected and they are not guaranteed to be optimal. From Fig. 3.2, it can

be seen that LT-W decoding has a significantly lower percentage of overhead packets compared to that of LT decoding. Notably, when $k > 60$, LT-W decoding requires less than 10% overhead while LT decoding requires over 60% overhead to achieve 95% probability of successful decoding. Moreover, we also compare the overhead required for more stringent successful decoding probability of 98%, against the number of input symbols, as depicted in Fig. 3.3. As can be seen, the overhead required of LT-W decoder is always smaller than that of the LT decoder. The overhead for the LT-W decoder quickly drops and stays just above zero for very small numbers of input symbols, demonstrating the effectiveness of our design in achieving a lower packet overhead.

Figure 3.2: The overhead for 95% probability of successful decoding.

The superior performance results of LT-W decoding depicted in Fig. 3.2 and Fig. 3.3 can be explained as follows. In LT decoding, the decoding process terminates whenever the ripple is empty. Maintaining the symbol release rate from the ripple during the decoding is critical to ensure the continuation of decoding until all the symbols are recovered. Since the symbol release rate is governed by the preset degree distribution, the variation in the actual degree distribution directly influences the decodability of LT decoding. In Section 3.1, we have illustrated that for some small numbers of symbols, the KL distance between the preset and actual degree distributions can be large. Thus the

Figure 3.3: The overhead for 98% probability of successful decoding.

desirable degree distribution is not achieved, and the decoding process often terminates prematurely. On the other hand, in LT-W decoding, as long as the coefficient matrix is of full rank, the symbols are guaranteed to be recoverable. Since the full rank property is less susceptible to the imperfectness of the actual degree distribution, LT-W decoding requires a far lower packet overhead to achieve the same success probability of decoding.

3.3 Number of Borrowed Symbols

In the previous subsection, we have reported the significantly lower packet overheads for successful decoding using LT-W decoding algorithm. This achievement is made with the introduction of an additional decoding method, namely, the Wiedemann algorithm. It is of interest here to investigate the computational complexity of LT-W decoding algorithm

that combines LT decoding and the Wiedemann algorithm.

In the LT-W decoding algorithm, the decoding always starts with the LT decoding procedure. Whenever the ripple becomes empty, the Wiedemann algorithm is activated to borrow a new symbol for LT decoding to continue. In [29], it is reported that the complexity of the Wiedemann's method is $O(k(\omega + n) \log k \log \log k)$, where k is the number of unrecovered symbols and ω is the number of nonzero entries in the coefficient matrix. Compared with the complexity of LT decoding which is $O(k \cdot \log(k/\delta))$ [10], the complexity of the Wiedemann algorithm is higher. In the worst case where all symbols are decoded by the Wiedemann algorithm, the computational complexity of the LT-W decoding algorithm follows that of the Wiedemann algorithm. However, in our observation, the Wiedemann's method is usually required only for a few times in the entire decoding process. Hence, the expected efficiency of the LT-W decoding operation is closer to that of the LT decoding process. To quantify this efficiency for a successful decoding process, in the following, we study the number of times that the Wiedemann's method is called in a single successful decoding process, which is also the number of symbols borrowed in a single successful decoding process.

We recall from Section 2.1.2 that a state (r, c, u) is a snapshot at a particular moment of the evolution of LT decoding, where r is the number of symbols in the ripple, c is the number of symbols in the cloud, and u is the number of unrecovered symbols. LT decoding terminates prematurely and is considered to have failed when it evolves into a state where $r = 0$ and $c \geq u > 0$. In the LT-W decoding algorithm, this event triggers the borrowing of a symbol using the Wiedemann's method, and the decoding process can proceed further. Hence, the LT-W decoding state evolution follows closely to that of the LT decoding process, except in states where $r = 0$ and $c \geq u > 0$, from which it continues to evolve after the borrowing of a symbol. In other words, in a state $(0, c, u)$

where $c \geq u > 0$, the LT-W decoding calls the Wiedemann's method to borrow a symbol, and it transits to state $(1, c, u - 1)$ with probability one.

We use a polynomial function in the form px^w as the state transition function, where p describes the probability of the state transition, $w = 0, 1$ indicates whether a symbol is borrowed in the transition, and x is a dummy variable. Note that in the system of LT decoding, w is always zero. The product of the state transition functions along a certain transition path, say, from state (r_1, c_1, u_1) to state (r_2, c_2, u_2) gives a polynomial term where the coefficient describes the probability that LT-W decoding evolves from state (r_1, c_1, u_1) to state (r_2, c_2, u_2) with the power of x indicating the number of borrowed symbols involved in the evolution. Summing the products of all the possible transition paths from state (r_1, c_1, u_1) to state (r_2, c_2, u_2) produces a generating function denoted as $\Upsilon((r_1, c_1, u_1); (r_2, c_2, u_2))$ that summarizes the probability of each transition path having a certain number of borrowed symbols involved. Let W be the random variable describing the number of times that the Wiedemann's method is called in a successful LT-W decoding procedure for k input symbols and n received packets. Then its probability generating function, $G_W(x)$, can be determined by

$$G_W(x) = \sum_{r_0, c_0, u_0} \sum_{c < c_0} \Upsilon((r_0, c_0, u_0); (r, c, 0)) \quad (\text{Eq. 3.2})$$

and we use numerical computation to obtain the expected number $E[W]$ of borrowed symbols from the above result.

As discussed earlier, the measure of the random variable W is only meaningful when LT-W decoding is successful. The condition that LT-W decoding is successful depends on the rank of the coefficient matrix. In other words, the generating function of W can be determined given that a full-rank coefficient matrix is reached. Since in (Eq. 2.3), the state transition probability $\Psi((r + 1 + s - t, c + t, u); (r, c, u - 1))$ includes cases of non full-rank coefficient matrix, we adjust this quantity to exclude those cases by applying

the full-rank coefficient matrix condition to (Eq. 2.3). Given in Section 2.2.4 that the quantity $P_f(k, n)$ is the probability that a full-rank coefficient matrix can be obtained for n encoded packets with respect to k original packets, the state transition probability under a full-rank coefficient matrix, $\Psi_{(LT-W)}((r + 1 + s - t, c + t, u); (r, c, u - 1))$ from state $(r + 1 + s - t, c + t, u)$ to state $(r, c, u - 1)$, is

$$\Psi_{(LT-W)}((r + 1 + s - t, c + t, u); (r, c, u - 1)) = \frac{\Psi((r + 1 + s - t, c + t, u); (r, c, u - 1))}{P_f(u, c + t)}. \quad (\text{Eq. 3.3})$$

In Fig. 3.4, we present the results of the expected number of borrowed symbols in a single successful LT-W decoding process for different numbers of input symbols, k . Two parameters used in Robust Soliton distribution are $\lambda = 0.2$ and $\delta = 0.1$. Besides, we conduct simulation tests to validate our approximated numerical results. As we can see from the figure, the result from our approximation is very close to the result from simulation when k is very small (i.e., $k = 10$ and 20). However, when $k = 30$, our approximation underestimates that of the simulation results. This discrepancy becomes more obvious when $k = 60$. This is because in our analysis, we treat the decoding process after using the Wiedemann's method as a new LT decoding process for analytical tractability. However, the actual degree distribution when the LT decoding process prematurely terminates no longer exhibits the robust Soliton distribution, and the recovery of one symbol using the Wiedemann's method retains the distorted distribution. As a result, the following LT decoding executions terminate prematurely more easily, resulting in a higher number of executions of the Wiedemann's method than that determined by our analysis.

Despite the discrepancy between the numerical and simulation results, both results presented in Fig. 3.4 indicate that the Wiedemann's method is not frequently called in a single LT-W decoding process. For the case of $k = 30$, to achieve 95% probability of

 3.4.a: $k = 10$

 3.4.b: $k = 20$

 3.4.c: $k = 30$

 3.4.d: $k = 60$

Figure 3.4: The expected number of borrowed symbols with respect to the number of received packets.

successful decoding, an overhead of 30% is required according to the results shown in Fig. 3.2, which translates into receiving 39 packets or $n = 39$. Our numerical results shown in Fig. 3.4 optimistically suggest that the expected number $E[W]$ of borrowed symbols is 0.75 while the simulation results suggest that it is 1.5. With a total of 30 symbols to recover, LT decoding accounts for 95% of the symbol recovery based on our simulation, and 97.5% based on our optimistic numerical estimation. Both the findings suggest that the efficiency of the LT-W decoding process is largely dominated by that of

LT decoding.

Likewise, for the case of $k = 60$, our results suggest that the highest expected number of borrowed symbols is around eight based on the simulation results, and around five based on the numerical results. Given a total of 60 symbols to recover, it is obvious that most symbols are decoded using LT decoding. Moreover, as n increases, the number of borrowed symbols reduces exponentially, and this leads to a far smaller number of borrowed symbols required. This again confirms that the use of LT-W decoding reduces the need for receiving excessive packets while maintaining computational efficiency.

3.4 LT-W V.S. Inactivation

As introduced in 2.3, inactivation decoding is a similar decoding scheme with LT-W decoding. The fundamental difference between these two decoding methods is how the decoder behaves when LT decoder cannot find another encoded symbol with degree 1. In LT-W, whenever the ripple is empty, we solve one symbol from the encoded symbols to reboot the LT decoder whilst the decoder only inactive one symbol and express the rest uncovered symbols with this inactive symbol. Note that it does not actually decode the symbols in inactivation decoding but keep the relationships between symbols and inactive symbols. It first identify all the inactive symbols. Then it solves these inactive symbols by Guassian elimination. At last, it uses those solved inactive symbols to decode the rest symbols through XOR operations.

Obviously, inactivation decoding trades off decoding complexity for low decoding overhead, which is the same as LT-W does. The most time cost part is solving the inactive symbols by Guassian elimination. The number of inactive symbols determines the cost. In this section, we compare the number of inactive symbols with the number borrowed symbols to show the performances of two methods.

