Ph.D. DISSERTATION

Applications of Sparse Codes: Batched Zigzag Fountain Codes and WOM Codes

저밀도 부호의 응용: 묶음 지그재그 파운틴 부호와 WOM 부호

BY

Jun Bohwan

FEBRUARY 2017

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Applications of Sparse Codes: Batched Zigzag Fountain Codes and WOM Codes

저밀도 부호의 응용: 묶음 지그재그 파운틴 부호와 WOM 부호

BY

Jun Bohwan

FEBRUARY 2017

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Applications of Sparse Codes: Batched Zigzag Fountain Codes and WOM Codes

저밀도 부호의 응용:
묶음 지그재그 파운틴 부호와 WOM 부호

지도교수 노 종 선
이 논문을 공학박사 학위논문으로 제출함

2017년 2월

서울대학교 대학원

전기 컴퓨터 공학부

전 보 환

전보환의 공학박사 학위 논문을 인준함

2017년 2월

위 원 장: _____
부위원장: _____
위 원: _____
위 원: _____
위 원: _____

Abstract

This dissertation contains the following two contributions on the applications of sparse codes.

- Fountain codes
  - Batched zigzag (BZ) fountain codes
  - Two-phase batched zigzag (TBZ) fountain codes
- Write-once memory (WOM) codes
  - WOM codes implemented by rate-compatible low-density generator matrix (RC-LDGM) codes

First, two classes of fountain codes, called batched zigzag fountain codes and two-phase batched zigzag fountain codes, are proposed for the symbol erasure channel. At a cost of slightly lengthened code symbols, the involved message symbols in each batch of the proposed codes can be recovered by low complexity zigzag decoding algorithm. Thus, the proposed codes have low buffer occupancy during decoding process. These features are suitable for receivers with limited hardware resources in the broadcasting channel. A method to obtain degree distributions of code symbols for the proposed codes via ripple size evolution is also proposed by taking into account the released code symbols from the batches. It is shown that the proposed codes outperform Luby transform codes and zigzag decodable fountain codes with respect to intermediate recovery rate and coding overhead when message length is short, symbol erasure rate is

low, and available buffer size is limited.

In the second part of this dissertation, WOM codes constructed by sparse codes are presented. Recently, WOM codes are adopted to NAND flash-based solid-state drive (SSD) in order to extend the lifetime by reducing the number of erasure operations. Here, a new rewriting scheme for the SSD is proposed, which is implemented by multiple binary erasure quantization (BEQ) codes. The corresponding BEQ codes are constructed by RC-LDGM codes. Moreover, by putting RC-LDGM codes together with a page selection method, writing efficiency can be improved. It is verified via simulation that the SSD with proposed rewriting scheme outperforms the SSD without and with the conventional WOM codes for single level cell (SLC) and multi-level cell (MLC) flash memories.

keywords: Fountain codes, low-density generator matrix (LDGM) codes, low-density parity-check (LDPC) codes, NAND flash memory, solid-state drive (SSD), write-once memory (WOM) codes.

student number: 2011-20923

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The fundamental goal of digital communication is *reliable transmission* of information such as speech, audio, and data from *source* to *sink* via noisy channel. When there are a source and a sink, it is called point-to-point communication problem. In 1948, Shannon [1] proved in a landmark paper that, by applying suitable encoding including *source and channel encoding*, errors induced by a noisy channel can be removed to any desired level without reducing the rate of information transmission, as long as the information rate is less than the *capacity* of the channel. It is also shown that the point-to-point communication problem can be decomposed into two separate problems, i.e., source and channel coding problems. The block diagram of the decomposed problem is shown in Fig. 1.1. The first part is the source encoder which transforms given information $\mathbf{m}$ into a bit stream $\mathbf{u}$. It removes all redundancy from the source, which results in the bit stream having the smallest size while reproduction of the source is still possi-

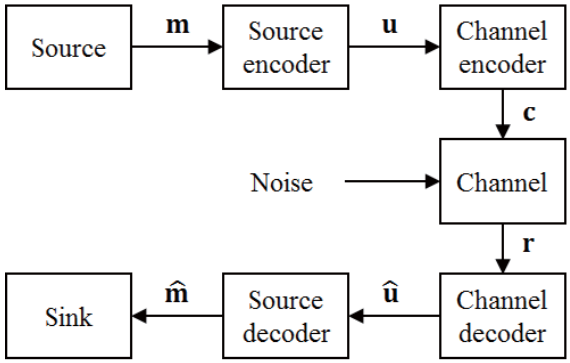

Figure 1.1: Block diagram of a point-to-point communication problem.

ble. In other words, it is impossible to compress the data such that the average number of bits per symbol is less than the entropy of the source. Secondly, the channel encoder adds redundancy, called *parity* bits, to the given bit stream, where the parity bits protect the bit stream from errors or erasures. Here, the encoded bit stream is called a *codeword* and denoted as $\mathbf{c}$. Then, the codeword is transmitted through the channel such as wired, wireless, and storage medium. After that, the channel decoder receives a vector $\mathbf{r}$ which may contain errors or erasures and tries to find which codeword was sent over the channel. The estimated codeword and the corresponding information are denoted by $\hat{\mathbf{c}}$ and $\hat{\mathbf{u}}$, respectively. Finally, the source decoder transforms the estimated information bit stream into $\hat{\mathbf{m}}$ and delivers it to the sink.

Unfortunately, Shannon used a random coding technique which is good for proving the channel capacity theorem, but hard to be implemented in practice. Thus, a number of constructions of implementable codes, i.e., codes with low encoding and decoding complexities, have been studied. A well known subclass of channel codes is a linear block code which contains codewords generated by the linear combination of basis

and thus is defined by the corresponding generator or parity-check matrices.

In 1962, *low-density parity-check (LDPC) codes* are invented by Gallager [2] as forward error correction (FEC) codes. However, their excellent potentials are undiscovered due to the limits of implementation in the 1960s. At that time, the probabilistic iterative decoding algorithm requiring high computational complexity was hard to be applied and thus LDPC codes were almost forgotten and neglected over 35 years. *Turbo codes* were first presented to the coding theorists by Berrou, Glavieux, and Thitimajshima [3] in 1993. They introduced a new family of convolutional codes built from concatenation of two recursive systematic codes with an interleaver. The name was derived from its iterative decoding algorithm, i.e., the part of decoded output is reintroduced at the input and processed again, like a turbo engine. After the appearance of turbo codes, LDPC codes were rediscroved by McKay and Neal [4] in 1996. They revealed that the iterative decoding algorithm for turbo codes was a special case of the decoding algorithm for Gallager's LDPC codes. Since then, LDPC codes have been one of the main research area due to their capacity-approaching performance.

From early 2000, generalization of LDPC codes and theoretical analysis of their performance have been studied by a number of researchers including Richardson, Urbanke, Luby, Mitzenmacher, Shokrollahi, and Spielman [5]–[8]. In addition to theoretical interests in LDPC codes, they have been adopted in many communication systems such as wireless metropolitan area network (WMAN), wireless local area network (WLAN), 10GBase-T Ethernet, and digital video broadcasting [9]–[12].

Similarly, another subclass of sparse codes, low-density generator matrix (LDGM) codes have been attracted attention [13], [14], where LDGM codes can be considered as dual codes of LDPC codes. However, it is shown that LDGM codes are asymptot-

ically bad since poor distance properties cause high error floor problems [15]. While maintaining the advantages of low complexity iterative decoding, the error floor problem of the LDGM codes can be solved by properly concatenating LDGM codes and another codes, which results in compound or concatenated codes such as MacKay-Neal (MN) codes [16].

Recently, alternative to fixed rate codes, concept of digital fountain is introduced [17], which allows any number of heterogeneous clients to acquire data with optimal efficiency even though channel state of each receiver is unknown to the transmitter. In 2002, Luby proposed practical fountain code with simple XOR operation [18], called *Luby transform (LT) codes*. In view of sparse codes, LT codes can be considered as LDGM codes. Therefore, LT codes may suffer from the error floor problem as mentioned earlier and thus *raptor codes* are invented by Shokrollahi [19], which are concatenation of the LT code and a precode.

While sparse codes such as LDPC codes, LDGM codes, and several fountain codes are used as error correction codes (ECC), they also have been applied to various areas due to their outstanding performance with low complexity decoding algorithm. The research areas adopting sparse codes are described as follows:

- ECC for the conventional communication systems
- Quantum error correction in quantum computing [20]–[22]
- Cryptosystem [23]–[26]
- Data compression [27]–[31]
- Compressed sensing [32]–[34]

- Peak-to-average power ratio (PAPR) control in orthogonal frequency division multiplexing (OFDM) systems [35]–[38]
- Write-once memory (WOM) codes [39], [40].

Especially in this dissertation, two classes of fountain codes defined by sparse graphs with zigzag coding technique and WOM codes implemented by rate-compatible (RC) LDGM codes are mainly discussed.

1.2 Overview of Dissertation

This dissertation is organized as follows. Chapter 2 introduces a well known subclass of sparse graph codes, i.e., LDPC codes. Firstly, basic notions and definitions of linear code are given in Section 2.1 and Section 2.2 reviews concepts of LDPC codes. Moreover, a simple message passing algorithm for LDPC code is briefly presented in Section 2.3.

In Chapter 3, two new classes of fountain codes are proposed, which can be regarded as sparse LDGM codes. Section 3.1 overviews LT codes and ZD codes. BZ fountain codes are proposed in Section 3.2. Degree distributions of code symbols for the proposed BZ fountain codes are derived in Section 3.3. Further, the TBZ fountain codes are proposed in Section 3.4. The performance improvement of the proposed codes is verified via numerical analysis in Section 3.5.

In Chapter 4, WOM codes constructed by RC-LDGM codes are introduced. Section 4.1 overviews characteristics of NAND flash memory, rewriting schemes for flash memory, and the construction of rewriting codes with BEQ codes. In Section 4.2, a

new rewriting scheme for NAND flash memory is proposed and a construction of the corresponding RC-LDGM codes is described in Section 4.3. The performance of the proposed WOM codes is verified via simulation in Section 4.4.

Chapter 2

Sparse Codes

In this chapter, some preliminaries of linear codes, sparse codes such as LDPC codes, and their message passing decoders are introduced. First, the basic concepts of linear codes are described and background of LDPC codes which are well known sparse codes are provided. Finally, a message passing decoder which has low complexity is introduced.

2.1 Linear Block Codes

In this section, we review definitions of linear codes and their properties. Let $\mathbb{F}_q$ be the finite field with q elements, where q is a prime power. Then a linear code is defined as follows.

Definition 2.1 *If $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a subspace of $\mathbb{F}_q^n$, then $\mathcal{C}$ is said to be a linear code.*

Since $\mathcal{C}$ is a subspace, there is a basis $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$, where k is the dimension of the subspace. It is known that any element in $\mathcal{C}$, called *codeword*, can be represented

as the linear combination of k basis vectors. Hence, the code $\mathcal{C}$, denoted by $(n, k)_q$, can be defined by its generator matrix.

Definition 2.2 *A matrix $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ is said to be a generator matrix of $\mathcal{C}$ if its k rows span $\mathcal{C}$.*

In other words, we have $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{q^k}\}$ and each codeword is given as $\mathbf{c}_i = \mathbf{m}_i\mathbf{G}$, where $\mathbf{m}_i$ is k-tuple vector, called message vector for $i = 1, \ldots, q^k$. Further, the rate of a code $\mathcal{C}$ is denoted by R given as

$$R = \frac{\log |\mathcal{C}|}{n \log |\mathbb{F}_q|} = \frac{k}{n}. \tag{2.1}$$

Also, the code $\mathcal{C}$ can be described in terms of a parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ such that $\mathbf{H}\mathbf{c}^\intercal = \mathbf{0}^\intercal$ for all codewords $\mathbf{c}$, where $\mathbf{0}$ denotes all zero row vector and $(\cdot)^\intercal$ represents the transpose. With the parity-check matrix, we define a syndrome vector of received vector $\mathbf{y}$ as follows.

Definition 2.3 *A vector $\mathbf{s} = \mathbf{H}\mathbf{y}^\intercal$ is said to be the syndrome of given vector $\mathbf{y}$.*

Clearly, a linear code is a subspace of $\mathbb{F}_q^n$ and thus the dual or orthogonal space can be defined.