In practice, the strategy of choosing inactive symbol has effect on the number of inactive symbols. In [39], the inactive symbol is randomly selected, which would lead to consecutive inactivation. In LT-W, a borrowed symbol is selected such that the most encoded symbols contains it. An inactive symbols can be chosen by this criteria also.

We conduct simulations to get the average number of inactive symbols under different parameter setups. The results shown in Fig. 3.5 are obtained by averaging 500 rounds of simulations. Two parameters used in Robust Soliton distribution are $\lambda = 0.2$ and $\delta = 0.1$. Inactive-1 represents choosing inactive symbol randomly and inactive-2 represents choosing inactive symbol by the above mentioned strategy. From Fig. 3.5,

Figure 3.5: Comparison between inactivation decoding and LT-W decoding.

inactivation decoding with strategically choosing inactive symbol performs similar with LT-W decoding, while inactivation decoding with randomly choosing inactive symbols performs the worst. This phenomena can be interpreted as follows. Choosing an inactive symbol randomly may cause a consecutively inactivation and the overall number of inactivation symbols may be larger. Choosing an inactive symbol with the same strategy as

LT-W can increase the probability of finding a new symbol with degree 1. Inactivation decoding solves those inactive symbol at the final stage, while LT-W decoding solves borrowed symbol during decoding. There is no fundamental difference between these two methods, thus the performance of these two methods are similar.

3.5 Running Time

In the previous subsections, we have demonstrated the superior performance of our LT-W decoder in terms of overhead packets and the number of borrowed symbols. In this subsection, we shall present the actual running time of the LT-W decoder.

We have implemented both LT and LT-W decoding procedures using MATLAB R2009a on a PC equipped with a 2.66 GHz Intel 2-core CPU. The symbol size is set to 512 bytes. The success probability of decoding is set to 95%. The result is shown in Fig 3.6. As can be seen from this figure, LT-W outperforms LT, especially when the number of input symbols is small. The gap between the two results closes as the number of symbols increases. This is because, when the number of input symbols is small, LT-W operates with a lower packet overhead, and the reduction in runtime as a result exceeds the runtime for the recovery of borrowed symbols. Hence, a shorter runtime is reported for LT-W when the number of symbols is small.

3.6 Conclusion

In this chapter, we have first discussed the drawbacks of the LT codes in handling small numbers of input symbols and the main idea of LT-W decoding. To confirm the efficiency of LT-W decoding, we have analyzed the packet overhead and computational efficiency in terms of the expected number of borrowed symbols. Our analysis confirms that the computational efficiency of LT-W decoding is largely dominated by that of LT decoding.

Figure 3.6: The actual running time with respect to the number of input symbols.

We have further compared the runtime between LT-W and LT, with results showing a shorter runtime for LT-W when the number of input symbols is small. Our results confirm that LT-W decoding algorithm not only exhibits lower packet overheads, but also maintains high computational efficiency.

For the comparison with other solutions, we choose inactivation decoding. The result shows that LT-W decoding and inactivation decoding are similar. The only difference is LT-W solves a borrowed symbol to reboot the LT decoding, while inactivation decoding just inactivates a symbol and solves all the inactive symbols at last.

Chapter 4

Joint Unequal Loss Protection and LT Coding for Layer-Coded Media Delivery

In this chapter, we propose a joint Unequal Loss Protection (ULP) and LT coding (ULP-LT) scheme for layered media delivery, where different numbers of Forward Error Correction (FEC) are allocated to different layers to guarantee that a lower layer can always be decoded before decoding its higher layers and LT codes are used to deal with the varying channel conditions. The ULP FEC allocation is independent of the channel condition since its main objective is to facilitate the intermediate outputs for LT codes. To avoid introducing large overhead, the total amount of FEC is limited to a small percentage. On the other hand, LT codes are directly used, which avoids the change of the degree distribution. We analyze the feasibility of joint FEC and LT decoding and propose a simple method for FEC allocation. Simulation results show that with a small amount of overhead our ULP-LT scheme can produce good intermediate performance while still enjoying the nice features provided by LT codes.

4.1 Review of ULP

The concept of unequal loss protection has been widely exploited for prioritized media delivery. The basic idea is to add different amount of FEC to different media layers. Fig. 4.1 shows a general layered ULP structure, where FEC is added to each row and each column is a packet. Suppose Reed-Solomon (RS) code is used for FEC, a decoder can decode all the data from layer L_1 to layer L_i if the received number of packets exceeds the number of data columns for layer L_i . Most of ULP methods such as [89] study how to find the optimal FEC allocation given a total bandwidth and a certain channel condition. As aforementioned, the FEC allocation is only optimal for the designed channel condition. When the channel condition is changed, the FEC allocation becomes unoptimal. On the contrary, in our joint ULP and LT design, the FEC allocation is independent of the channel and varying channel conditions can be well handled by LT codes.

Figure 4.1: A layered unequal loss protection structure.

4.2 Proposed Joint ULP and LT Coding System

Fig. 4.2 shows the proposed joint ULP and LT coding system in a multicast scenario. In particular, the sender first allocates different FEC codes to different layers under the constraint of a given total FEC redundancy. Then, the original data bits of each layer together with the FEC bits are placed according to Fig. 4.1. Packetization is done vertically such that each packet contains a proportion of data bits of each layer. After packetization, the sender encodes these packets using LT codes and injects the encoded packets into the network. The end users collect these encoded packets and perform decoding process when enough encoded packets are received.

Figure 4.2: The system diagram of proposed UEPLT scheme.

4.2.1 Encoding and decoding

For simplicity, we will illustrate the decoding process of data without priority. To decode data with priority, the receiver just needs to perform the decoding process on each layer.

Suppose there are k original packets $X = [x_1, x_2, \dots, x_k]^T$ to be transmitted at the sender. The sender first generates $n - k$ FEC packets for these k packets. We use random linear network coding to generate the FEC packets. Every FEC packet is a linear combination of the original packets. The generating process can be expressed in a matrix form:

$$Y_{n \times 1} = C_{n \times k} X_{k \times 1} \quad (\text{Eq. 4.1})$$

where

$$C = \begin{pmatrix} & I_k & \\ c_{1,1} & c_{1,2} & \dots & c_{1,k} \\ c_{2,1} & c_{2,2} & \dots & c_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-k,1} & c_{n-k,2} & \dots & c_{n-k,k} \end{pmatrix}, c_{i,j} \in \mathbf{GF}(2^q).$$

Note that all the operations are under $\mathbf{GF}(2^q)$. For a finite field with characteristic 2, addition is accomplished using the XOR operator, which is compatible with the subsequent LT coding. After generating the FEC packets, the sender performs LT encoding process which can also be expressed in the matrix form.

$$Z = V \cdot Y, \quad (\text{Eq. 4.2})$$

where each row in V is a coding vector in LT codes and each row in Z is an encoded packet. Each element in V is either 0 or 1. Since the LT codes are rateless, the number of rows in V and Z can be infinite.

To the end users, they only collect a finite number of encoded packets to decode the original data packets. Suppose a user receives m encoded packets $\hat{Z}$, which is generated

by

$$\hat{Z}_{m \times 1} = \hat{V}_{m \times n} Y_{n \times k} = \hat{V}_{m \times n} C_{n \times k} X_{k \times 1} \quad (\text{Eq. 4.3})$$

Each coding vector in $\hat{V}$ can be obtained from the header of the corresponding received packet. The random elements in C are generated according to a certain seed value. One example is that the sender can use the unique media content ID as the seed for generating random numbers. As long as a receiver has the same media content ID, it can generate the same set of the random coefficients.

Let $G = \hat{V}C$, named as coding matrix. If a receiver wants to recover all the original data packets, the rank of G should be equal to k . When the receiver gets enough rank, it can perform Gaussian elimination to decode the original data packets. Considering that the complexity of the Gaussian elimination method, i.e. $O(k^3)$, is much higher than that of the LT decoding, i.e. $O(k \ln k)$, the decoding strategy proposed in chapter 3 can be adopted here to reduce the decoding complexity. The idea is to make use of LT decoding as much as possible. Only when LT decoding is unable to decode, the Gaussian elimination method is then used to decode a few source packets out so as to reactivate the LT decoding.

Fig. 4.3 shows the simulation results of the probability of a receiver reaching the full rank state under different numbers of original packets and different proportions of FEC packets. From the figure, we can see that at most of the different settings combining FEC with LT codes only needs a slight overhead, about 2 to 3 packets more than the number of original data packets, for reaching the full rank with extremely high probability. Only when the number of original packets is small (10 or 20) and the FEC proportion is also small (10%), it requires more overhead to reach the 100% decodable state. This is mainly because of the large overhead introduced by LT codes at small number of source packets.

4.3.a:

4.3.b:

4.3.c:

4.3.d:

Figure 4.3: The probability of a receiver reaching the full rank state under different numbers of original packets and different proportions of FEC packets. (a) 10 original data packets. (b) 20 original data packets. (c) 30 original data packets. (d) 40 original data packets

4.2.2 FEC Allocation

In our scheme, the placement of the original data bits and the FEC bits follows the same style as that in [89] illustrated in Fig. 4.1. However, different from [89], the FEC allocation here should be no longer dependent on the channel loss probability. This is because rateless codes are not customized for one packet loss rate but for any packet loss rate.

Our objective to bring ULP into LT codes is to gradually decode more and more source data layers according to the importance of the layers with the increasing number of received packets. In practice, it is highly desired to decode a certain layer when the number of received packets is accumulated to a certain percentage. Let α_i denote the targeted percentage of the received packets at which the receiver is expected to decode the i -th layer. Assume that all the α_i values are pre-set either by the user or according to the applications. Our FEC allocation problem can be summarized as: given the targeted percentage α_i for decoding each individual layer, the total FEC redundancy rate β , and the number of source packets k_i for each individual source layer, how to find the FEC allocation f_i for each layer? Note that all the percentages are with respect to the total number of source packets N_s and f_i is the number of FEC columns allocated to the i -th layer in the $N \cdot L$ block shown in Fig. 4.1, where N and L are the total number of packets (original data plus FEC) and the packet length, respectively.

It is clear that allocating too much FEC to a layer results in an unnecessary early decoding of the layer and wastes the resource, while allocating insufficient FEC to a layer leads to a delay in decoding the layer and is unable to well match the requirements. To trade off between resource utilization and meeting the expectation, we formulate our

FEC allocation problem as

$$\begin{aligned} \min_{f_i} d &= \sum_{i=1}^M |\alpha_i \cdot N_s - R_i(f_i)| \\ \text{s.t.}, \quad &\begin{cases} \sum_{i=1}^M \frac{k_i L}{N - f_i} \cdot f_i \leq \beta \cdot N_s L \\ f_1 \geq f_2 \geq \dots \geq f_M \end{cases} \end{aligned} \quad (\text{Eq. 4.4})$$

where R_i is the actual number of received packets needed to ensure the decoding of the i -th layer with a FEC allocation of f_i , $(1 + \beta) \cdot N_s = N$, and $\sum_{i=1}^M k_i = N_s$. The basic idea of (Eq. 4.4) is to match the actual layer decoding time to the expected one as close as possible. The first constraint in (Eq. 4.4) is to limit the overall FEC redundancy and the second constraint implies unequal loss protection, i.e. more protection for data in lower layers.

Note that R_i can be figured out from the curves illustrated in Fig. 4.3, which are pre-generated by simulations. Specifically, given the number of source packets $N - f_i$ and the FEC overhead f_i for the i -th layer, we are able to find the minimum number of received packets needed to reach the full rank state with extremely high probability. As discussed in chapter 3, when the number of source packets $N - f_i$ is not small, the overhead for reaching the full rank state is small. Thus, we can consider that $R_i \approx N - f_i$ for relatively large $N - f_i$ values.

Considering the characteristics of layer-coded media, whose rate-distortion function is typically exponential [90], it is common to assume that the data size of a lower layer is much smaller than that of a higher layer, i.e. $k_1 \ll k_2 \ll \dots \ll k_M$. Furthermore, considering a practical requirement of smooth layer decoding, we set $\alpha_i \gg \frac{k_i}{N_s}, \forall i$. All these priors facilitate a small overall FEC redundancy of β to play an important role in generating intermediate decoding results.

Here, we propose a simple low-complexity iterative algorithm to find the FEC allocations, which can be summarized as follows:

- Step 1: Initially, allocate the same amount of FEC to all the layers, i.e. $f_1 = f_2 = \dots = f_M$.
- Step 2: Starting from the highest layer, the redundancy assigned to a higher layer is moved to a lower layer, from the closest layer to the lowest layer, as long as the new allocation $\vec{f}$, $\vec{f} = \{f_1, f_2, \dots, f_M\}$, satisfies the second constraint in (Eq. 4.4) and the distance d in (Eq. 4.4) is reduced.
- Step 3: Repeat Step 2 until no further improvement can be achieved.

4.3 Simulation Results

In this section, we validate the effectiveness of the proposed joint ULP and LT coding scheme. All the simulation parameters are listed in Table. 4.1.

Table 4.1: Symbol Definitions

N_s	the total number of source packets ($N_s = 150$)
L	length of packets ($L = 2000$ bits)
N	the total number of source+UEP packets
M	the number of layers for UEP ($M = 5$)
β	the percentage of UEP redundancy $\beta = (\frac{N}{N_s} - 1) \times 100\%$
α_i	the required percentage to decode the i -th layer $\alpha_i = i \cdot 20\%$, $i \in [1, M - 1]$, $\alpha_M = 1 + \beta$.
n_i	the number of packets for the i -th layer 5, 10, 20, 40, 75

We create a pseudo media content of five layers with 5, 10, 20, 40, 75 packets for layer 1, 2, 3, 4, 5, respectively. The length of a packet is fixed to 2000 bits. All the numbers here can be scaled up to large values and the simulation results will have similar trends. The expected decoding points for layer 1 to 5 are chosen to be uniformly spaced as 40%,

60%, 80%, 100% and $(1 + \beta)$, respectively. All the simulation results described below are obtained through 300 independent trials.

Fig. 4.4 shows the decoding probabilities of different layers. It can be seen that the proposed ULP-LT scheme can provide good intermediate decoding outputs. The results also show that a lower layer can always be decoded before the decoding of a higher layer. We observe that a few higher layers have the same decoding probability which is due to insufficient amount of FEC budget. Compared the results with 5% and 10% FEC overhead, we can see that the more FEC redundancy we have, the better intermediate results we can get. On the other hand, the overall FEC redundancy should be small in order not to affect the major role played by LT codes. We feel 10% FEC is a reasonable value for practical applications. Note that all the percentages are with respect to the total number of source packets $N_s = 150$

Figure 4.4: The decoding probabilities of different layers. Left: with 5% FEC redundancy. Right: with 10% FEC redundancy.

Fig. 4.5 shows the average number of decoded layers using different methods. It can be seen that both LT and LT-W cannot provide intermediate outputs. Surprisingly, in terms of the final output, i.e. decoding all the layers, ULP-LT with 5% FEC overhead

requires about the same number of packets as the FLT method and ULP-LT with 10% FEC overhead needs smaller number of received packets than the original LT coding. This is mainly because LT coding is inefficient when coding small number of packets. We are thinking to compare with the methods in [46] and [47]. However, both of the methods are difficult to be re-implemented due to their complexity and no publicly available details. On the other hand, both methods can only provide intermediate outputs in the probability sense while our ULP-LT method guarantees a lower layer can always be decoded before the decoding of a higher layer.

Figure 4.5: The average number of decoded layers using different methods.

Next, we consider a simple multicast scenario, where a sender is sending the same pseudo media content to three heterogeneous receivers with Packet Loss Probability (PLR) of 0.05, 0.1, and 0.2, respectively. We use the packet loss model with the exponential Probability Mass Function (PMF), the same as that in [91]. Fig. 4.6 shows the

results of the average number of the decoded layers for different users. We can see that different receivers can decode different number of layers, which correspond to different media qualities, without any adaption at the sender.

Figure 4.6: The average number of decoded layers for heterogeneous users. Left: with 5% FEC redundancy. Right: with 10% FEC redundancy.

4.4 Conclusion

In this chapter, we presented a novel joint ULP and LT coding for layered media delivery. The idea is to give a small amount of overhead for ULP to guarantee the priorities among the layers while using LT coding to deal with varying channel conditions. We have discussed the feasibility of joint ULP and LT decoding and how to perform FEC allocation. Our simulation results have demonstrated that the proposed scheme can achieve good intermediate performance with controllable overhead.

Chapter 5

Delay-Optimized File Retrieval under LT-Based Cloud Storage

Different from those existing works reviewed in chapter 2 which treat the retrieval performance problem from the OS level, we deal with this problem from network level in this chapter. In [92, 93, 94, 95], the authors discussed some mechanisms for reducing data fetching latency in distributed networks. Those mechanisms are designed for general content without considering any coding metrics. In this chapter, we focus on designing optimized retrieval method for a specific coding technique, LT code. Those mechanisms discussed in the literature can be also applied as an extension of our method. As stated in [96], traffic hot spots emerge at the core of cloud storage system which is responsible for communications between the system and end users. We call this core as a *portal*. In this chapter, we want to address the problem how to minimize the retrieval delay generated by the portal in an LT-based cloud storage system. We study the tradeoff between the successful LT decoding probability and storage retrieval delay and introduce a multiple stage storage retrieval scheme where we show that storage retrieval delay can be reduced without sacrificing the performance on decoding probability. We further demonstrate the optimal retrieval setup for the case of a two-stage retrieval.

Figure 5.1: System Architecture for a cloud storage system.

5.1 System Model and Problem statement

In this section, we first present a schematic description of a cloud storage architecture, in which LT-encoded data packets are spread out across a pool of storage nodes for high availability. Following that, we formulate an delay-optimal file-retrieval problem, which aims to minimize the retrieval delay by strategically scheduling packet retrieval requests.

5.1.1 System Architecture

In this chapter, we focus on a distributed cloud storage system (e.g., a community-based cloud storage system as in Tahoe-LAFS [97]). As illustrated in Fig. 5.1, the cloud storage system manages a set of storage nodes through a communication network. The

users can access to the storage service through a dedicated portal. One particular storage application of interest in this research is the file storage.

We assume that the storage service provides both premium LT-based file storage and regular file storage. For the LT-based file storage, the portal, upon receiving a file from a user request, encodes the file with a chosen LT code into a set of LT fragments or packets¹ and spreads them across different nodes for reliability and high availability. Upon receiving a file-retrieval request, the portal sends a few packet retrieval requests to different storage nodes, and gathers sufficient encoded packets to be forwarded to the user. For the regular file storage, the portal simply stores the received user file into a chosen storage node, with some potential replication in another set of nodes.

We assume that the number of subscribers for the premium LT-based storage service is much smaller than that for the regular storage service. As such, it is with high probability that, at one instant, only one user will retrieve his/her LT-coded files and the majority of the file-retrieval requests are for the regular storage service. In this chapter, we will analyze such a simplified use case. The insights obtained through mathematical framework and numerical analysis will shed lights on more practical application scenarios.

5.1.2 Traffic Model

In this architecture, the portal node observes two types of packets. The first one is packets for the regular storage service, considered as a process of ambient traffic. The ambient traffic follows a particular arrival process with an arrival rate of λ_a . Moreover, the length of ambient packets is modeled as a random variable with mean l and variance σ^2 . The second one is the LT-coded packets for the premium storage service. We model the arrival delay of each LT-coded packet is identically distributed, each of which is an

¹Here we use the term *packet* interchangeably with *fragment* as we assume that an IP packet carries an LT encoded fragment in the system.