Definition 2.4 *If $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a linear code, then its dual code $\mathcal{C}^\perp$ is defined as*

$$\mathcal{C}^\perp = \left\{\mathbf{z} \in \mathbb{F}_q^n | \mathbf{z}\mathbf{c}^\intercal = 0, \forall \mathbf{c} \in \mathcal{C}\right\}. \tag{2.2}$$

With above definitions, some properties of linear code are described as the following lemmas.

Lemma 2.5 *A parity-check matrix $\mathbf{H}$ for a code $\mathcal{C}$ is a generator matrix for the dual code $\mathcal{C}^\perp$.*

Lemma 2.6 $\dim(\mathcal{C}) + \dim(\mathcal{C}^{\perp}) = n$. *Thus, if* $\mathcal{C}$ *is an* $(n,k)_q$ *code, then* $\mathcal{C}^{\perp}$ *is an* $(n, n-k)_q$ *code.*

Lemma 2.7 $\left(\mathcal{C}^{\perp}\right)^{\perp} = \mathcal{C}$.

Here is a simple example of a linear code and its dual code.

Example 2.8 *Let* $\mathcal{C} = \{000, 110, 011, 101\}$ *be a* $(3,2)_2$ *linear code. Then the generator matrix is given as*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}. \tag{2.3}$$

Also the corresponding dual code $\mathcal{C}^{\perp}$ *is*

$$\begin{aligned} \mathcal{C}^{\perp} &= \left\{(v_1, v_2, v_3) | v_1 + v_2 = 0, v_2 + v_3 = 0, v_i \in \{0,1\} \text{ for } i = 1,2,3\right\} \\ &= \left\{(0,0,0), (1,1,1)\right\}. \end{aligned} \tag{2.4}$$

2.2 LDPC Codes

In this section, some preliminaries of LDPC codes are introduced. An LDPC code is a linear code which is defined by its parity-check matrix consisting of mostly 0's and a few nonzero elements. Especially, when the finite field is $\mathbb{F}_2$, the number of 1's in the matrix is very small compared to that of 0's, which results in a sparse matrix. An alternative representation of LDPC code is a bipartite graph $\mathcal{G} = (V, C, E)$, generally called *Tanner graph*, where V and C are sets of variable and check nodes and E is a set of edges connecting two nodes from different sets, i.e., V and C. Generally, a variable and a check nodes are represented as a circle and a square, respectively. Note that the

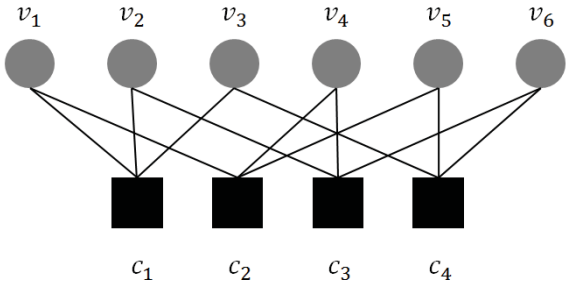

Figure 2.1: A Tanner graph of LDPC code.

number of edges in a node is called *degree*. An LDPC code who has variable and check nodes with only degree d_v and d_c, respectively is called (d_v, d_c) *regular LDPC codes*. Whereas, if the degrees of variable (check) nodes are different from each other, the code is said to be *irregular*.

Example 2.9 *A parity-check matrix of a binary* $(2, 3)$ *regular LDPC code with length 6 is given as*

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{2.5}$$

and the corresponding Tanner graph $\mathcal{G}_{\mathbf{H}}$ *is shown in Fig. 2.1. Here, it can be checked that* $V = \{v_1, v_2, \ldots, v_6\}$ *and* $C = \{c_1, c_2, c_3, c_4\}$.

2.3 Message Passing Decoder

In general, message passing decoding algorithm is used for LDPC codes, where its operations can be represented by passing the messages along the edges in a Tanner graph. It is also called *iterative decoder* since the outgoing messages from variable (check) nodes are put again into check (variable) nodes iteratively. The decoder stops until it finds a valid codeword or the number of iteration reaches the predetermined maximum iteration number.

In this dissertation, binary erasure channel (BEC) is mainly considered and thus a message passing algorithm for BEC is introduced. Note that a received bit is always correct if it is not erased. It is known that the message passing algorithm can be divided into three parts, i.e., variable node update part, check node update part, and decision part at each variable node. Let $M_{i,j}$ and $E_{j,i}$ be outgoing messages from the ith variable node to the jth check node and from the jth check node to the ith variable node, respectively. Initially, every variable node does not have any incoming messages. Hence, the outgoing message of the ith variable node is the same as its received value if ith bit is not erased, otherwise, the message is set as $*$, where $*$ denotes an erasure. Note that variable node sends the same outgoing messages to all neighboring check nodes. Now, the jth check node receives d_j incoming messages, where d_j is the number of neighbors of the jth check node. When only one message, passed from the ith variable node, among d_j incoming messages of the jth check node is an erasure, the check node calculates the value of unknown (erased) bit by adding all messages except for $M_{i,j}$ and sends the recovered value $E_{j,i}$ as a new outgoing message. However, when there are more than one incoming messages with $*$ at the check node, the check

node cannot recover erased bits. Thus, the check node sends the outgoing message as $*$. Finally, for the erased variable node, if any received message from the neighboring check node is not $*$, the erased bit at the variable node is recovered by the incoming message.

Remark 2.10 *When a message passing algorithm is performed sequentially, it is called peeling decoder. It is known that the performance of parallel and sequential message passing algorithms are the same in BEC.*

Chapter 3

New Fountain Codes with Improved Intermediate Recovery Based on Batched Zigzag Coding

Fountain codes are capacity-approaching codes which provide efficient transmission over point-to-multipoint channels. Luby transform (LT) codes [18] and Raptor codes [19] are the most popular practical fountain codes with low encoding and decoding complexities. In contrast to fixed rate codes, a fountain encoder generates k message symbols into sufficiently many code symbols and transmits them to multiple receivers without knowing their individual channel states until all receivers recover the k message symbols. Each receiver collects at least k code symbols and decodes them to recover all the message symbols with high probability.

To generate code symbols in LT codes, an encoder starts by randomly selecting the degree of each code symbol from a given *degree distribution*. Then the encoder selects the corresponding number of distinct message symbols uniformly at random and performs bitwise XOR operation on them. The code symbols of conventional LT codes are decoded by iteratively performing bitwise XOR operation on the received

code symbols with the recovered neighboring message symbols. Each decoder has two types of storages, *ripple* and *buffer*, which intermediately store the recovered message symbols and the received code symbols that are not fully decoded yet, respectively.

The ratio of the number of received code symbols to k is called *coding overhead*, which is one of the important performance criteria of LT codes. In order to reduce the coding overhead of LT codes, lots of research have been done. Since the coding overhead of LT codes is clearly related to the degree distribution and the size of the ripple, various degree distributions have been researched by using an analytic tool called *ripple size evolution*, which represents the expected ripple size during the decoding process [41]. In order to keep the average ripple size one, Luby proposed the ideal soliton distribution (ISD) [18] but a small variance of adding released code symbols into the ripple can cause decoding failure. To resolve this drawback, the robust soliton distribution (RSD) was proposed [18] to keep the average ripple size as a constant value larger than one. On the other hand, Sørensen *et al.* [41] proposed a new method to obtain degree distributions with decreasing ripple size. They show that the decreasing ripple size distribution (DRSD) provides a significant improvement of coding overhead. In addition, the degree distributions of LT codes which improve their performance are also studied in [42]–[44].

When code symbols with degree larger than one remain in the buffer and the ripple is empty, standard iterative decoding fails. Recently, a new class of fountain codes adopting *zigzag decodable* (ZD) structure was proposed in [45]–[47]. Using not only bitwise XOR operation but also bit-level shift operation, these fountain codes give us a performance improvement in terms of coding overhead. If all selected message symbols are bitwisely shifted and then combined by bitwise XOR operation when gener-

ating a code symbol, we can make the leftmost or the rightmost bit of the code symbol depend on only one of the selected message symbols. These bits can help decoding process further, which is called zigzag decoding. It is known that the ZD structures are adopted in various applications other than the fountain codes, e.g., wireless networks, distributed storage systems, and index coding [48]–[52].

Generally, most of the fountain codes are designed to minimize the coding overhead but they do not consider the *intermediate symbol recovery rate* (ISRR), which is defined as the ratio of the number of recovered message symbols to the number of total message symbols when the number of received code symbols is less than k. Low ISRR implies that the decoder has to store a lot of code symbols in the buffer during decoding process. Low ISRR is fatal for the receivers with limited memory and computational power due to their extremely low cost requirements such as electronic shelf labels in warehouse [53] and smart sensors, which are key ingredients in Internet of Things (IoT) [54]–[56]. At an intermediate stage of decoding, if the number of undecoded code symbols reaches the capacity of the buffer, the decoder randomly selects and discards one code symbol from the buffer. Then it receives a new code symbol. Clearly, the limitation of the buffer size causes the increase of the coding overhead.

There have been several works on the ISRR [57]–[63]. An upper bound of ISRR was derived and asymptotically optimal degree distributions for the ISRR were found in [57]. Talari and Rahnavard [59] designed degree distributions employing genetic algorithms to have the optimal ISRR. However, high portion of low degree code symbols, e.g., degrees one and two, causes high coding overhead. They also proposed the rateless coded symbol sorting (RCSS) algorithm for further ISRR improvement, which additionally requires $O(k^2)$ encoding complexity. Other approaches for im-

proving ISRR and reducing the usage of buffer were introduced in [60]–[63]. In these fountain coding schemes, the receivers frequently use feedbacks in order to inform the current decoding status to the transmitter, which makes it possible to search the optimal degree. However, for point-to-multipoint communication systems, it is not easy to use many feedback transmissions. Further, feedback signals from different receivers may collide with each other in wireless channel and the frequent retransmissions cause significant power consumption in the receivers.

In this chapter, a new class of fountain codes called *batched zigzag (BZ) fountain codes* is proposed to improve the ISRR without feedback. In terms of batched coding, encoding based on common message symbols is first introduced in batched sparse codes [64], which are concatenation of fountain codes and random linear codes over large finite field. On the contrary, in the proposed BZ fountain codes, batches are generated based on a common subset of message symbols with the bit-level shift and the bitwise XOR operations so that they can be zigzag decodable. It is noted that the bit-level shift operation in the proposed codes gives rise to a small amount of additional overhead, called *bit-level overhead*, which causes the effect of slightly lengthening each code symbol. We also analyze their bit-level overhead and computational complexity. While ZD fountain codes in [45] do not also consider ISRR performance, the proposed BZ fountain codes are designed to have a higher ISRR by generating zigzag decodable batches. Also, we extend the ripple size evolution analysis of LT codes in [41] to the proposed codes so that we derive new degree distributions for BZ fountain codes.

In order to reduce the coding overhead further, we modify the proposed BZ fountain codes to *two-phase batched zigzag (TBZ) fountain codes*. Encoding procedures of

the TBZ fountain codes are separated into two phases by the stored indices of the previously selected message symbols for generation of code symbols. At the first phase, the encoder picks previously unselected message symbols to generate the next code symbols and constructs batches with various sizes as in the BZ fountain codes. At the second phase, subsets of message symbols are randomly selected from the entire message symbols and batches with size one are generated. It is noted that information of all previously encoded symbols is stored in [65] but the memory at the encoder of TBZ fountain code stores whether message symbols are selected or not for the previous generation, which is much simpler.

Finally, the contributions of this chapter are summarized as follows: i) we proposed BZ fountain codes to improve ISRR, ii) we obtained a proper degree distribution for the proposed codes by using ripple size evolution analysis, iii) we also proposed TBZ fountain codes by splitting encoding procedure into two phases. We verify by numerical analysis that the proposed BZ fountain codes improve ISRR and the proposed TBZ fountain codes outperform LT codes and ZD fountain codes [45] with respect to ISRR and coding overhead when the available buffer size is limited.