Figure 5.2: Arrival process of LT encoded packets seen at the portal for a file-retrieval request.

exponential random variable with rate θ . The length of LT-coded packets is assume to be a constant, denoted by l_{LT} .

When the portal requests n LT-coded packets from the storage grid, the packet arrival process is illustrated in Figure 5.2. We assume that the packet request is sent out at time 0 (i.e., $t_0 = 0$), and denote the arrival time of the i -th LT-coded packet as t_i . We also denote the inter-arrival time between the $i - 1$ -th packet and the i -th packet as $\tau_i = t_i - t_{i-1}$. Since the arrival delay of each LT-coded packets are iid with exponential distribution (rate of θ), τ_i is an exponential random variable with rate of $(n - i + 1)\theta$ [98].

5.1.3 Problem Statement

In the considered distributed storage system, the bottleneck lies on the portal, because a large number of users can issue their file requests to the portal. In this chapter, we focus on the file-retrieval delay, defined as the duration between the time for the portal receiving an LT-coded file request and the time when the last LT-coded packet is sent out by the portal. The file-retrieval delay is a good indicator of user experience. Therefore, we aim to reduce the file-retrieval delay by strategically scheduling the LT-coded packet requests.

The delay-optimal file-retrieval problem is stated as follows. We assume a probabilistic file-retrieval model. In order to achieve the probability of successful decoding of p , the portal needs to request n LT encoded packets from the storage grid. Traditionally, the n packets are requested in one shot. This scheme is referred to as a *one-stage request scheme*. In this chapter, we propose to use a *multiple-stage request scheme* to improve the file-retrieval performance. Specifically, the user can divide the request into s stages, each consisting of requests for $n_1, n_2, \dots, n_s$ packets ($\sum_i n_i = n$). If the user successfully decodes the file after stage m , it will stop requesting the rest $\sum_{i=m+1}^s n_i$ packets.

In our proposed multi-stage request scheme, the designing objective is to minimize the average file-retrieval delay, for a given number of stages. Mathematically, it can be formulated as the following optimization problem,

$$\min \quad \overline{D}_s(n_1, n_2, \dots, n_s) \quad (\text{Eq. 5.1})$$

$$\text{s.t.} \quad \sum_{i=1}^s n_i = n, \quad (\text{Eq. 5.2})$$

$$n_i \geq 0, 1 \leq i \leq s. \quad (\text{Eq. 5.3})$$

Here $\overline{D}$ denote the average retrieval delay under a given request scheme.

The result we will obtain in this research is surprising. Intuitively, multiple stages of packet retrieval incurs additional delay in storage retrieval. However, we will show that the multi-stage retrieval scheme can outperform a single-stage retrieval scheme in terms of the average file-retrieval delay. This delay reduction originates from the following observation. In the multi-stage retrieval scheme, retrieving a just adequate number of packets may already give sufficiently high success decodability, eliminating the need for further rounds of packet retrieval. Only when the file cannot be decoded with the set of received packets, will additional packet requests be sent in subsequent stages. In such a case, the single-stage retrieval scheme is a special case of our proposed multi-stage

Figure 5.3: Fitting result versus simulation result.

retrieval scheme, when $s = 1$. From an optimization perspective, the performance of the multi-stage retrieval scheme cannot be worse than that of the single-stage scheme.

5.1.4 LT Decodability

Existing research on characterizing the LT decodability has not resulted in a closed-form solution yet. In [28], the authors proposed a dynamic programming algorithm to calculate the LT decodability, with a computing complexity of $O(n^3 \log^2 n)$. Another algorithm in [99] reduces the computing complexity to $O(k^2 \log k)$. However, none of these previous research efforts resulted in a closed-form solution to the LT decodability, while relying on numerical evaluation using dynamic programming.

In this chapter, we derive an empirical model for the LT decodability, by leveraging

Table 5.1: Fitting result for α and β

k	100	200	300	400	1000
α	6.108	10.268	14.907	17.912	35.65
β	1.263	1.3315	1.3373	1.3235	1.286
SSE	0.003096	0.005643	0.01805	0.01113	0.02308

the numerical evaluation techniques from [99]. Notice that, in an LT-based cloud storage system, the decodability under 50% is of less engineering interest. As a result, we focus on the decodability over 50% in the rest of this chapter. We use a two-step procedure to derive the empirical model for the LT decodability, as follows:

- Firstly, we plot the LT decodability as a function of the number of encoded packet, by using the numerical technique suggested in [99].
- Secondly, we use a generic function of $f_k(n) = 1 - e^{-\alpha(\frac{n}{k} - \beta)}$ to curve fit the numerical results, where a and c are two fitting parameters related to k .

We illustrate this approach in Figure 5.3. First, we have obtained the simulated LT decodability for the cases of $k = 100$, $k = 200$, $k = 300$, $k = 400$ and $k = 1000$ as illustrated with the solid curves. Second, using MATLAB's "Curve Fitting Tool" with a specific fitting function of $f_k(n) = 1 - e^{-\alpha(\frac{n}{k} - \beta)}$, we obtain the fitting results of α and β together with the sum of squares due to error (SSE) in Table. 5.1. Finally, we compare the LT decodability between the simulation results and the fitting results (the dotted curves) for $k = 100, 200, 300, 400, 1000$, in Fig. 5.3. It can be observed that the empirical model captures the simulation results with a decent accuracy.

In the rest of this chapter, we will use this empirical model of the LT decodability as a basis for our optimization framework.

Figure 5.4: Components of delay of one stage request

5.2 File-Retrieval Delay Analysis

In this section, we characterize the average file-retrieval delay for different request schemes in the presence of ambient traffic. To validate our analytical results, we use computer simulation written in Java using NetBeans 7.1. All the simulation results are obtained by averaging 1000 simulation instances. In simulations, we set that the length of packets in the ambient traffic follows an exponential distribution. Although it is a simplified model, the insights obtained with this simple model can be applied to guide practical system design. Table 5.2 lists the default values of some parameters in the simulator.

Table 5.2: Default values in Simulation

Notation	Physical Meaning	Value
l	Length of packet for ambient traffic	1024 Bytes
l_{LT}	Length of LT-coded packet	1024 Bytes
r	Transmission rate	800 kbps

5.2.1 Delay Analysis for One Stage Request Scheme

We first investigate the file-retrieval delay $D_1(n)$ for n LT packets in the one-stage request case. This delay consists of two parts as shown in Fig. 5.2. The first part arises from the traffic arriving before the LT request and the second part arises from the traffic arriving after the LT request. In this analysis, we assume Poisson arrival process for

ambient traffic and LT-coded packets. The first part only contains ambient traffic and it is treated as a classical M/G/1 queue² with delay denoted by W . The second part can be further divided into two sub-parts. Firstly, it contains a constant which is the transmission time of n LT packets, which can be derived as $\frac{nl_{LT}}{r}$. The rest is the time to process the ambient traffics which arrive at the portal during the inter-arrival time of each LT packet. This part is denoted by T . Thus, the delay $D_1(n)$ can be expressed as

$$D_1(n) = W + \frac{nl_{LT}}{r} + T. \quad (\text{Eq. 5.4})$$

Taking an expectation on both sides, we obtain

$$\begin{aligned} \overline{D}_1(n) &= \overline{W} + \overline{T} + \frac{nl_{LT}}{r} \\ &= \frac{\lambda_a(\sigma^2 + l^2)}{2r^2(1 - \frac{\lambda_a l}{r})} + \frac{\lambda_a l}{r} \sum_{i=1}^n \tau_i + \frac{nl_{LT}}{r} \\ &= \Gamma + \frac{\lambda_a l}{r} \sum_{i=1}^n \frac{1}{i\theta} + \frac{nl_{LT}}{r} \\ &\simeq \Gamma + \frac{\lambda_a l}{r\theta} (\ln(n) + 1) + \frac{nl_{LT}}{r}, \end{aligned} \quad (\text{Eq. 5.5})$$

where Γ is a constant when the parameters for the traffic are determined.

There are three terms in (Eq. 5.5), Γ , the logarithm term and the linear term. In order to ensure the portal having a stable state, λ_a should be less than 100 packets per second under our setting stated in Table. 5.2. Under this condition, the order of magnitude of Γ is 10^{-3} . The number of source symbols k usually exceeds 100 which implies the number of requested packets n is also larger than 100. Thus, comparing to the linear term which has the order of magnitude equal to 10 or 10^2 , Γ is negligible. If we assume that λ_a and θ are comparable (*e.g.* $\frac{\lambda_a}{\theta} \approx 1$), the logarithm term has the order of magnitude equal to 10^{-2} which is also negligible. As a result, the linear term $\tilde{D}_1(n) = \frac{nl_{LT}}{r}$ can be used to approximate $\overline{D}_1(n)$. In Fig. 5.5, we plot the values of $\overline{D}_1(n)$ and $\tilde{D}_1(n)$ for n varying

²More information on M/G/1 queue can be found in [100].

Figure 5.5: Comparison between $\bar{D}_1(n)$ and $\tilde{D}_1(n)$.

from 100 to 800 when $\lambda_a = 30$ and $\theta = 50$. We can observe that Γ and logarithm term contribute little to the retrieval delay. So it is reasonable we use $\tilde{D}_1(n)$ to approximate $\bar{D}_1(n)$. Similar results are found with different setting of λ_a and θ . For brevity, those results are not reported.

In Fig. 5.6, we plot the numerical retrieval delay $\tilde{D}_1(n)$, compared with the simulation result, as a function of the number of packets request under different λ_a and θ settings. Notice that the numerical results match the simulation result well, confirming that using $\tilde{D}_1(n)$ to approximate (Eq. 5.5) is applicable.

5.2.2 Delay Analysis for Multiple Stage Request

In this subsection, we first investigate the file-retrieval delay for the two-stage request scheme. The packet arrival process is illustrated in Fig. 5.2.2.