3.1 Preliminaries

3.1.1 Definitions and Notation

We consider broadcasting k binary l-tuple message symbols from one transmitter to multiple receivers over symbol erasure channel with symbol erasure rate ϵ. Let $\mathbf{m}_i = (m_{i,1}, m_{i,2}, \ldots, m_{i,l})$ denote the ith message symbol, where $m_{i,j} \in \{0, 1\}$

for $i = 1, \ldots, k$ and $j = 1, \ldots, l$. The ith message symbol can also be represented in a polynomial form as

$$m_i(z) = m_{i,1} + m_{i,2}z + \cdots + m_{i,l}z^{l-1}. \tag{3.1}$$

Obviously, this setting can be seen as a transmission of k sequential packets each of which consists of l bits. We assume that each receiver has so strict memory limit that it can store at most $k\beta$ code symbols in the buffer, where $0 < \beta < 1$.

Let γ and μ denote the ratios of the number of received code symbols and recovered message symbols to the number of message symbols k, respectively. In a fountain code setting, collecting arbitrary $k\gamma$ code symbols at each receiver leads to recovering all message symbols with high probability, i.e., $\mu = 1$ for $\gamma \geq \gamma_{succ}$. Here, we call γ_{succ} as coding overhead. For capacity-achieving fountain codes, we have $\gamma_{succ} = 1$.

Degree of a code symbol $\mathbf{c}$ is the number of involved message symbols in the encoder and the corresponding message symbols are called *neighbors* of $\mathbf{c}$, denoted by $\mathcal{N}(\mathbf{c})$. The degree of code symbols is drawn from a certain degree distribution $\Omega(x) = \sum_{d=1}^{k} \Omega_d x^d$, where Ω_d is the probability that a code symbol has degree d.

3.1.2 LT Codes

In this subsection, we will briefly overview LT codes. An encoder of LT codes sequentially generates code symbols until every receiver recovers the k message symbols as follows. First, sample a degree d of code symbol from a given degree distribution. Secondly, choose d distinct message symbols uniformly at random out of k message symbols and perform bitwise XOR operation on the d chosen message symbols. The decoder of LT codes iterates the following procedures until all message symbols are

recovered.

Step 1) Store a newly received code symbol in the buffer.

Step 2) If the received code symbol is degree-one, add the neighboring message symbol into the ripple and go to Step 5). Otherwise, go to Step 1).

Step 3) Store a newly received code symbol in the buffer and perform bitwise XOR operations with already recovered neighboring message symbols if any.

Step 4) If the degree of the newly received code symbol becomes one, called released code symbol, add the remaining neighbor of the code symbol into the ripple.

Step 5) If the ripple is not empty, select a message symbol randomly from the ripple. Otherwise, go to Step 8).

Step 6) Perform bitwise XOR operations with the selected message symbol on every code symbol in the buffer who has the message symbol as a neighbor. Move the selected message symbol from the ripple to the memory of the recovered message symbols. The message symbol is called recovered.

Step 7) Find the released code symbols in the buffer and move the corresponding message symbols to the ripple if any. Then go to Step 5).

Step 8) Stop if all message symbols are successfully recovered. Otherwise, go to Step 3).

3.1.3 Zigzag Decodable Codes

In this subsection, we introduce ZD codes. These codes use not only bitwise XOR but also bit-level shift operation. As a result, the length of the code symbol is slightly larger than that of the message symbol l.

Definition 3.1 *A d-tuple vector $\mathbf{s}_i^d = (s_{i,1}, \ldots, s_{i,d})$ is a shift vector of the ith code symbol $\mathbf{c}_i$, where $s_{i,j}$ is a nonnegative integer representing the shift value of the jth neighbor of $\mathbf{c}_i$ for $j = 1, \ldots, d$. A matrix $\mathbf{S}_{T\times d} = \left[\left(\mathbf{s}_1^d\right)^\intercal \quad \left(\mathbf{s}_2^d\right)^\intercal \quad \ldots \quad \left(\mathbf{s}_T^d\right)^\intercal\right]^\intercal$ denotes a shift matrix of T code symbols, where $(\cdot)^\intercal$ denotes the transpose.*

Clearly, the shift vector of the conventional LT coding is *all-zero shift vector*, denoted by $\mathbf{s}_{(0)}^d = (0, 0, \ldots, 0)$ for degree-d code symbol. In general, it can be assumed that the minimum shift value in the shift vector $\mathbf{s}_i^d$ is equal to zero. The maximum shift value in $\mathbf{s}_i^d$ is defined as $s_{max,i}^d = \max_j\{s_{i,j}\}$. Then $s_{max,i}^d$ represents the number of additional bits for the ith code symbol due to the shift and XOR operations. Also, let $\delta = \max_{d,i}\{s_{max,i}^d\}$ be the maximum number of additional bits for all code symbols.

Definition 3.2 ([45]) *For a given δ_{R} and d, let $\hat{\mathbf{s}}^d = (\hat{s}_1, \ldots, \hat{s}_d)$ be a nonnegative integer vector such that $\hat{s}_i$ is selected from $[0, \delta_{\mathrm{R}}]$ uniformly at random for $i = 1, \ldots, d$. Find the minimum value in $\hat{\mathbf{s}}^d$, i.e., $\hat{s}_{min} = \min_{i\in\{1,\ldots,d\}}\{\hat{s}_i\}$ and subtract the minimum value from all elements in $\hat{\mathbf{s}}^d$. Then it is called a random shift vector, that is, $\mathbf{s}_{(\mathrm{R})}^d = (\hat{s}_1 - \hat{s}_{min}, \ldots, \hat{s}_d - \hat{s}_{min})$.*

From the shift matrix $\mathbf{S}_{T\times d} = [s_{i,j}]$, the generator matrix of the ZD codes is defined

as

$$\mathbf{G}_{T\times d}(z) = \begin{bmatrix} z^{s_{1,1}} & z^{s_{1,2}} & \dots & z^{s_{1,d}} \\ z^{s_{2,1}} & z^{s_{2,2}} & \dots & z^{s_{2,d}} \\ \vdots & \vdots & \ddots & \vdots \\ z^{s_{T,1}} & z^{s_{T,2}} & \dots & z^{s_{T,d}} \end{bmatrix}. \tag{3.2}$$

The zigzag encoding of d message symbols results in T code symbols and it can be represented in a matrix form as

$$\begin{bmatrix} c_1(z) & \dots & c_T(z) \end{bmatrix}^{\mathsf{T}} = \mathbf{G}_{T\times d}(z) \begin{bmatrix} m_{I_1}(z) & \dots & m_{I_d}(z) \end{bmatrix}^{\mathsf{T}} \tag{3.3}$$

where I_j is an index of the jth neighbor of the code symbols for $j = 1, \dots, d$ satisfying $1 \le I_1 < \cdots < I_d \le k$.

Nozaki [45] proposed the decoding algorithm for the fountain codes based on ZD codes using bitwise Tanner graph. First, the bitwise Tanner graph of the received code symbols is constructed. Then the LT decoding which is described in the previous subsection is applied bitwisely. We call the code symbols *zigzag decodable* if every involved message symbol is successfully recovered by bitwise LT decoding.

Example 3.3 *Consider the ZD code with $T = d = 3$, and $l = 5$. Also the shift matrix is set as $\mathbf{S}_{3\times 3} = \begin{bmatrix} (0,1,2)^{\mathsf{T}} & (1,2,0)^{\mathsf{T}} & (1,0,0)^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}}$, where clearly $\delta = 2$. The code symbols $\mathbf{c}_1, \mathbf{c}_2$, and $\mathbf{c}_3$ are graphically described in Fig. 3.1 and the corresponding bitwise Tanner graph $\mathcal{G}_{\mathrm{b}} = (M_{\mathrm{b}}, C_{\mathrm{b}}, E_{\mathrm{b}})$ is shown in Fig. 3.2, where M_{b} and C_{b} are sets of message and code bit nodes, respectively, and E_{b} is a set of edges such that $(m, c) \in E_{\mathrm{b}}$ for $m \in M_{\mathrm{b}}$ and $c \in C_{\mathrm{b}}$. The circle and the square nodes represent message and code bit nodes, respectively. Clearly, these code symbols are zigzag decodable, i.e., we can decode them in the following order, $m_{1,1}$, $m_{3,1}$, $m_{3,2}$, $m_{2,1}$, etc.*

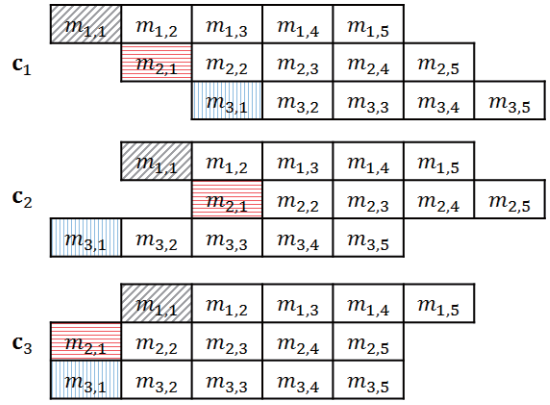

Figure 3.1: Code symbols of ZD code.

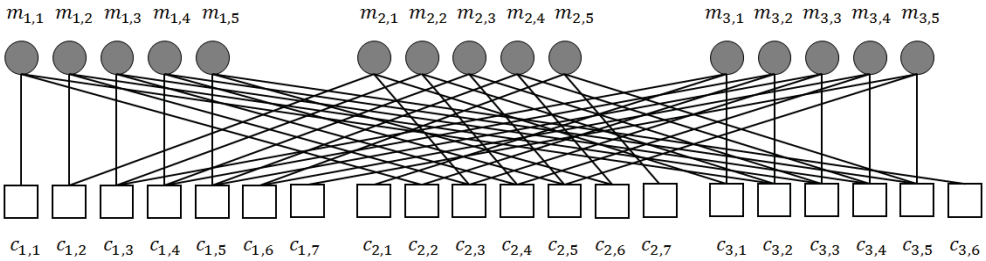

Figure 3.2: A bitwise Tanner graph of ZD codes.

3.1.4 Bit-Level Overhead

Let $L_{\Omega}^{(\mathcal{C})}$ be a random variable which represents the length of code symbol of the fountain code $\mathcal{C}$ with zigzag coding when $\Omega(x)$, l, and δ are given. Let r be the number of additional bits for code symbols of $\mathcal{C}$ and let $R_{\Omega}^{(\mathcal{C})}$ denote a random variable of r such that $L_{\Omega}^{(\mathcal{C})} = l + R_{\Omega}^{(\mathcal{C})}$.

Definition 3.4 *A bit-level overhead of the fountain code $\mathcal{C}$ with degree distribution $\Omega(x)$ is defined as*

$$\eta = \frac{\mathbb{E}\left[L_{\Omega}^{(\mathcal{C})}\right]}{l} \tag{3.4}$$

where $\mathbb{E}[\cdot]$ denotes the expectation and $1 \leq \eta \leq 1 + \delta/l$.

Obviously, for the conventional LT codes, $\eta = 1$ because $\delta = 0$. On the other hand, all code symbols in the ZD fountain codes proposed in [45] are generated by $\mathbf{s}_{(\mathrm{R})}^{d}$ and thus the expected length of the code symbols is derived in the following lemma.

Lemma 3.5 ([46], Corollary 1) *Consider ZD fountain codes with $\Omega(x)$, l, and δ_{R}. Then the expected length of the code symbols is given as*

$$\mathbb{E}\left[L_{\Omega}^{(\mathrm{ZD})}\right] = l + \delta_{\mathrm{R}} - 2\sum_{i=1}^{\delta_{\mathrm{R}}} \Omega\left(\frac{i}{\delta_{\mathrm{R}}+1}\right). \tag{3.5}$$

3.2 New Fountain Codes Based on Batched Zigzag Coding

We can construct a set of code symbols using the same subset of message symbols, called *batch*, i.e., $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_T\}$, where $\mathcal{N}(\mathbf{c}_1) = \cdots = \mathcal{N}(\mathbf{c}_T)$. The *size* and the *degree* of a batch denote the number of code symbols and the number of message symbols involved in each code symbol in the batch, respectively. We consider the fountain codes such that a transmitter generates code symbols in the form of batches and broadcasts them to a number of receivers. Clearly, with only bitwise XOR operation on any subset of message symbols, we can generate batches of size one. Here, we propose

new fountain codes, called batched zigzag fountain codes such that we can generate batches with size larger than or equal to one by using the bit-level shift and XOR operations on subsets of the message symbols, that is, adopting the zigzag coding for batches. Batched zigzag encoding and decoding denote the encoding of batches having zigzag structures and the decoding of batches with the zigzag decoding algorithm to recover all the involved message symbols, respectively.

3.2.1 Construction of Shift Matrix

There are various types of shift matrices for the zigzag encoding. Here we introduce a $d \times d$ shift matrix which can be used in the proposed codes.

Definition 3.6 ([50]) *A $d \times d$ shift matrix denoted by $\mathbf{S}_{d\times d}^{(\mathrm{EV})} = [s_{i,j}]$ is called an extended Vandermonde shift matrix for $1 \leq i \leq d$ and $1 \leq j \leq d$, where $s_{i,j} = p_i + (i-q)(j-1)$, $q = \lceil d/2 \rceil$, and*

$$p_i = \begin{cases} (q-i)(d-1), & \text{if } i \leq q \\ 0, & \text{otherwise.} \end{cases} \tag{3.6}$$

It is shown that for any d, every code symbol obtained by arbitrary square submatrix of $\mathbf{S}_{d\times d}^{(\mathrm{EV})}$ is zigzag decodable [50]. Further, by using $\mathbf{S}_{d\times d}^{(\mathrm{EV})}$, Chen *et al.* [50] reduce δ by at least 50% compared to the shift matrix proposed in [49]. Hence, the decoder can recover all neighboring message symbols from any t already recovered neighbors and $d-t$ received coded symbols in the same batch generated by $\mathbf{S}_{d\times d}^{(\mathrm{EV})}$.

Algorithm 3.1: Encoding of BZ fountain codes

Input: $k, \{m_1(z), \ldots, m_k(z)\}$, $\Psi(x)$, d_m, $\mathcal{S} = \{\mathcal{S}_2, \ldots, \mathcal{S}_{d_m}\}$

Initialization: $\mathcal{I} \leftarrow \{1, \ldots, k\}$

Step 1) Sample a degree d from $\Psi(x)$.

Step 2) If $1 \leq d \leq d_m$, $T \leftarrow d$. Otherwise, $T \leftarrow 1$.

Step 3) Select d indices of message symbols from $\mathcal{I}$ uniformly at random. $\mathcal{I}_{sel} \leftarrow \{I_1, \ldots, I_d\}$.

Step 4) If $T > 1$, $\mathbf{S}_{T\times d} = [s_{j,w}] \leftarrow \mathcal{S}_d$. Otherwise, $\mathbf{S}_{T\times d} \leftarrow \mathbf{s}^d_{(0)}$.

Step 5) $\mathbf{C}(z) \leftarrow \mathbf{G}_{T\times d}(z)\begin{bmatrix} m_{I_1}(z) & \cdots & m_{I_d}(z) \end{bmatrix}^{\mathsf{T}}$, where the entry of $\mathbf{G}_{T\times d}(z)$ is $g_{j,w} = z^{s_{j,w}}$ for $j = 1, \ldots, T$ and $w = 1, \ldots, d$.

Step 6) Stop if every receiver successfully recovers the k message symbols. Otherwise, go to Step 1).

3.2.2 Encoding and Decoding of the Proposed BZ Fountain Codes

Now we propose a new class of fountain codes based on batched zigzag coding as follows. Encoding and decoding procedures of the proposed BZ fountain codes are described in Algorithms 3.1 and 3.2, where $\Psi(x)$, d_m, and $\mathcal{S}$ denote the batch degree distribution, the maximum size of batch, and the set of predetermined $d \times d$ shift matrices for $2 \leq d \leq d_m$, respectively. Here, the batch degree distribution $\Psi(x)$ is given as $\sum_{d=1}^{k} \Psi_d x^d$, where Ψ_d is the probability that the batch has degree d.

With Algorithm 3.1, d code symbols are generated from the same subset of message symbols for $2 \leq d \leq d_m$ at one time and single code symbol is generated for $d = 1$ or $d > d_m$. Then the encoder broadcasts the generated code symbols one by one in the batches. The decoding algorithm for the proposed BZ fountain codes uses both the LT decoding and the zigzag decoding for the batches. Clearly, all the code symbols in the batch are zigzag decodable if all code symbols in the same batch are received without any erasures. Now we consider the case that t code symbols are erased in the channel when a batch $\mathbf{C}$ of size d is transmitted. Since the proposed decoding algorithm does not have any difference from the conventional LT decoding for $t = d - 1$, we focus on the case of $1 \leq t \leq d-2$ for $d \geq 3$. A batch $\mathbf{C}$ is zigzag decodable when the following two cases occur.

- Case 1) $t = 1$; Let $\mathbf{m}_{\text{cur}}$ be a message symbol currently selected from the ripple. If $\mathbf{m}_{\text{cur}} \in \mathcal{N}(\mathbf{C})$, then we remove $\mathbf{m}_{\text{cur}}$ from $\mathbf{C}$ by XOR with its code symbols to decrease the number of unrecovered message symbols to $d - 1$. Then, the batch of size $d-1$ with degree $d-1$ is zigzag decodable and the corresponding $d - 1$ message symbols are newly added into the ripple.

- Case 2) $2 \leq t \leq d - 2$; Assume that $t - 1$ message symbols out of d message symbols in $\mathcal{N}(\mathbf{C})$ are already recovered. If $\mathbf{m}_{\text{cur}} \in \mathcal{N}(\mathbf{C})$, then the batch of size $d-t$ with degree $d-t$ is zigzag decodable and the remaining $d-t$ message symbols are added into the ripple.

When the ripple becomes empty, a conventional LT decoder halts its decoding process and receives a new code symbol. On the other hand, for the BZ fountain codes, the recovery of message symbols can be additionally done by zigzag decoding, which

implies the increase of the ripple size and further recovery of the message symbols from the given number of the received code symbols. Consequently, the proposed BZ fountain codes have the improved ISRR.

Now, we derive the expected length of code symbols for the proposed BZ fountain codes. Throughout the chapter, we assume that the extended Vandermonde shift matrix $\mathbf{S}_{d\times d}^{(\mathrm{EV})}$ is used as the predetermined shift matrix $\mathcal{S}_d$. Here we derive the expected length of code symbols using the code symbol degree distribution $\Omega(x)$ rather than the batch degree distribution $\Psi(x)$, whose relation will be derived in the next section.

Theorem 3.7 *Consider BZ fountain codes with code symbol degree distribution $\Omega(x)$, l, d_m, and the set of predetermined shift matrices $\mathcal{S} = \left\{\mathbf{S}_{2\times 2}^{(\mathrm{EV})}, \ldots, \mathbf{S}_{d_m\times d_m}^{(\mathrm{EV})}\right\}$. Then the expected length of code symbols is given as*

$$\mathbb{E}\left[L_{\Omega}^{(\mathrm{BZ})}\right] = l + \sum_{d=2}^{d_m} h(d)\Omega_d \tag{3.7}$$

where $h(d)$ denotes the expected number of additional bits for the degree-d code symbol given by

$$h(d) = \begin{cases} \frac{d(d-1)}{4}, & \text{if } d \text{ is even} \\ \frac{(d^2-1)(d-1)}{4d}, & \text{if } d \text{ is odd} \end{cases} \tag{3.8}$$

and the maximum number of additional bits for a code symbol δ is $\left\lfloor d_m/2 \right\rfloor (d_m - 1)$.

Proof. For $2 \le d \le d_m$, there are three types of rows in $\mathbf{S}_{d\times d}^{(\mathrm{EV})}$, i.e., decreasing, constant, and increasing rows. The ith row belongs to decreasing, constant, and increasing rows if i is less than, equal to, and larger than $\lceil d/2 \rceil$, respectively. It is easy to check that $s_{max,i}^{d} = \left|\lceil d/2 \rceil - i\right| (d-1)$ for $1 \le i \le d_m$. Since there are total d code

symbols, $h(d)$ can be written as

$$h(d) = \frac{d-1}{d}\left(\sum_{i=1}^{\lceil d/2\rceil-1}\left(\lceil d/2\rceil - i\right) + \sum_{i=\lceil d/2\rceil+1}^{d}\left(i - \lceil d/2\rceil\right)\right) \tag{3.9}$$

which results in (3.8). Furthermore, we can show that

$$\delta = \max_{d,i}\{s^{d}_{max,i}\} = \left\lfloor d_m/2 \right\rfloor (d_m - 1) \tag{3.10}$$

by the definition. □

3.2.3 Storage and Computational Complexity

In this subsection, we analyze the storage and computational complexities of the proposed BZ fountain codes. In order to recover all message symbols, a receiver needs to store $k\gamma_{succ}$ code symbols, where each code symbol has $\mathbb{E}\left[L_{\Omega}^{(\mathrm{BZ})}\right]$ bits in average. For the worst case, the length of all received code symbols is equal to $l + \delta$. Thus, the BZ fountain codes require at most $k\gamma_{succ} \cdot (l + \delta)$ bits when buffer size is unlimited.

To compute the overall encoding and decoding complexities, we need to check the size (number of edges) of the bitwise Tanner graph for received code symbols. If the code symbol degree distribution is fixed to $\Omega(x)$, then there are $k\gamma_{succ} \cdot \mathbb{E}\left[L_{\Omega}^{(\mathrm{BZ})}\right] \cdot \Omega'(1)$ edges in the bitwise Tanner graph while it is given as $k\gamma_{succ} \cdot l \cdot \Omega'(1)$ in LT codes, where $\Omega'(x)$ denotes the derivative of $\Omega(x)$. Since $\delta = \left\lfloor d_m/2 \right\rfloor (d_m - 1)$, we choose d_m small enough, e.g., $d_m \leq 5$, which results in $\delta \leq 8$. Thus, by setting l strictly larger than δ, e.g., $l \geq 50$, the increased computational complexity of the proposed codes is negligible compared to that of the conventional LT codes when the same degree distribution is applied.

Algorithm 3.2: Decoding of BZ Fountain Codes

Step 1) Store a newly received code symbol in the buffer.

Step 2) If the received code symbol is degree-one, add the neighboring message symbol into the ripple and go to Step 6).

Step 3) If a batch is received without any erasure, perform the zigzag decoding to the batch and add all the neighboring message symbols into the ripple. Then go to Step 6). Otherwise, go to Step 1).

Step 4) Store a newly received code symbol in the buffer and perform bitwise XOR operations with already recovered neighboring message symbols.

Step 5) If the current degree of the code symbol is one, add the remaining neighbor of the code symbol, i.e., the message symbol into the ripple.

Step 6) If the ripple is not empty, select a message symbol randomly from the ripple. Otherwise, go to Step 9).

Step 7) Perform bitwise XOR operations with the selected message symbol to every code symbol in the buffer who has the message symbol as a neighbor. Move the selected message symbol from the ripple to the memory of the recovered message symbols.

Algorithm 3.2: Decoding of BZ Fountain Codes (continued)

Step 8) Find the released code symbols in the buffer and if any, move the corresponding code symbols, i.e., the message symbols to the ripple. Go to Step 6).

Step 9) Perform the zigzag decoding to the batches in the buffer. If there are zigzag decodable batches, add all the neighboring message symbols into the ripple and go to Step 6).

Step 10) Stop if all message symbols are successfully recovered. Otherwise, go to Step 4).

3.3 Degree Distribution of BZ Fountain Codes

3.3.1 Relation Between $\Psi(x)$ and $\Omega(x)$

In Algorithm 3.1, an encoder generates an unlimited sequence of batches using the batch degree distribution $\Psi(x)$. For a sampled degree d, $1 \leq d \leq d_m$, the size of the batch is equal to d, which means that d code symbols with the same neighbors are generated simultaneously. As a result, d code symbols with degree d are generated. Using this property, the code symbol degree distribution $\Omega(x)$ can be derived from the given batch degree distribution $\Psi(x)$.