Figure 5.6: Delay in one stage request. 1: $\lambda_a = 10$, $\theta = 25$; 2: $\lambda_a = 30$, $\theta = 50$; 3: $\lambda_a = 60$, $\theta = 50$.

Suppose in the first stage, the user requests n_1 LT encoded packets. The delay for the first stage is $D_1(n_1)$. After decoding these n_1 packets, if the user fails to decode the original file, it continues to request the n_2 packets. Here we assume that during the decoding process of n_1 packets, the transmission queue in the portal has already returned to the steady state. This assumption is reasonable since it does take some time for the

Figure 5.7: Components of delay of two stage request

user to determine if the n_1 LT-encoded packets are decodable. Thus, the delay for the second stage is identical to the first stage except for the number of encoded packets the user requests. As a result, the overall file-retrieval delay for the two stage request case is given by

$$\begin{aligned}
 D_2(n_1, n_2) &= f_k(n_1)D_1(n_1) \\
 &+ (1 - f_k(n_1))(D_1(n_1) + D_1(n_2)) \\
 &= D_1(n_1) + (1 - f_k(n_1))D_1(n_2),
 \end{aligned} \tag{Eq. 5.6}$$

where $f_k(n)$ is the probability of successful decoding with n LT encoded packets. Using the fitting result presented in section 5.1.4, we can calculate $f_k(n)$ approximately. Taking expectation on both sides, we have

$$\overline{D}_2(n_1, n_2) = \overline{D}_1(n_1) + (1 - f_k(n_1))\overline{D}_1(n_2). \tag{Eq. 5.7}$$

We plot in Fig. 5.8 and Fig. 5.9 the average file-retrieval delay for the two-stage request scheme, as a function of the number of packets requested in the first stage, under different setting of k , n and traffic loads. Notice that the numerical results match the simulation results well, regardless of the traffic load. Moreover, in all results, the file-retrieval delay first decreases and then increases as the number of packets requested in the first stage increases. This consistent trend indicates a potential for file-retrieval delay minimization by choosing an appropriate number of packets requested in the first stage. We shall address this optimization in section 5.3.

The result for the two-stage request scheme can be generalized into arbitrary t -stage request. Specifically, the delay expression for t -stage request can be defined recursively,

$$\begin{aligned}
 D_t(n_1, n_2, \dots, n_t) &= (1 - f_k(n_1))(D_1(n_1) \\
 &+ D_{t-1}(n_2, \dots, n_t)) \\
 &+ f_k(n_1)D_1(n_1),
 \end{aligned}$$

 Figure 5.8: Delay in two stage request when $k = 100$ and $n = 200$.

where the initial values for D_1 and D_2 are defined in (Eq. 5.4) and (Eq. 5.6).

In Fig. 5.10 and Fig. 5.11, we plot the optimal delay comparison obtained by different request schemes for different number of original symbols under different traffic load. Both figures show that multiple stage request scheme outperform traditional one-stage request. More specifically, two-stage request scheme can reduce retrieval delay by 15% comparing with one stage request. Whilst, three-stage request scheme further reduces the delay 4% only. It is obvious that to find an optimal request scheme in three-stage request is much more difficult than that in two-stage request and the improvement of three-stage request is not so significant. Consequently, we will only focus on finding an optimal two-stage

Figure 5.9: Delay in two stage request when $k = 400$ and $n = 700$.

request scheme in the rest of this chapter.

5.3 Optimal Scheduling for Packet Retrieval

In this section, we will investigate the optimal scheduling of packet requests to minimize the file-retrieval delay for two-stage request scheme.

The results in Fig. 5.8 and Fig. 5.9 suggest an optimal two stage request scheme from (Eq. 5.6). Another observation is that for a rational target successful probability (larger than 90%), the optimal number of LT-coded packets requested in the first stage for a two-stage request scheme yields $f_k(n_1) > 50\%$. Thus, we can apply the fitting function

Figure 5.10: Minimal delay comparison among one-stage request, two-stage request, three-stage request when $k = 100$.

for $f_k(n)$ to (Eq. 5.6) and simplify the optimization problem presented in section. 5.1.3 for two-stage request. For the convenience of discussion, we formalize the simplified optimization problem here.

$$\begin{aligned}
 \min \quad & \overline{D}_2(n_1) = \overline{D}_1(n_1) + e^{-\alpha(\frac{n_1}{k} - \beta)} \overline{D}_1(n - n_1) \\
 \text{s.t.} \quad & k\beta \leq n_1 \leq n, n_1 \in \mathbb{N}
 \end{aligned} \tag{Eq. 5.8}$$

By solving the above optimization problem, we can get the optimal number of packets requested in the first stage for a two-stage request scheme. Instead of finding the exact

Figure 5.11: Minimal delay comparison among one-stage request, two-stage request, three-stage request when $k = 400$.

solution, we relax the integer constraint, and assume n_1 is a positive real number to solve the approximate problem without integer constraint. Note that the minimum $\overline{D}_2(n_1)$ without the integer constraint is a lower bound of the minimum $\overline{D}_2(n_1)$ with integer constraint.

We observed in section 5.2.1 that $\overline{D}_1(n)$ can be approximated as $\frac{l_{LT}}{r}n$. Thus, $\overline{D}_2(n_1)$ in (Eq. 5.8) is convex.

Now, take differentiation with respect to n_1 on both sides of (Eq. 5.7) and set $\frac{\partial \overline{D}_2}{\partial n_1} = 0$,

we get the equation $e^{\frac{\alpha}{k}(n_1 - k\beta)} = \frac{\alpha}{k}(n - n_1) + 1$. The solution to this equation is

$$n_1^* = n + \frac{k}{\alpha} - \frac{k}{\alpha} W(e^{\alpha(\frac{n}{k} - \beta) + 1}). \quad (\text{Eq. 5.9})$$

Here, $W(x)$ is Lambert function [101] which is defined as $x = W(x)e^{W(x)}$. Note that $W(x)$ cannot be expressed in terms of elementary functions, thus calculating the real value of $W(x)$ is not an easy task. There exists literatures on designing algorithm to compute $W(x)$ [102, 103]. However, we are more interested in analytical bound of $W(x)$.

In [104], the authors proved the following bound of $W(x)$,

Lemma 5.1 *For $x > 1$ we have*

$$W(x) \geq \frac{\log x}{1 + \log x} (\log x - \log \log x + 1), \quad (\text{Eq. 5.10})$$

with equality only for $x = e$.

Fig. 5.12 shows the comparison between the real value of $W(x)$ and the lower bound in (Eq. 5.10). It is clear that this lower bound is tight enough and we can use this bound to approximate $W(x)$.

By applying (Eq. 5.10) on (Eq. 5.9), we have

$$\begin{aligned} n_1^* &= n + \frac{k}{\alpha} - \frac{k}{\alpha} W(e^{\alpha(\frac{n}{k} - \beta) + 1}) \\ &\approx k\beta + \frac{k}{\alpha} \frac{(\alpha(\frac{n}{k} - \beta) + 1) \cdot \log(\alpha(\frac{n}{k} - \beta) + 1)}{\alpha(\frac{n}{k} - \beta) + 2}. \end{aligned} \quad (\text{Eq. 5.11})$$

It is trivial that $n_1^* > k\beta$, and

$$\begin{aligned} n_1^* &\leq k\beta + \frac{\alpha}{k} \log(\alpha(\frac{n}{k} - \beta) + 1) \\ &\leq k\beta + \frac{k\alpha(\frac{n}{k} - \beta)}{\alpha} \\ &= n. \end{aligned} \quad (\text{Eq. 5.12})$$

Figure 5.12: Comparison between $W(x)$ and its bound.

In (Eq. 5.12), it shows $n_1^* \leq n$. Thus, we confirm that n_1^* is the optimal request in first stage which minimizes the two-stage file-retrieval delay.

In Fig. 5.13, we show the comparison of optimal first stage request between simulation and approximation under different k . In the simulation, the traffic parameters λ_a and θ are set to 30 packets per second and 50 packets per second respectively. The results confirm that the approximation of optimal first stage request obtained from (Eq. 5.11) is practical. In (Eq. 5.11), the optimal first-stage request is unrelated to traffic load (*i.e.* λ_a, θ). When the number of original symbols k is fixed (*i.e.* the parameters a and c are known.), the optimal first-stage request is determined. This is consistent with the results shown in Fig. 5.8- 5.9. This phenomena can be explained by following reason. Compared with the LT encoded packets request in each stage, the packets belonging to ambient traffic are negligible. The file-retrieval delay is mainly caused by the processing the requested LT encoded packets.

Figure 5.13: Comparison of optimal first stage request between simulation and approximation under different k .

By employing Eq. 5.11, if we increase the target successful probability (*i.e.* increase the number of requested packets n), the optimal number of packets requested in first stage n_1^* will be also increased. The ratio $\frac{n_1^*}{n}$ is decreased when n is increased. We plot the ratio in Fig. 5.14. This ratio dropping phenomena can be understood as follows. From the aspect of decodability probability, every packet contributes in the decoding process. From Fig. 5.3, we see an unequal contribution of each packet to the decodability probability. The contribution of an additional packet to the decodability probability is high when the probability stays at a lower level. As the probability progresses higher, the contribution of each additional packet drops. In other words, to make a same gain in

the decodability probability, more packets are needed when the probability is at a higher level. Notice from Fig. 5.5 that delay in one stage request scheme increases linearly with the number of requested packets. Consequently, tradeoff exists between decodability and delay. Instead of requesting all the packets required by the target decodability probability at once, requesting a small portion of the number of packets needed in the first stage will generate a decoding probability that contributes most to the final decoding probability; while the marginal contribution from the second stage is relatively small. As the decodability increased, the contribution from the second stage becomes smaller. In other word, a proper number of packets requested in the first stage will give us a relatively high decoding probability. As a result, setting a very high decoding probability is not rational.