Assume that an encoder generates B batches with N_{tot} code symbols. For the degree d, $1 \leq d \leq d_m$, there are d code symbols of degree d in each batch of size d. Therefore, we have $\Omega_d = Bd\Psi_d/N_{\text{tot}}$. For $d > d_m$, we have $B\Psi_d$ batches composed of single code symbol of degree d and thus $\Omega_d = B\Psi_d/N_{\text{tot}}$. Since $\Omega(1) = 1$, it can be written as

$$\sum_{d=1}^{k} \Omega_d = \frac{B}{N_{\text{tot}}} \left(1 + \sum_{d=2}^{d_m} (d-1)\Psi_d \right) = 1. \tag{3.11}$$

Here, θ is defined as

$$\theta = \frac{N_{\text{tot}}}{B}. \tag{3.12}$$

We can easily check that $\theta \geq 1$ and we have

$$\Omega_d = \begin{cases} \frac{d\Psi_d}{\theta}, & \text{if } 1 \leq d \leq d_m \\ \frac{\Psi_d}{\theta}, & \text{otherwise.} \end{cases} \tag{3.13}$$

3.3.2 Derivation of $\Omega(x)$ via Ripple Size Evolution

In [41], they analyzed the ripple size evolution for LT codes and obtained DRSD. In this chapter, we extend this approach to obtain a code symbol degree distribution for the proposed BZ fountain codes. Since code symbols in the same batch are designed to be zigzag decodable, code symbol degree distributions of the BZ fountain codes should be determined by the target channel parameter. Hence, the proposed BZ fountain codes are designed for the given symbol erasure rate while the conventional fountain codes are designed for the successful transmission oblivious of individual channel states. As a result, universality in channel parameters does not work in BZ fountain codes. However, we consider applications whose channel parameter does not vary much in our scenario. Assume that decoding process starts after receiving n code symbols for the purpose of ripple analysis, where $n = k\gamma_{succ}$. In each step of decoding process, one message symbol is selected from the ripple for decoding of the received code symbols in the buffer and then, it is removed from the ripple and becomes the recovered message symbol. Let L be the number of unrecovered message symbols, $L = k, k-1, \ldots, 0$. Let $Q^{(L)}$ denote the desired number of message symbols which are added into the ripple in the $(k-L)$th decoding step in order to keep the desired ripple size. For the proposed BZ fountain codes, $Q^{(L)}$ is composed of $Q_{\mathrm{LT}}^{(L)}$ and $Q_{\mathrm{B}}^{(L)}$ as

$$Q^{(L)} = Q_{\mathrm{LT}}^{(L)} + Q_{\mathrm{B}}^{(L)} \tag{3.14}$$

where $Q_{\mathrm{LT}}^{(L)}$ and $Q_{\mathrm{B}}^{(L)}$ are for the LT decoding and the batched zigzag decoding, respectively. Let $R^{(L)}$ be the ripple size at the end of the $(k-L)$th decoding step. Then, we have a simple relation as

$$R^{(L)} = R^{(L+1)} - 1 + Q^{(L)} \tag{3.15}$$

for $L < k$ and $R^{(L)} = Q^{(L)}$ for $L = k$.

First, we analyze the ripple size evolution of LT decoding based on a symbol-wise Tanner graph $\mathcal{G}_{\rm s} = (M, C, E)$, where M and C are sets of message and code symbol nodes, respectively, and E is a set of edges between M and C. We assume for simplicity that degree-d code symbols with the same neighbors occur only when they belong to the same batch for $2 \leq d \leq d_m$. In this setting, code symbols with the same neighbors can be assumed to be redundant in terms of the symbol-wise Tanner graph and thus the code symbols of each batch are represented by one code symbol node, called effective code symbol. Thus, when there are n code symbol nodes in $\mathcal{G}_{\rm s}$, the number of effective code symbol nodes is given as

$$|C'| = n \sum_{d=1}^{k} \frac{\Omega_d}{\mathbb{E}\left[W_{d,\epsilon}\right]} \tag{3.16}$$

where $\mathbb{E}[W_{d,\epsilon}]$ is the expected number of successfully received code symbols in a batch of degree d for the symbol erasure rate ϵ. Here, we have

$$\mathbb{E}\left[W_{d,\epsilon}\right] = \begin{cases} \sum_{w=1}^{d} w \frac{\binom{d}{w}(1-\epsilon)^w \epsilon^{d-w}}{1-\epsilon^d}, & \text{if } 2 \leq d \leq d_m \\ 1, & \text{otherwise.} \end{cases} \tag{3.17}$$

Example 3.8 *Suppose that batches of the proposed BZ fountain codes are given as* $\{\mathbf{C}_1, \ldots, \mathbf{C}_5\}$ *with* $\{\mathbf{m}_1, \ldots, \mathbf{m}_8\}$. *Let* $d_m = 3$. *The corresponding symbol-wise Tanner graph is shown in Fig. 3.3. In the Tanner graph, the white, the black, and the gray nodes denote erased, effective, and redundant code symbol nodes at the receiver, respectively. While the total number of successfully received code symbols is 6, the number of effective code symbols is 4.*

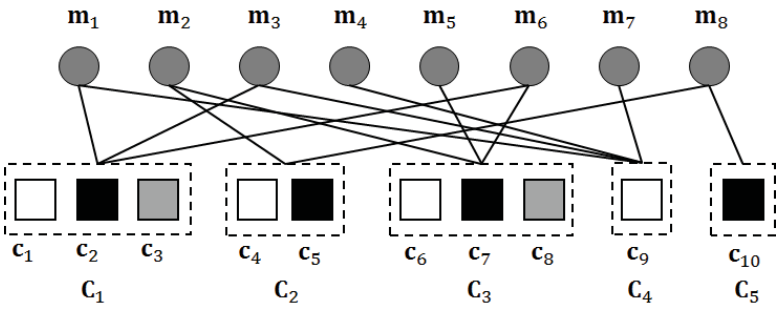

Figure 3.3: A symbol-wise Tanner graph of code symbols.

In LT codes, the total number of released code symbols depends on the code symbol degree distribution $\Omega(x)$ and the probability that a code symbol is released and added into the ripple. This probability is derived in the following lemma.

Lemma 3.9 ([41], Lemma 1) *Let $q(d, L, R)$ be the probability that a code symbol of degree d is released and added into the ripple, when L out of k message symbols remain unrecovered for the given ripple size R at the beginning of the each decoding step. Then we have*

$$q(d,L,R) = \begin{cases} 1, & \text{if } d=1, L=k, R=0 \\ \frac{\binom{L-(R-1)}{1}\binom{1}{1}}{\binom{k}{2}}, & \text{if } d=2, 1 \le R \le L \le k-1 \\ \frac{\binom{k-(L+1)}{d-2}\binom{1}{1}\binom{L-(R-1)}{1}}{\binom{k}{d}}, & \text{if } 3 \le d \le k, \\ & 1 \le R \le L \le k-d+1 \\ 0, & \text{otherwise.} \end{cases} \tag{3.18}$$

Using Lemma 3.9, the total number of the code symbols which are released and added into the ripple for the given ϵ, is

$$Q_{\mathrm{LT}}^{(L)} = \sum_{d=1}^{k} \frac{n\Omega_d}{\mathbb{E}\left[W_{d,\epsilon}\right]} \cdot q\left(d, L, R^{(L+1)}\right) \tag{3.19}$$

for $1 \leq L \leq k$, where we set that the ripple is initially empty, i.e., $R^{(k+1)} = 0$.

Secondly, we analyze the effects caused by batched zigzag decoding. Similar to Lemma 3.9, the probability that the message symbols in a batch are released and added into the ripple is shown in the following lemma.

Lemma 3.10 *Let $q_{\mathrm{B}}(d, L, R, t)$ be the probability that the message symbols in a batch of code symbols of degree d are released and added into the ripple when t code symbols are erased from the channel and L out of k message symbols remain unrecovered for the given ripple size R at the beginning of each decoding step. Then $q_{\mathrm{B}}(d, L, R, t)$ is given as*

$$q_{\mathrm{B}}(d, L, R, t) =$$

$$\begin{cases} 1, & \text{if } 2 \leq d \leq d_m, L = k, R = 0, t = 0 \\ \frac{\binom{k-(L+1)}{t-1}\binom{1}{1}\binom{L-(R-1)}{d-t}}{\binom{k}{d}}, & \text{if } 2 \leq d \leq d_m, \\ & 1 \leq t \leq \min\left(d-2, k-L\right), \\ & \quad L-(R-1) \geq d-t \\ 0, & \text{otherwise.} \end{cases} \tag{3.20}$$

Proof. As in the proof in [41, Lemma 1], we derive probabilities that $t-1$ neighbors belong to $k-(L+1)$ recovered message symbols, one neighbor is the currently

processing symbols at the $(k-L)$th decoding step, and the last $d-t$ neighbors belong to the $L-(R-1)$ unprocessed message symbols not in the ripple. $\square$

Note that $d-t$ received code symbols of degree d with t known message symbols compose a complete batch and $d-t$ new message symbols are released and added into the ripple. Thus, the number of message symbols which are released and added into the ripple with the batched zigzag decoding at the $(k-L)$th decoding step can be written as follows:

$$Q_{\mathrm{B}}^{(L)}(d) = \begin{cases} n\Omega_d \cdot \rho(d,0,\epsilon) q_{\mathrm{B}}(d,L,0,0), & \text{for } L=k \\ n\Omega_d \cdot \left(\sum_{t=1}^{d-2} \rho(d,t,\epsilon) q_{\mathrm{B}}\left(d,L,R^{(L+1)},t\right) \right), & \text{for } L<k \end{cases} \tag{3.21}$$

where $\rho(d,t,\epsilon)$ is the conditional probability that there are t erasures in the received batch of degree d for the given ϵ as

$$\rho(d,t,\epsilon) = \frac{\binom{d}{t}(1-\epsilon)^{d-t}\epsilon^t}{1-\epsilon^d}. \tag{3.22}$$

Consequently, the overall expected number of message symbols that are added into the ripple at the $(k-L)$ decoding step is derived as

$$Q^{(L)} = \begin{cases} n\Omega_1 + \sum_{d=2}^{d_m} n\Omega_d \cdot \rho(d,0,\epsilon), & \text{for } L=k \\ \sum_{d=1}^{k} n\Omega_d \left[\frac{q\left(d,L,R^{(L)}\right)}{\mathbb{E}\left[W_{d,\epsilon}\right]} \right. & \\ \qquad \left. + \sum_{t=1}^{d-2} \rho(d,t,\epsilon) q_{\mathrm{B}}\left(d,L,R^{(L+1)},t\right) \right], & \text{for } L<k. \end{cases} \tag{3.23}$$

The desired ripple size evolution of the BZ fountain codes is set as

$$R^{(L)} = \begin{cases} c_1 L^{1/c_2}, & \text{if } c_1 L^{1/c_2} \leq L \\ L, & \text{otherwise} \end{cases} \tag{3.24}$$

where $c_1 > 0$ and $c_2 > 1$ are suitably chosen design parameters.

Plugging (3.24) into (3.15) gives the number of message symbols required to be released and added into the ripple. To obtain the code symbol degree distribution $\Omega(x)$ satisfying $Q^{(L)}$ in (3.23), we use the nonnegative least square approximation as used in [41]. Here, we have two more design parameters, i.e., d_m and ϵ. Thus, we fix ϵ and find a suitable 3-tuple vector (d_m, c_1, c_2), which results in the minimum coding overhead. Finally, $\Psi(x)$ can be easily obtained from $\Omega(x)$ by using (3.13).

Note that there is a difference between (3.24) and that in [41] for some values of c_2. The condition of design parameter c_2 is changed from $c_2 > 2$ to $c_2 > 1$. While LT codes require $d - 2$ already recovered message symbols out of d neighbors to release a degree-d code symbol, the BZ fountain codes need $t - 1$ recovered message symbols. Since $t - 1 < d - 2$, the addition of the recovered messages into the ripple for the proposed codes is much faster than that of LT codes at the early decoding process. Hence, the initial ripple size of the BZ fountain codes can be larger than that of LT codes when the symbol erasure rate is low. In this point of view, we choose the condition of the design parameter $c_2 > 1$ for the desired ripple size evolution.

Selecting the proper portion of degree-one code symbols is very important because it determines the initial ripple size of the conventional LT codes. If the probability of degree-one code symbols is too small, the ripple size may become zero before all the message symbols are recovered. In other words, it is vulnerable to decoding failure. On

the contrary, high probability of degree-one code symbols results in weak connection, which degrades recovery ratio at the late decoding steps. However, for the proposed codes, the initial ripple size is determined by not only degree-one code symbols but also degree-d code symbols, $2 \leq d \leq d_m$. Using this property, we can reduce the portion of degree-one code symbols and have additional room for code symbols with high degrees.

Example 3.11 *Degree distributions of LT codes and the BZ fountain code are obtained for* $k = 32$. *Using the method in [41], the optimized design parameters of DRSD for LT codes are obtained as* $c_1 = 1.2$ *and* $c_2 = 2.4$ *and the corresponding degree distribution as* $\Omega^{(\mathrm{DRSD})}(x) = 0.1206x + 0.4190x^2 + 0.1095x^3 + 0.1464x^4 + 0.0635x^6 + 0.0182x^7 + 0.1228x^{15}$. *On the other hand, the degree distributions for the proposed BZ fountain codes are obtained with* $d_m = 4$ *and various target symbol erasure rates are considered, i.e.,* $\epsilon \in \{0.1, 0.2, 0.5, 0.9\}$. *Numerically optimized* (c_1, c_2) *for the given* ϵ *is* $(1.0, 1.1)$, $(1.0, 1.2)$, $(1.0, 3.5)$, *and* $(1.0, 3.8)$, *respectively and the obtained degree distributions* $\Omega^{(\epsilon)}(x)$ *are shown in Table 3.1. We can see that* Ω_1 *of the proposed BZ fountain codes is much smaller than that of DRSD, which is almost negligible. The corresponding ripple size evolutions are shown in Fig. 3.4. We can check that the initial ripple size increases as* ϵ *becomes smaller.*

Table 3.1: Degree distributions of BZ fountain codes with $d_m = 4$ and $k = 32$.

ϵ	$\Omega^{(\epsilon)}(x)$
0.1	$0.0001x + 0.5575x^2 + 0.4424x^5$
0.2	$0.0002x + 0.5577x^2 + 0.3939x^5 + 0.0482x^{16}$
0.5	$0.0003x + 0.2625x^2 + 0.5340x^3 + 0.0740x^9 + 0.0493x^{10} + 0.0341x^{23} + 0.0458x^{24}$
0.9	$0.0409x + 0.4606x^2 + 0.1445x^3 + 0.0858x^4 + 0.0985x^5 + 0.0598x^{12} + 0.0421x^{13} + 0.0408x^{26} + 0.0270x^{27}$

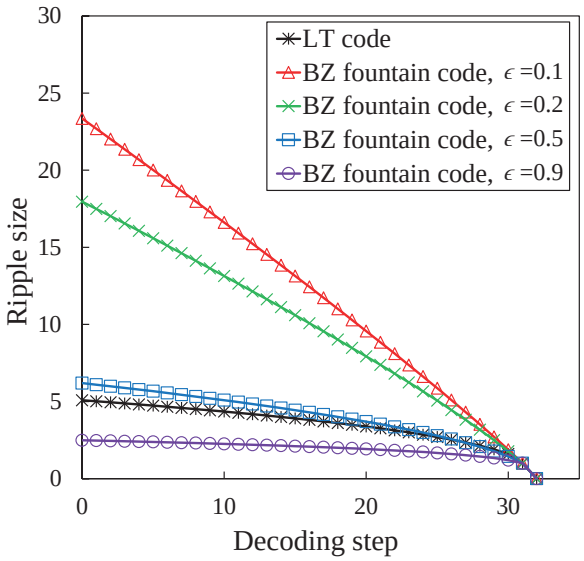

Figure 3.4: Desired ripple size evolutions for the conventional LT codes and the proposed BZ fountain codes.

3.4 Two-Phase Batched Zigzag Fountain Codes with Additional Memory

In Section 3.2, we proposed the BZ fountain codes. With slightly additional bit-level overhead, decoding process of the proposed BZ fountain codes contains both the LT decoding and the BZ decoding, which gives improved ISRR. However, the recovered message symbol ratio slowly increases around $\gamma = 1$ and it leads to performance degradation with respect to the coding overhead. This phenomenon is due to the fact that the BZ fountain codes consistently generate redundant code symbols for the decoder. In other words, the encoder is apt to continuously generate inefficient batches with already recovered neighbors (message symbols) especially at the late decoding steps, where producing batches of d code symbols with the same neighbors repeatedly for $2 \leq d \leq d_m$ may be wasteful.

Similar to the drawback of the BZ fountain codes, the conventional LT codes also suffer from high chance of receiving redundant code symbols at the late decoding steps, especially for short k. In [65], two main reasons are identified for this phenomenon. Firstly, some of particular message symbols may not have been selected as neighbors of code symbols until γ becomes large. Secondly, even though some message symbols are involved, the message symbols can be included only in high degree code symbols. Hence, some message symbols remain unrecovered and the decoder has to receive more code symbols. To resolve this drawback, LT codes with added memory (LTAM) scheme was proposed in [65] such that the encoder selects distinct d message symbols, which previously have not been selected for encoding. Also in [66], the authors used instantaneous degrees of the message symbols for encoding. Similarly, we

apply these approaches to the BZ fountain codes.

3.4.1 Code Construction

Now we propose two-phase batched zigzag fountain codes to reduce the coding overhead of the BZ fountain codes. The proposed TBZ fountain codes separate the encoding procedure into two phases as described in Algorithm 3.3, that is, the first encoding phase is from Step 1) to Step 5) and the second encoding phase is from Step 6) to Step 10). Note that the encoding phase is determined by the additional memory which stores indices of the selected message symbols for the previously generated code symbols. When all message symbols are selected at least once as neighbors of batches with size larger than one, the encoder switches the encoding phase from the first phase to the second phase. In Algorithm 3, δ_{R} is set to $d_m - 1$. Moreover, the TBZ fountain codes use Algorithm 3.2 for their decoding procedure. Since encoding procedure of the first phase of TBZ fountain codes is similar to that of the BZ fountain codes, we use code symbol and batch degree distributions obtained in Subsection 3.3.2.

One of key features of the TBZ fountain codes is that the encoder has additional memory $\mathcal{I}'$. Using $\mathcal{I}'$, the message symbols which have not been selected for the previously generated code symbols are preferentially selected at Step 3). This approach ensures that all message symbols are involved in the batches of size $T > 1$ at least once. Since neighbors of all batches of size $T > 1$ are new to the receiver at the first phase except the last batch, there are no redundant code symbols. Further, $\mathcal{I}'$ leads to a transition of code symbol degree distribution. At the first phase, the encoder uses $\Psi(x)$ for the degree of batches while the degree distribution of the generated code symbols is $\Omega(x)$, which is derived from $\Psi(x)$. However, we keep the size of all batches as one

Algorithm 3.3: Encoding of TBZ Fountain Codes

Input: $k, \{m_1(z), \ldots, m_k(z)\}$, $\Psi(x)$, d_m, δ_{R}, $\mathcal{S} = \{\mathcal{S}_2, \ldots, \mathcal{S}_{d_m}\}$

Initialization: $\mathcal{I} \leftarrow \{1, \ldots, k\}$, $\mathcal{I}' \leftarrow \emptyset$

(First phase)

Step 1) Sample a degree d from $\Psi(x)$.

Step 2) If $1 \leq d \leq d_m$, $T \leftarrow d$. Otherwise, $T \leftarrow 1$.

Step 3) If $T > 1$, $r \leftarrow |\mathcal{I} \setminus \mathcal{I}'|$.

If $r \geq d$, select distinct d indices of message symbols $\mathcal{I}_{sel} = \{I_1, \ldots, I_d\}$ from $\mathcal{I} \setminus \mathcal{I}'$ uniformly at random.

Otherwise, select distinct r indices of message symbols $\mathcal{I}_{sel}^{(1)} = \{I_1, \ldots, I_r\} \leftarrow \mathcal{I} \setminus \mathcal{I}'$ and select the remaining $d - r$ indices $\mathcal{I}_{sel}^{(2)} = \{I_{r+1}, \ldots, I_d\}$ from $\mathcal{I}'$ uniformly at random.

$\mathcal{I}_{sel} \leftarrow \mathcal{I}_{sel}^{(1)} \cup \mathcal{I}_{sel}^{(2)}$.

$\mathcal{I}' \leftarrow \mathcal{I}' \cup \mathcal{I}_{sel}$ and $\mathbf{S}_{T \times d} = [s_{j,w}] \leftarrow \mathcal{S}_d$.

Otherwise, select distinct d indices of message symbols from $\mathcal{I}$ uniformly at random.

$\mathcal{I}_{sel} \leftarrow \{I_1, \ldots, I_d\}$ and $\mathbf{S}_{T \times d} = [s_{j,w}] \leftarrow \mathbf{s}_{(\mathrm{R})}^{d}$.

Step 4) $\mathbf{C}(z) \leftarrow \mathbf{G}_{T \times d}(z) \begin{bmatrix} m_{I_1}(z) & \cdots & m_{I_d}(z) \end{bmatrix}^{\mathsf{T}}$, where entry of $\mathbf{G}_{T \times d}(z)$ is $g_{j,w} = z^{s_{j,w}}$ for $j = 1, \ldots, T$ and $w = 1, \ldots, d$.

Step 5) Stop if every receiver successfully recovers the k message symbols.

Otherwise, if $|\mathcal{I}'| < k$, go to Step 1). If $|\mathcal{I}'| = k$, go to Step 6).

Algorithm 3: Encoding of TBZ Fountain Codes (continued)

(Second phase)

Step 6) Sample a degree d from $\Psi(x)$.

Step 7) $T \leftarrow 1$.

Step 8) Select distinct d indices of message symbols from $\mathcal{I}$ uniformly at random.

$\mathcal{I}_{sel} \leftarrow \{I_1, \ldots, I_d\}$ and $\mathbf{S}_{T\times d} = [s_{1,w}] \leftarrow \mathbf{s}^d_{(\mathrm{R})}$.

Step 9) $\mathbf{C}(z) \leftarrow \mathbf{G}_{T\times d}(z)\begin{bmatrix} m_{I_1}(z) & \cdots & m_{I_d}(z) \end{bmatrix}^{\mathsf{T}}$, where entry of $\mathbf{G}_{T\times d}(z)$ is $g_{j,w} = z^{s_{j,w}}$ for $j = 1, \ldots, T$ and $w = 1, \ldots, d$.

Step 10) Stop if every receiver successfully recovers the k message symbols.

Otherwise, go to Step 6).

with the random degrees at the second phase. Thus, generation of code symbols at the second phase simply follows $\Psi(x)$. In other words, code symbol degree distribution changes from $\Omega(x)$ to $\Psi(x)$.

Another key feature of the proposed TBZ fountain codes is applying the shift vector $\mathbf{s}_{(\mathrm{R})}^{d}$ to batches of size one, which improves performance of the coding overhead as investigated in [45]. It is known that code symbols with $\mathbf{s}_{(\mathrm{R})}^{d}$ have higher chance to be released than those with $\mathbf{s}_{(0)}^{d}$. Since neighbors of batches of size one are randomly selected without considering the selection history, they can help recovering incompletely received batches with $T > 1$. Thus, applying the random shift vector accelerates the message recovering process.

Proposition 3.12 *Consider TBZ fountain codes with code symbol degree distribution $\Omega(x)$ when symbol erasure rate is ϵ. Then the ratio of the number of received code symbols generated at the first phase to the number of message symbols is given as*

$$\gamma_1 = \frac{1-\epsilon}{\sum_{d=2}^{d_m} \Omega_d}. \tag{3.25}$$

Proof. Suppose that B batches with N_{tot} code symbols are generated at the first phase. We compute the total number of generated code symbols in the batches of size $T > 1$. In Algorithm 3, we maintain the memory for the selected message symbol history. Therefore, unselected message symbols are picked first. It implies that total number of code symbols in the batches of size $T > 1$ is equal to $k + d_m - r \approx k$, that is,

$$B \sum_{d=2}^{d_m} d\Psi_d = k. \tag{3.26}$$

Since a decoder receives $N_{\text{tot}}(1-\epsilon) = k\gamma_1$ code symbols, we have (3.25) from (3.12) and (3.13). □

Proposition 3.13 *If the code degree distribution $\Omega(x)$ satisfies*

$$\sum_{d=2}^{d_m} \frac{d-1}{d}\Omega_d \leq \frac{1}{2} \tag{3.27}$$

then $\Psi_d \leq \Omega_d$ for $2 \leq d \leq d_m$ and $\Psi_d \geq \Omega_d$ for $d = 1$ or $d > d_m$.