Figure 5.14: Optimal first stage request ratio when $k = 100$ and $k = 400$.

5.4 Delay-Decodability Tradeoff

We now investigate the fundamental tradeoff between the targeted decoding probability and the file-retrieval delay for different request schemes. This delay-decodability tradeoff is important since it helps users to make rational request scheme depending on the requirement of their applications.

Figure 5.15: Tradeoff between decoding probability and delay when $k = 100$ and $k = 400$.

In Fig. 5.15, we plot the delay-decodability tradeoff for $k = 100$ and $k = 400$ respectively. The numerical result is obtained by combining (Eq. 5.6) and (Eq. 5.11). Since traffic parameters do not have strong affect the optimal retrieval delay (refer to the figures shown in Section. 5.2), we choose $\lambda_a = 30, \theta = 50$ as an example. As observed in these figures, the delay for one-stage request increases sharply when the decoding probability crosses 95%. In comparison, for the two-stage request scheme, the delay is consistently lower. This advantage reflects the gain from being able to decode in the first stage with an adequate instead of excessive number of LT-coded packets. Based on the reported delay-decodability tradeoff, user can choose an appropriate target of decodability probability, *i.e.* for some delay sensitive applications like stream video, a lower decodability probability can be adopted such that the average retrieval delay can be maintained under a certain threshold.

5.5 General Traffic Model

In Sections 5.3 and 5.4, we assume that the arrival process of ambient traffic follows Poisson process. In this section, we use a different traffic model to examine how our two-stage request scheme performs.

The Markov-modulated Poisson process (MMPP) [105] has been extensively used for modeling processes whose arrival rates vary randomly over time [106]. We use MMPP to model the ambient traffic here since it is more general than simple Poisson process and can capture some features to study burstiness of ambient traffic.

In our simulations, the MMPP traffic has arrival rates of λ_1 and λ_2 for the two phases, and an exponentially distributed rate switching duration with a mean value equal to 0.1s. We fix $\theta = 50$ and vary the value of λ_1 and λ_2 . The simulation results are shown in Figure 5.16 where Figure 5.16.a has $k = 100, n = 200$ and Figure 5.16.b has $k = 400, n = 700$.

Figure 5.16: Simulation result of MMPP model.

Comparing with the simulation results shown in Figure 5.8 and Figure 5.9, we can see that our two-stage request scheme exhibits the same behavior even when the bursty MMPP traffic model is used. This observation confirms the optimal request in first stage in Eq. (Eq. 5.11) which is independent of the property of ambient traffic. We believe that different models of ambient traffic only affect on the overall delay while using our two-stage request scheme can always give the minimal delay.

5.6 Conclusion

In this chapter, we investigate the problem of delay optimal file-retrieval under a distributed cloud storage system. The file is first LT-encoded and spread out into a list of distributed storage nodes. During retrieval, the user schedules the packet request in a multi-stage manner, with an objective to minimize the average file-retrieval delay. We developed an accurate model to characterize the average file-retrieval delay for different request strategies. Using this model, we derived an optimal two-stage request scheme for a given decoding probability. Both simulation and numerical results confirm that this optimal scheme can reduce the average delay dramatically. Our analysis offers a way for storage system operators to design an optimized storage retrieval scheme for LT-based distributed cloud storage system.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis, we first investigated the high overhead problem for LT decoding when the number of symbols is small. Our close investigation of LT decoding revealed that the LT decoding process often terminates prematurely even when the other yet-to-be-decoded symbols can be recovered by some other suitable decoding algorithms. We proposed a decoding algorithm called LT-W that incorporates the Wiedemann's method into the LT decoding process to extend its decodability.

LT-W decoding ensures successful decoding when a full-rank coefficient matrix is received. We analyzed the rank behavior of a random coefficient matrix which shows that full rank can be achieved with low packet overheads. This result motivates the design of decoding algorithms like ours that can achieve successful decoding given a full rank coefficient matrix. In our design, we introduced the concept of borrowed symbols in the LT decoding process. Whenever the LT decoding process terminates prematurely, a customized Wiedemann's method is called to recover a new symbol, and this new symbol is placed into the ripple so that the LT decoding process can continue.

To confirm the efficiency of our proposed solution, we analyzed the packet overhead

and computational efficiency in terms of the expected number of borrowed symbols. Our analysis confirms that the computational efficiency of LT-W decoding is largely dominated by that of LT decoding. We further compared the runtime between LT-W and LT, with results showing a shorter runtime for LT-W when the number of input symbols is small. Our results confirmed that the proposed LT-W decoding algorithm not only exhibits lower packet overheads, but also maintains high computational efficiency.

Two applications applying LT codes, layered content delivery and LT-based cloud storage system are also discussed.

When applying LT codes in layered content delivery, the main problem is unequal loss protection for different layers with different priorities. While LT codes treat all the symbols equally, we presented a novel joint ULP and LT coding for layered content delivery. The idea is to give a small amount of overhead for ULP to guarantee the priorities among the layers while using LT coding to deal with varying channel conditions. We have discussed the feasibility of joint ULP and LT decoding and how to perform FEC allocation. Our simulation results demonstrated that the proposed scheme can achieve good intermediate performance with controllable overhead.

In LT-based cloud storage system, we investigated the problem of delay optimal file-retrieval. In such a system, the file is first LT-encoded and spread out into a list of distributed storage nodes. When retrieval, the user schedules the packet request in a multi-stage manner, with an objective to minimize the average file-retrieval delay. We developed an accurate model to characterize the average file-retrieval delay for different request strategies. Using this model, we derived an optimal two-stage request scheme for a given decoding probability. Both simulation and numerical result confirm that this optimal scheme can reduce the average delay dramatically. Our analysis offers a way for storage system operators to design an optimized storage retrieval scheme for LT-based distributed cloud storage system.

6.2 Future Directions

Firstly, rateless codes such as LT codes and Raptor codes work only with binary erasure channels where packets are lost with some probability. However, in other channel models like Additive White Gaussian Noise (AWGN) channel and Binary Symmetric Channel (BSC), the error is not packet loss but bit-flip inside packet. To apply rateless codes in these channels, additional checksums must be added in encoded packets for decoder to check integrity. If received encoded packet is corrupted, it would be treated as lost during transmission. This would certainly hurt the channel throughput. Some pioneer works [107, 108, 109] studied the performance of rateless codes over AWGN channel and BSC. While in [110, 111], layered codes which combine existing fixed-rate codes to generate encoded packets in a rateless manner over AWGN channel were proposed. These works all attempted to extend the compatibility of rateless codes over AWGN channel and BSC. However, can rateless codes achieve Shannon capacity over AWGN channel or BSC? This fundamental problem remains open for us to explore.

Secondly, unlike fixed-rate codes which require determined number of encoded packets for decoding, the number of encoded packets need for LT decoding is probabilistic. However, in storage systems, we want to know exactly how many encoded packets are needed for recovering original data. To achieve this, only exhaustive search method can be used to find a deterministic LT encoding scheme [17]. However, it is not clear whether exhaustive search method works efficiently or even correctly [112]. To promote LT codes in storage systems, a systematic design of LT codes which can achieve deterministic decoding performance is urgently required. In [113], Wang proposed array BP-XOR codes by employing the underlying ideas of efficient Belief Propagation (BP) decoding process in LT codes. This array BP-XOR codes can be considered as deterministic version of LT codes. Although this new class of codes is maximum distance separable and can

be decoded in linear time, the codes can only support degree one or two encoded packets. Moreover, the codes are constructed based on specific storage system setups. Once the setup is changed, we need to reconstruct the codes. Fully flexible and deterministic rateless codes are still in the wild.

Lastly, our optimal retrieval scheme for LT-based cloud storage system presented in chapter 5 assumed that there were only one LT user in the system. In reality, there might be a group of LT users in the system who want retrieve their own data or shared data. In the case of retrieving shared data, the solution is trivial since portal can be considered as a multicast server and the problem is transferred to multicast using LT codes. In the case of retrieving own data, it is not clear whether the tradeoff presented in chapter 5 still exists in that scenario. Each user wants minimizing its own retrieval delay and simple greedy strategy might be not optimal since the bandwidth in the portal is limited. There might be a balance point for the users to obtain acceptable delay performance while the portal is not congested. One possible tool to investigate this problem is game theory [114].

Author's Publications

Journal Papers

- J1 **Haifeng Lu**, Chuan Heng Foh, Yonggang Wen, Jianfei Cai, “Delay-Optimized File Retrieval under LT-Based Cloud Storage”, *IEEE Transactions on Cloud Computing*
- J2 **Haifeng Lu**, Feng Lu, Chuan Heng Foh, Jianfei Cai, “LT-W: Improving LT Decoding with Wiedemann Solver”, *IEEE Transaction on Information Theory*.

Conference Papers

- C1 **Haifeng Lu**, Chuan Heng Foh, Jianfei Cai, “Scalable Data Dissemination Protocol for Wireless Sensor Networks” *IEEE ICON*, pp. 471-476, 2012.
- C2 **Haifeng Lu**, Chuan Heng Foh, Yonggang Wen, Jianfei Cai, “Optimal File Retrieval Scheme for LT Based Multimedia Cloud Computing System”, *IEEE GLOBECOM*, pp. 1920-1925, 2012.
- C3 **Haifeng Lu**, Jianfei Cai, Chuan Heng Foh, “Joint Unequal Loss Protection and LT Coding for Layer-Coded Media Delivery”, *IEEE GLOBECOM*, 1-5, 2010.