Proof. If $\Omega(x)$ satisfies (3.27), then we can write as $1-1/\theta \leq 1/2$, which becomes $\theta \leq 2$ from (3.12). From (3.13), we have $\Psi_d \leq \Omega_d$ for $2 \leq d \leq d_m$. Also, we can easily check $d = 1$ and $d > d_m$ since $\theta \geq 1$. □

The above condition on the code symbol degree distribution $\Omega(x)$ leads to increasing the fraction of high degree code symbols at the second phase, which ensures stronger connection between message symbols and code symbols. Although the fraction of degree-one code symbols is also increased, it is less affected if we keep Ω_1 small enough. In [41], it is shown that low and high degree code symbols are more likely to be released at the early and late decoding processes, respectively. Therefore, changing from the first phase to the second phase at the late decoding steps in the proposed TBZ fountain codes is effective for decoding performance improvement.

Example 3.14 *Consider a code symbol degree distribution $\Omega(x)$ for the TBZ fountain codes for $d_m = 4$ and $k = 128$, which is given as $\Omega(x) = 0.0002x + 0.5476x^2 + 0.2897x^5 + 0.0762x^{19} + 0.0863x^{20}$. Then the corresponding batch degree distribution is $\Psi(x) = 0.0004x + 0.3770x^2 + 0.3989x^5 + 0.1049x^{19} + 0.1188x^{20}$. From Proposition 3.13, it is shown that $\Psi_d \geq \Omega_d$ except for $d = 2$. With Proposition 3.12, the*

decoder receives the code symbols generated at the first phase until it collects $k\gamma_1$ code symbols, where $\gamma_1 = 0.91$ for the target erasure rate $\epsilon = 0.5$. From numerical analysis, we have $\tilde{\gamma}_1 = 0.90$ for the receiver which has code symbols generated in both the first and second phases when the buffer is unlimited, i.e., $\beta = \infty$. In fact, the average coding overhead obtained from the simulation is $\gamma_{succ} = 1.08$. For the high symbol erasure rate, the number of released code symbols by the batched zigzag decoding at the first phase decreases. Hence, the decoder also uses code symbols generated in the second phase. Indeed, they are the same as those of ZD fountain codes using the code symbol degree distribution which gives higher average degree. They compensate the coding overhead performance degradation. Therefore, changing from the first phase to the second phase at late decoding process is effective for the decoding performance improvement.

Remark 3.15 *As the portion of high degree code symbols increases due to switching of encoding phase in TBZ fountain codes, the encoding and decoding complexities also increase.*

Remark 3.16 *While standard LT decoding and inactivation decoding can be easily parallelized, BZ and TBZ fountain codes do not support parallel decoding.*

3.4.2 Bit-Level Overhead

We derive the average length of the code symbols for the proposed TBZ fountain codes. Since there are two encoding phases in the proposed codes, we analyze them separately.

At first, we study on the effect of adopting the random shift vector in TBZ fountain codes. For a code symbol $\mathbf{c}$ with degree d and the number of additional bits r, we have the shift vector $\mathbf{s}^d = (s_1, \ldots, s_d)$, where $0 \leq s_j \leq r$ for $j = 1, \ldots, d$. Let a_u be the number of $s_j = u$ in $\mathbf{s}^d$, where $\sum_{u=0}^{r} a_u = d$. Then, a vector $\mathbf{a} = (a_0, \ldots, a_r)$ denotes the distribution of shift values in $\mathbf{s}^d$. We define a set of $(r+1)$-tuple vectors as

$$\mathcal{A}_{r,d} = \left\{ \mathbf{a} \;\middle|\; \sum_{u=0}^{r} a_u = d, 1 \leq a_0, a_r \right\} \tag{3.28}$$

where clearly, $0 \leq a_u \leq d-2$ for $1 \leq u \leq r-1$. Let $g(r,d)$ be the number of shift vectors with r for the given d. Then $g(r,d)$ is given as

$$g(r,d) = \begin{cases} 1, & \text{for all } d \text{ and } r = 0 \\ 0, & \text{for } d = 1 \text{ and } 1 \leq r \leq \delta_{\mathrm{R}} \\ \sum_{\mathbf{a} \in \mathcal{A}_{r,d}} \frac{d!}{\prod_{u=0}^{r} a_u!}, & \text{otherwise.} \end{cases} \tag{3.29}$$

Theorem 3.17 *Consider the TBZ fountain codes with $\Omega(x), l$, d_m, $\delta_{\mathrm{R}} = d_m - 1$, and the set of predetermined shift matrices $\mathcal{S} = \left\{ \mathbf{S}_{2\times 2}^{(\mathrm{EV})}, \ldots, \mathbf{S}_{d_m \times d_m}^{(\mathrm{EV})} \right\}$. Then the average length of code symbols at the first phase is derived as*

$$\begin{aligned} \mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ},1)} \right] =& \mathbb{E}\left[L_{\Omega}^{(\mathrm{BZ})} \right] + \mathbb{E}\left[L_{\Omega}^{(\mathrm{ZD})} \right] - l \\ & - \sum_{r=1}^{d_m - 1} \left[r f(r, d_m) \right] \end{aligned} \tag{3.30}$$

where

$$f(r,k) = \sum_{d=1}^{k} \frac{(\delta_{\mathrm{R}} + 1 - r) g(r,d)}{(\delta_{\mathrm{R}} + 1)^d} \Omega_d. \tag{3.31}$$

Proof. First, let R and D denote the random variable of additional bits for code symbol and degree of code symbol for TBZ fountain codes, respectively. Then we divide the number of additional bits into two parts as

$$\mathbb{E}[R] = \sum_{d=2}^{d_m} \mathbb{E}[R|D=d]\Omega_d + \underbrace{\sum_{d=d_m+1}^{k} \mathbb{E}[R|D=d]\Omega_d}_{\Delta}. \tag{3.32}$$

The first term leads to $\mathbb{E}\left[L_{\Omega}^{(\mathrm{BZ})}\right] - l$. Again the second term can be split into

$$\Delta = \mathbb{E}\left[L_{\Omega}^{(\mathrm{ZD})}\right] - l - \sum_{d=1}^{d_m} \Omega_d \sum_{r=1}^{d_m-1} r \Pr[R=r|D=d]. \tag{3.33}$$

The conditional probability that for the given d, the number of additional bits is r is given as

$$\Pr[R=r|D=d] = \frac{(\delta_{\mathrm{R}}+1-r)g(r,d)}{(\delta_{\mathrm{R}}+1)^d}. \tag{3.34}$$

Thus, (3.33) leads to the last three terms in (3.17). □

For the second phase, since the encoding is the same as that of the ZD fountain codes, we have $\mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ},2)}\right] = \mathbb{E}\left[L_{\Psi}^{(\mathrm{ZD})}\right]$.

Theorem 3.18 *Consider the TBZ fountain codes with $\Omega(x), l$, d_m, $\delta_{\mathrm{R}} = d_m - 1$, and the set of predetermined shift matrices $\mathcal{S} = \left\{\mathbf{S}_{2\times 2}^{(\mathrm{EV})}, \ldots, \mathbf{S}_{d_m\times d_m}^{(\mathrm{EV})}\right\}$. Then the average length of code symbols is given as*

$$\begin{aligned}\mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ})}\right] =& l + \kappa\left(\mathbb{E}\left[R_{\Omega}^{(\mathrm{TBZ},1)}\right] - l\right) \\ &+ (1-\kappa)\left(\mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ},2)}\right] - l\right)\end{aligned} \tag{3.35}$$

for $\gamma_1 < \gamma_{succ}$, where $\kappa = \gamma_1/\gamma_{succ}$ and

$$\mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ})}\right] = \mathbb{E}\left[L_{\Omega}^{(\mathrm{TBZ},1)}\right] \tag{3.36}$$

for $\gamma_1 \geq \gamma_{succ}$.

Proof. Using Theorem 3.17 and (3.25), it is not difficult to prove it. □

3.5 Numerical Analysis

We show that the proposed TBZ fountain codes outperform LT codes and ZD fountain codes [45] with a lower buffer occupation at the receiver when symbol erasure rate is low. Since it is hard to analytically compute the coding overhead, we employ Monte Carlo simulation. We verify that the proposed TBZ fountain codes have low coding overhead and high ISRR.

We set $l = 50$ for all simulations. The DRSD is used for both LT codes and ZD fountain codes. On the other hand, the degree distributions obtained from the proposed method in Subsection 3.3.2 are applied to BZ and TBZ fountain codes.

Due to additional bits at the encoding process, the proposed codes have the bit-level overhead larger than one, i.e., $\eta > 1$. Hence, we adjust the received code symbol ratio as $\gamma' = \gamma\eta$ for fair comparison. Let b denote the ratio of the number of stored code symbols in the buffer to k. Similarly, b is adjusted as $b' = b\eta$ for all simulations. Despite these adjustments, we still want to use notations γ and b together with η instead of γ' and b' for the rest of the chapter.

For improved performance of the proposed BZ and TBZ fountain codes, we obtain degree distributions for different $d_m \in \{2, 3, 4, 5\}$ when $k = 128$, $\epsilon = 0.2$, and $(c_1, c_2) = (1.2, 1.1)$. We use the set of extended Vandermonde shift matrices for the proposed BZ and TBZ fountain codes. Clearly, d_m and the obtained degree distribution

determine the bit-level overhead. Fig. 3.5 shows that as d_m increases, the coding overhead decreases and the bit-level overhead increases. We try to keep the upper bound of bit-level overhead small. Thus, we fix $d_m = 4$ for all simulations. It is reasonable because the coding overhead is not considerably improved for $d_m > 4$. For fair comparison, we also apply $\delta_{\mathrm{R}} = d_m - 1$ to ZD fountain codes.

Assume that the buffer size is limited to $k\beta$. When buffer is full, LT codes and ZD fountain codes randomly discard a code symbol and receive a new code symbol. For BZ and TBZ fountain codes, we can also discard a code symbol randomly, called *random discarding approach*. Furthermore, we can apply a strategy for the proposed fountain codes so that a code symbol in the batch which has the largest number of erased code symbols is randomly discarded, called *batch discarding approach*, since its batch is most unlikely zigzag decodable.

In Fig. 3.6–Fig. 3.9, μ and b of LT codes, ZD fountain codes, BZ fountain codes, and TBZ fountain codes are compared with respect to γ for $k = 32$ and $\epsilon = 0.2$. Here, BZ and TBZ fountain codes (p) and (r) represent BZ and TBZ fountain codes with the proposed batch discarding and random discarding approaches, respectively when the buffer size is limited. Design parameters for the proposed BZ and TBZ fountain codes are set to $(d_m, \epsilon, c_1, c_2) = (4, 0.2, 1.0, 1.2)$. In Fig. 3.6, both BZ and TBZ fountain codes outperform LT codes and ZD codes for $0 < \gamma < 0.97$ and TBZ fountain codes boost up the speed of message symbol recovery for $\gamma > 0.89$ due to the random shift vectors. Although ZD fountain codes show good coding overhead for $\gamma > 1$, it has low ISRR for $\gamma < 0.8$ as the conventional LT codes. In Fig. 3.7, we can see that the peak value of b for the proposed codes decreases from 0.59 to 0.44 for the improved ISRR. Next, we simulate the same codes when the size of the buffer is limited, i.e, $\beta = 0.6$.