References

- [1] C. E. Shannon, “A mathematical theory of communication,” *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [2] R. W. Hamming, “Error detecting and error correcting codes,” *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.
- [3] A. Hocquenghem, “Codes correcteurs derreurs,” *Chiffres*, vol. 2, no. 2, pp. 147–156, 1959.
- [4] R. C. Bose and D. K. Ray-Chaudhuri, “On a class of error correcting binary group codes,” *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.
- [5] R. C. Bose and D. K. Ray-Chaudhuri, “Further results on error correcting binary group codes,” *Information and Control*, vol. 3, no. 3, pp. 279–290, 1960.
- [6] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [7] R. G. Gallager, “A simple derivation of the coding theorem and some applications,” *IEEE Transactions on Information Theory*, vol. 11, no. 1, pp. 3–18, 1965.
- [8] R. G. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.
- [9] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, “A digital fountain approach to reliable distribution of bulk data,” *ACM SIGCOMM Computer Communication Review*, vol. 28, no. 4, pp. 56–67, 1998.

REFERENCES

- [10] M. Luby, “LT codes,” in *IEEE Symposium on Foundations of Computer Science*. IEEE, 2002, pp. 271–280.
- [11] A. Mahanti, D. L. Eager, M. K. Vernon, and D. J. Sundaram-Stukel, “Scalable on-demand media streaming with packet loss recovery,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, pp. 195–209, 2003.
- [12] H. Weatherspoon and J. Kubiatowicz, “Erasure coding vs. replication: A quantitative comparison,” *Peer-to-Peer Systems*, pp. 328–337, 2002.
- [13] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, “Distributed fountain codes for networked storage,” in *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 5. IEEE, 2006, pp. V–V.
- [14] H. Xia and A. A. Chien, “Robustore: a distributed storage architecture with robust and high performance,” in *Proceedings of ACM/IEEE conference on Supercomputing*. ACM, 2007, pp. 1–11.
- [15] S. A. Aly, Z. Kong, and E. Soljanin, “Raptor codes based distributed storage algorithms for wireless sensor networks,” in *IEEE International Symposium on Information Theory*. IEEE, 2008, pp. 2051–2055.
- [16] Z. Kong, S. Aly, and E. Soljanin, “Decentralized coding algorithms for distributed storage in wireless sensor networks,” *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 261–267, 2010.
- [17] N. Cao, S. Yu, Z. Yang, W. Lou, and Y. T. Hou, “LT codes-based secure and reliable cloud storage service,” in *Proceedings of IEEE INFOCOM*. IEEE, 2012, pp. 693–701.
- [18] L. Rizzo, “Effective erasure codes for reliable computer communication protocols,” *ACM Computer Communication Review*, vol. 27, no. 2, pp. 24–36, 1997.
- [19] J. Bloemer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, “An XOR-based erasure-resilient coding scheme,” International Computer Science Insitution, Berkeley, California, Tech. Rep., 1995.

REFERENCES

- [20] M. Mitzenmacher, “Digital fountains: A survey and look forward,” in *IEEE Information Theory Workshop*. IEEE, 2004, pp. 271–276.
- [21] J. W. Byers, M. Luby, and M. Mitzenmacher, “A digital fountain approach to asynchronous reliable multicast,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1528–1540, 2002.
- [22] A. Shokrollahi, “Raptor codes,” *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.
- [23] B. Raghavan and A. C. Snoeren, “Decongestion control,” in *Proceedings of the Fifth Workshop on Hot Topics in Networks*, 2006, pp. 61–66.
- [24] P. Schelkens, A. Skodras, and T. Ebrahimi, *The JPEG 2000 suite*. Wiley, 2009, vol. 15.
- [25] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the scalable h.264/mpeg4-avc extension,” in *IEEE International Conference on Image Processing*. IEEE, 2006, pp. 161–164.
- [26] F. Lu, C. H. Foh, J. Cai, and L.-T. Chia, “LT codes decoding: Design and analysis,” in *IEEE International Symposium on Information Theory*. IEEE, 2009, pp. 2492–2496.
- [27] J. E. Gentle, *Numerical Linear Algebra with Applications in Statistics*. Springer, 1998.
- [28] R. Karp, M. Luby, and A. Shokrollahi, “Finite length analysis of LT codes,” in *IEEE Proceedings of International Symposium on Information Theory*. IEEE, 2004, p. 39.
- [29] D. Wiedemann, “Solving sparse linear equations over finite fields,” *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 54–62, 1986.
- [30] E. R. Berlekamp, *Algebraic coding theory*. McGraw-Hill New York, 1968, vol. 111.

REFERENCES

- [31] J. Massey, “Shift-register synthesis and bch decoding,” *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.
- [32] G. Gong. (1999) Sequence analysis. [Online]. Available: <http://www.karlin.mff.cuni.cz/~hojsik/tk/seq-main.pdf>
- [33] R. A. Horn and C. R. Johnson, *Matrix analysis*. Cambridge University Press, 2012.
- [34] E. Hyytia, T. Tirronen, and J. Virtamo, “Optimal degree distribution for LT codes with small message length,” in *IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 2576–2580.
- [35] L. Li and J. Zhao, “LT codes with a new degree distribution,” in *IEEE International Conference on Multimedia Information Networking and Security*. IEEE, 2010.
- [36] C. Studholme and I. Blake, “Windowed erasure codes,” in *IEEE International Symposium on Information Theory*. IEEE, 2006, pp. 509–513.
- [37] E. A. Bodine and M. K. Cheng, “Characterization of luby transform codes with small message size for low-latency decoding,” in *IEEE International Conference on Communications*. IEEE, 2008, pp. 1195–1199.
- [38] S. Kokalj-Filipovic, P. Spasojevic, and E. Soljanin, “Doped fountain coding for minimum delay data collection in circular networks,” *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 5, pp. 673–684, 2009.
- [39] A. Shokrollahi and M. Luby, *Raptor codes*. Now Publishers Inc, 2011.
- [40] P. G. Sherwood and K. Zeger, “Progressive image coding on noisy channels,” in *Proceedings of Data Compression Conference*. IEEE, 1997, pp. 72–81.
- [41] N. Thomos, N. V. Boulgouris, and M. G. Strintzis, “Wireless image transmission using turbo codes and optimal unequal error protection,” *IEEE Transactions on Image Processing*, vol. 14, no. 11, pp. 1890–1901, 2005.

- [42] X. Pan, A. Cuhadar, and A. H. Banihashemi, “Combined source and channel coding with jpeg2000 and rate-compatible low-density parity-check codes,” *IEEE Transactions on Signal Processing*, vol. 54, no. 3, pp. 1160–1164, 2006.
- [43] S. S. Arslan, P. C. Cosman, and L. B. Milstein, “Concatenated block codes for unequal error protection of embedded bit streams,” *IEEE Transactions on Image Processing*, vol. 21, no. 3, pp. 1111–1122, 2012.
- [44] Y. Huo, M. El-Hajjar, R. G. Maunder, and L. Hanzo, “Layered wireless video relying on minimum-distortion inter-layer fec coding,” *IEEE Transactions on Multimedia*, vol. 16, no. 3, pp. 697–710, 2014.
- [45] Y. Huo, C. Hellge, T. Wiegand, and L. Hanzo, “A tutorial and review on inter-layer fec coded layered video streaming,” *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, pp. 1–1, 2015.
- [46] A. Dimakis, J. Wang, and K. Ramchandran, “Unequal growth codes: Intermediate performance and unequal error protection for video streaming,” in *IEEE 9th Workshop on Multimedia Signal Processing*. IEEE, 2007, pp. 107–110.
- [47] S. K. Chang, K. C. Yang, and J. S. Wang, “Unequal-protected LT code for layered video streaming,” in *IEEE International Conference on Communications*. IEEE, 2008, pp. 500–504.
- [48] S. Arslan, P. Cosman, and L. Milstein, “Generalized unequal error protection LT codes for progressive data transmission,” *IEEE Transactions on Image Processing*, vol. 21, no. 8, pp. 3586–3597, Aug 2012.
- [49] J. Zao, M. Hornansky, P.-L. Diao, B.-W. Wang, C.-H. Wang, and L.-J. Chang, “Optimized rateless uep codes for scalable video streaming,” in *Visual Communications and Image Processing (VCIP), 2012 IEEE*, Nov 2012, pp. 1–6.
- [50] S. S. Woo and M. K. Cheng, “Prioritized LT codes,” in *Conference on Information Sciences and Systems*. IEEE, 2008, pp. 568–573.

REFERENCES

- [51] I. Hussain, M. Xiao, and L. K. Rasmussen, "Design of LT codes with equal and unequal erasure protection over binary erasure channels," *IEEE communications letters*, vol. 17, no. 2, pp. 261–264, 2013.
- [52] F. C. Zhang, X. Y. Duan, and M. Dong, "Construction of LT codes with unequal erasure protection for rateless data transmission," in *Wireless and Optical Communications Networks (WOCN), 2013 Tenth International Conference on*, July 2013, pp. 1–5.
- [53] K.-K. Yen, Y.-C. Liao, C.-L. Chen, J. K. Zao, and H.-C. Chang, "Integrating non-repetitive LT encoders with modified distribution to achieve unequal erasure protection," *IEEE Transactions on Multimedia*, vol. 15, no. 8, pp. 2162–2175, 2013.
- [54] S. M. Mirrezaei, K. Faez, and S. Yousefi, "Analysis and design of a new fountain codec under belief propagation," *IET Communications*, vol. 8, no. 1, pp. 27–40, 2014.
- [55] D. Sejdinovic, D. Vukobratovic, A. Doufexi, V. Senk, and R. Piechocki, "Expanding window fountain codes for unequal error protection," *IEEE Transactions on Communications*, vol. 57, no. 9, pp. 2510–2516, 2009.
- [56] D. Vukobratovic and V. Stankovic, "Unequal error protection random linear coding strategies for erasure channels," *IEEE Transactions on Communications*, vol. 60, no. 5, pp. 1243–1252, May 2012.
- [57] C. Ni, C. Hou, and W. Xiang, "A novel uep scheme based upon rateless codes," in *IEEE Wireless Communications and Networking Conference*. IEEE, 2012, pp. 592–596.
- [58] Z. Chen, L. Yin, M. Xu, and J. Lu, "Rateless codes with progressive recovery for layered multimedia delivery," in *IEEE Vehicular Technology Conference*. IEEE, 2012, pp. 1–5.
- [59] E. Namjoo, A. Aghagolzadeh, and J. Museviniya, "A new rateless code with unequal error protection property," *Computers & Electrical Engineering*, vol. 39, no. 7, pp. 1980–1992, 2013.

REFERENCES

- [60] Amazon S3. <http://aws.amazon.com/s3/>.
- [61] EMC Atmos Online Storage. <http://www.atmosonline.com/>.
- [62] L. Heilig and S. Voss, “A scientometric analysis of cloud computing literature,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 266–278, July 2014.
- [63] J.-W. Lin, C.-H. Chen, and J. Chang, “Qos-aware data replication for data-intensive applications in cloud computing systems,” *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 101–115, Jan 2013.
- [64] J. Hafner, “Weaver codes: highly fault tolerant erasure codes for storage systems,” in *The 4th conference on USENIX Conference on File and Storage Technologies*. USENIX Association, 2005, pp. 16–16.
- [65] J. Plank and L. Xu, “Optimizing cauchy Reed-Solomon codes for fault-tolerant network storage applications,” in *IEEE International Symposium on Network Computing and Applications*. IEEE, 2006, pp. 173–180.
- [66] C. Huang, M. Chen, and J. Li, “Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems,” in *IEEE International Symposium on Network Computing and Applications*. IEEE, 2007, pp. 79–86.
- [67] F. Oggier and A. Datta, “Self-repairing homomorphic codes for distributed storage systems,” in *Proceedings of IEEE INFOCOM*. IEEE, 2010, pp. 1215–1223.
- [68] Symform. <http://www.symform.com/>.
- [69] D. Connor, P. H. Corrigan, J. E. Bagley, and S. S. NOW, “Cloud storage: Adoption, practice and deployment,” *An Outlook Report from Storage Strategies NOW*, pp. 11–42, 2011.
- [70] M. Sardari, R. Restrepo, F. Fekri, and E. Soljanin, “Memory allocation in distributed storage networks,” in *IEEE International Symposium Information Theory*. IEEE, 2010, pp. 1958–1962.

REFERENCES

- [71] D. Leong, A. Dimakis, and T. Ho, “Distributed storage allocation for high reliability,” in *IEEE International Conference on Communications*. IEEE, 2010, pp. 1–6.
- [72] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.
- [73] D. Leong, A. Dimakis, and T. Ho, “Distributed storage allocations for optimal delay,” in *IEEE International Symposium Information Theory*. IEEE, 2011, pp. 1447–1451.
- [74] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, “Design and implementation of the sun network filesystem,” in *Proceedings of the Summer 1985 USENIX Conference*, 1985, pp. 119–130.
- [75] M. Nelson, B. Welch, and J. Ousterhout, “Caching in the sprite network file system,” *ACM Transactions on Computer Systems*, vol. 6, no. 1, pp. 134–154, 1988.
- [76] P. Corbett and D. Feitelson, “The vesta parallel file system,” *ACM Transactions on Computer Systems*, vol. 14, no. 3, pp. 225–264, 1996.
- [77] N. Nieuwejaar and D. Kotz, “The galley parallel file system,” *Parallel Computing*, vol. 23, no. 4-5, pp. 447–476, 1997.
- [78] P. Carns, W. Ligon III, R. Ross, and R. Thakur, “PVFS: A parallel file system for linux clusters,” in *Proceedings of the 4th annual Linux Showcase & Conference*. USENIX Association, 2000, pp. 391–430.
- [79] F. B. Schmuck and R. L. Haskin, “GPFS: A shared-disk file system for large computing clusters,” in *FAST*, vol. 2, 2002, p. 19.
- [80] M. Dunn and A. Reddy, *A new I/O scheduler for solid state devices*. Texas A&M University, 2010.

- [81] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum *et al.*, “The case for ramclouds: scalable high-performance storage entirely in dram,” *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 92–105, 2010.
- [82] Y. Wang, “LT codes for efficient and reliable distributed storage systems revisited,” *arXiv preprint arXiv:1207.5542*, 2012.
- [83] R. Zhang, C. Lin, K. Meng, and L. Zhu, “Bec: A reliable and efficient mechanism for cloud storage service,” in *IEEE International Conference on Communication Technology*. IEEE, 2013, pp. 717–721.
- [84] R. L. Rivest, “All-or-nothing encryption and the package transform,” in *Fast Software Encryption*. Springer, 1997, pp. 210–218.
- [85] M. Baldi, N. Maturo, E. Montali, and F. Chiaraluce, “AONT-LT: A data protection scheme for cloud and cooperative storage systems,” in *International Conference on High Performance Computing & Simulation*. IEEE, 2014, pp. 566–571.
- [86] R. Zhang, C. Lin, K. Meng, and L. Zhu, “A modeling reliability analysis technique for cloud storage system,” in *IEEE International Conference on Communication Technology*. IEEE, 2013, pp. 32–36.
- [87] C. Anglano, R. Gaeta, and M. Grangetto, “Exploiting rateless codes in cloud storage systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–11, 2014.
- [88] S. Kullback and R. Leibler, “On information and sufficiency,” *Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.
- [89] J. Cai, X. Li, and C. W. Chen, “Layered unequal loss protection with pre-interleaving for fast progressive image transmission over packet-loss channels,” *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 1, no. 4, pp. 338–353, 2005.

REFERENCES

- [90] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Prentice Hall Professional Technical Reference, 1990.
- [91] A. E. Mohr, E. A. Riskin, and R. E. Ladner, “Unequal loss protection: graceful degradation of image quality over packet erasure channels through forward error correction,” *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 819–828, June. 2000.
- [92] S. P. Vanderwiel and D. J. Lilja, “Data prefetch mechanisms,” *ACM Computing Surveys*, vol. 32, no. 2, pp. 174–199, 2000.
- [93] V. Gupta, M. Harchol Balter, K. Sigman, and W. Whitt, “Analysis of join-the-shortest-queue routing for web server farms,” *Performance Evaluation*, vol. 64, no. 9, pp. 1062–1081, 2007.
- [94] M. Bjorkqvist, L. Chen, M. Vukolic, and X. Zhang, “Minimizing retrieval latency for content cloud,” in *Proc. IEEE INFOCOM*, April 2011, pp. 1080–1088.
- [95] B. Guan, J. Wu, Y. Wang, and S. Khan, “Civsched: A communication-aware inter-vm scheduling technique for decreased network latency between co-located vms,” *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 320–332, July 2014.
- [96] T. Benson, A. Akella, and D. Maltz, “Network traffic characteristics of data centers in the wild,” in *Proceedings of the 10th Annual Conference on Internet Measurement*. ACM, 2010, pp. 267–280.
- [97] Z. Wilcox-O’Hearn and B. Warner, “Tahoe: the least-authority filesystem,” in *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability*, 2008, pp. 21–26.
- [98] A. Rényi, “On the theory of order statistics,” *Acta Mathematica Hungarica*, vol. 4, pp. 191–231, 1953.
- [99] E. Maneva and A. Shokrollahi, “New model for rigorous analysis of LT-codes,” in *IEEE International Symposium Information Theory*, 2006, pp. 2677–2679.

REFERENCES

- [100] D. Gross, J. F. Shortle, J. M. Thompson, and C. M. Harris, *Fundamentals of queueing theory*. Wiley. com, 2013.
- [101] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth, “On the lambert w function,” *Advances in Computational Mathematics*, vol. 5, no. 1, pp. 329–359, 1996.
- [102] F. N. Fritsch, R. E. Shafer, and W. P. Crowley, “Solution of the transcendental equation $we^w = x$,” *Communications of the ACM*, vol. 16, pp. 123–124, February 1973.
- [103] D. A. Barry, P. J. Culligan-Hensley, and S. J. Barry, “Real values of the w-function,” *ACM Transactions on Mathematical Software*, vol. 21, pp. 161–171, June 1995.
- [104] A. Hoorfar and M. Hassani, “Inequalities on the lambert w function and hyperpower function,” *Journal of Inequalities in Pure and Applied Mathematics*, vol. 9, no. 2, 2008.
- [105] D. R. Cox, “Some statistical methods connected with series of events,” *Journal of the Royal Statistical Society*, pp. 129–164, 1955.
- [106] W. Fischer and K. Meier-Hellstern, “The markov-modulated poisson process (mmp) cookbook,” *Performance Evaluation*, vol. 18, no. 2, pp. 149–171, 1993.
- [107] R. Palanki and J. S. Yedidia, “Rateless codes on noisy channels,” in *IEEE International Symposium on Information Theory*. IEEE, 2004, pp. 37–37.
- [108] E. Soljanin, N. Varnica, and P. Whiting, “Incremental redundancy hybrid arq with ldpc and raptor codes,” *IEEE Transactions on Information Theory*, pp. 1–37, 2005.
- [109] R. Barron, C. Lo, and J. Shapiro, “Global design methods for raptor codes using binary and higher-order modulations,” in *IEEE Military Communications Conference*, 2009, pp. 1–7.
- [110] A. Gudipati and S. Katti, “Strider: Automatic rate adaptation and collision handling,” in *ACM SIGCOMM*. ACM, 2011, pp. 158–169.

REFERENCES

- [111] U. Erez, M. D. Trott, and G. W. Wornell, “Rateless coding for gaussian channels,” *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 530–547, 2012.
- [112] J. Plank and G. Thomason, Michael, “A practical analysis of low-density parity-check erasure codes for wide-area storage applications,” in *International Conference on Dependable Systems and Networks*. IEEE, 2004, pp. 115–124.
- [113] Y. Wang, “LT codes for efficient and reliable distributed storage systems revisited,” *arXiv preprint arXiv:1207.5542*, 2012.
- [114] M. J. Osborne, *A course in game theory*. Cambridge, Mass.: MIT Press, 1994.