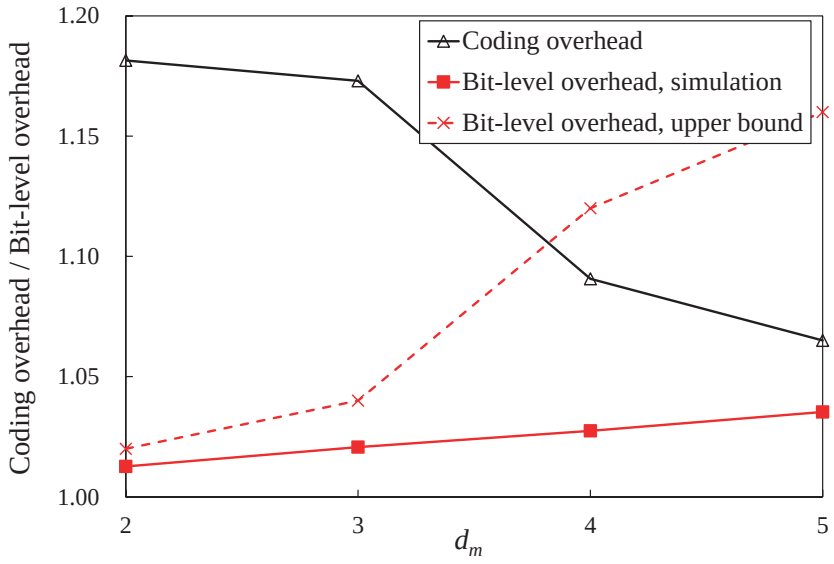

Figure 3.5: Coding overhead and bit-level overhead of TBZ codes when c_1 and c_2 are fixed.

The corresponding results are shown in Fig. 3.8 and Fig. 3.9. While the recovery ratios of LT codes and ZD fountain codes degrade due to discarding code symbols which cannot be stored in the buffer, those of the proposed codes are less affected. We can see how the proposed batch discarding approach gives additional performance gain.

For $\epsilon = 0.2$, coding overheads for various k and unlimited/limited buffers are shown in Fig. 3.10. With optimized design parameters, TBZ fountain codes have the smallest coding overhead for unlimited buffer size cases. We verify that the proposed TBZ fountain codes outperform LT codes and ZD fountain codes when storing at most $0.6k$ code symbols at the decoder. Even though ZD fountain codes also have similar performance without any constraint, the performance of ZD fountain codes degrades down to that of LT codes when the buffer size is strictly limited, while TBZ fountain

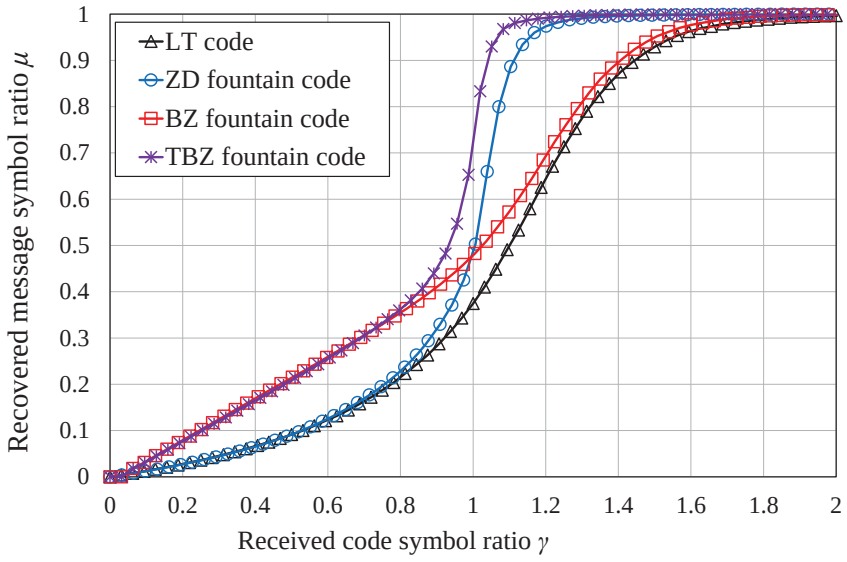

Figure 3.6: Recovered message symbol ratio with $\beta = \infty$.

codes have only small degradation of coding overhead. In fact, TBZ fountain code outperforms short ZD fountain code with $k' = 0.6k$ and $\beta' = 1$ when k is small, i.e., $k \leq 256$. Moreover, in order to use short length of message symbols, the message symbols should be divided into smaller subpackets, each of which is transmitted sequentially. Since we focus on broadcasting scenarios with many receivers, increasing the number of subpackets can cause overall latency problem.

The coding overhead of the TBZ fountain codes depends on the channel parameter. We fix the target symbol erasure rate to 0.2 but simulate for various ϵ at $k = 64$. Simulation results in Fig. 3.11 show that for the unlimited buffer size, the coding overhead of TBZ fountain code does not change much for $0.01 \leq \epsilon \leq 0.6$, which is similar to those of LT codes and ZD fountain codes. For the limited cases, around the target symbol erasure rate 0.2, i.e., $0.01 \leq \epsilon \leq 0.3$, the coding overhead of the proposed

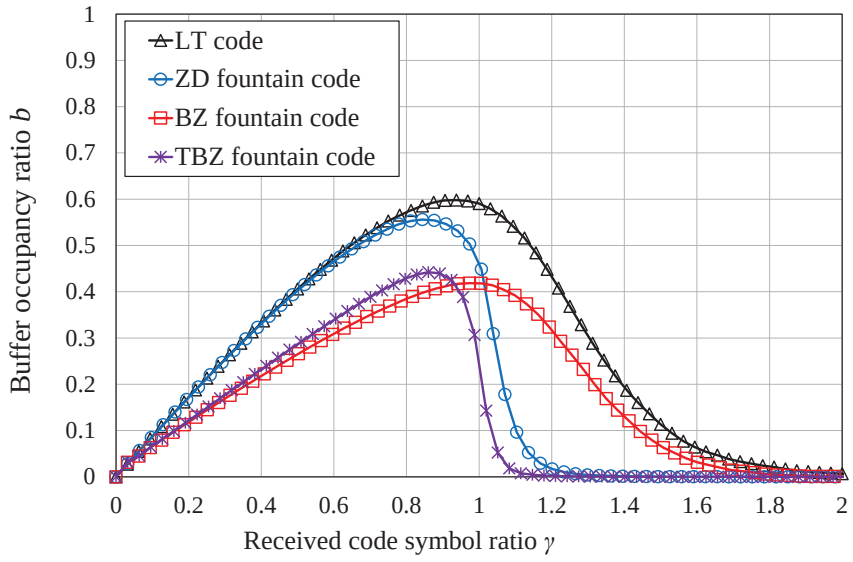

Figure 3.7: Buffer occupancy ratio with $\beta = \infty$.

code is almost constant and is better than those of LT codes and ZD fountain codes. However, when the symbol erasure rate becomes higher, the decoder rarely receives complete batches, which prevents batched zigzag decoding. Moreover, the fraction of degree-one code symbol is very small in the obtained code symbol degree distribution. Hence, the size of ripple frequently becomes zero and coding overhead performance degrades severely. Nonetheless, the simulation results show the robustness of the TBZ fountain codes around the target symbol erasure rate, which means that the proposed codes are useful in applications whose channel parameter does not vary much.

Here, we compare the performance of the TBZ fountain codes over channels with memory. The Gilbert-Elliott (GE) channel model has two states, a good and a bad states. Let ϵ_{g} and ϵ_{b} denote the symbol erasure rates of the good and bad states, respectively. The transition probability p_{s_1,s_2} represents the transition probability from

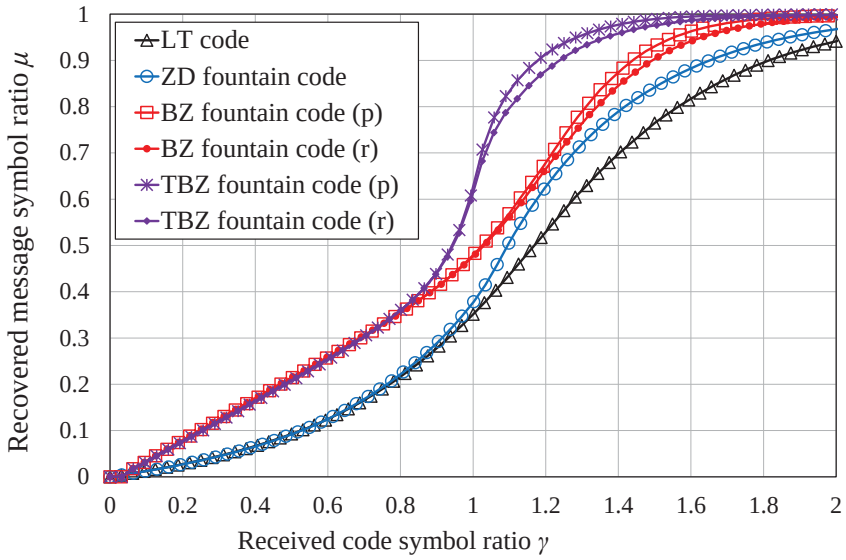

Figure 3.8: Recovered message symbol ratio with $\beta = 0.6$.

state s_1 to state s_2, where $s_i \in \{\mathrm{g}, \mathrm{b}\}$ for $i = 1, 2$. We set the parameters of the GE channel as $p_{\mathrm{g,g}} = p_{\mathrm{b,b}} = 0.9$ and $p_{\mathrm{g,b}} = p_{\mathrm{b,g}} = 0.1$. When average symbol erasure rate is fixed to 0.2 and the symbol erasure rate of good state is $\epsilon_{\mathrm{g}} = 0.01$ or $\epsilon_{\mathrm{g}} = 0.1$, the coding overheads of TBZ fountain codes over memoryless BEC and GE channel are shown in Table 3.2, where TBZ fountain codes are designed with the target erasure rate 0.2 for $k = 32$. When the channel is in bad state, most of batches are received incompletely. However, batches can be received without any erasure with high probability when a receiver stays in good channel state. Using batched zigzag decoding, the corresponding message symbols are easily recovered. In this sense, TBZ fountain codes can be applied to GE channel. However, since TBZ fountain codes are not universal, the performance of the proposed codes will be bad when channel is stuck in a bad state for a long time at the first phase. In other words, if some receivers start to

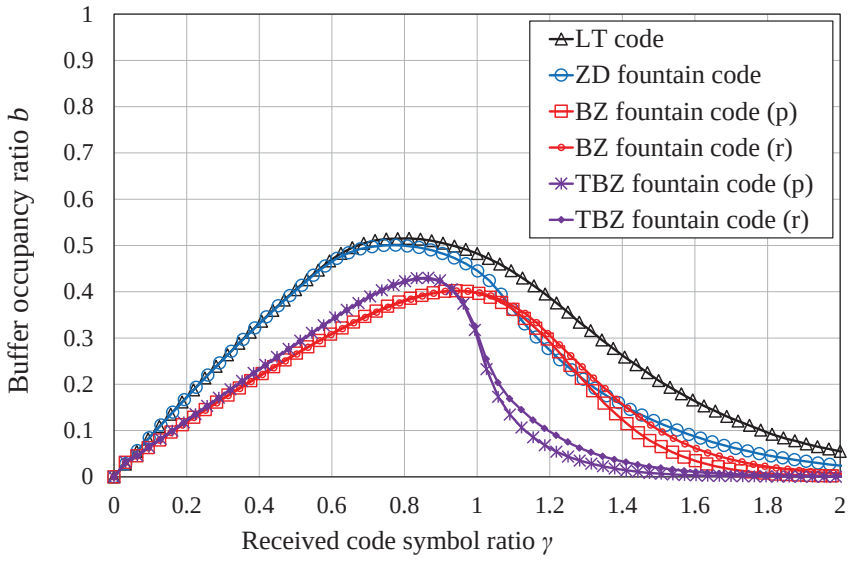

Figure 3.9: Buffer occupancy ratio with $\beta = 0.6$.

receive the code symbols at a later time, the performance of TBZ fountain codes will be degraded, which is not the case of the conventional fountain codes [17].

Now, we verify improvement on the coding overhead for TBZ fountain codes to LT codes with the ISRR distribution and the RCSS algorithm in [59]. Fig. 3.12 shows the recovered message symbol ratio μ with respect to the received code symbol ratio γ for $k = 100$. ISRR distributions with 3 different weighted vectors show almost optimal performance. However, due to the lack of high degrees in these codes, every distribution suffers from poor coding overhead. We can see that the intermediate recovery performance of TBZ fountain codes is between LT codes with the ISRR distribution and those with the general DRSD distribution without the RCSS algorithm. Consequently, while LT codes using the ISRR distributions and the RCSS algorithm require storing all neighbors of code symbols at the encoder and computing transmission priority,

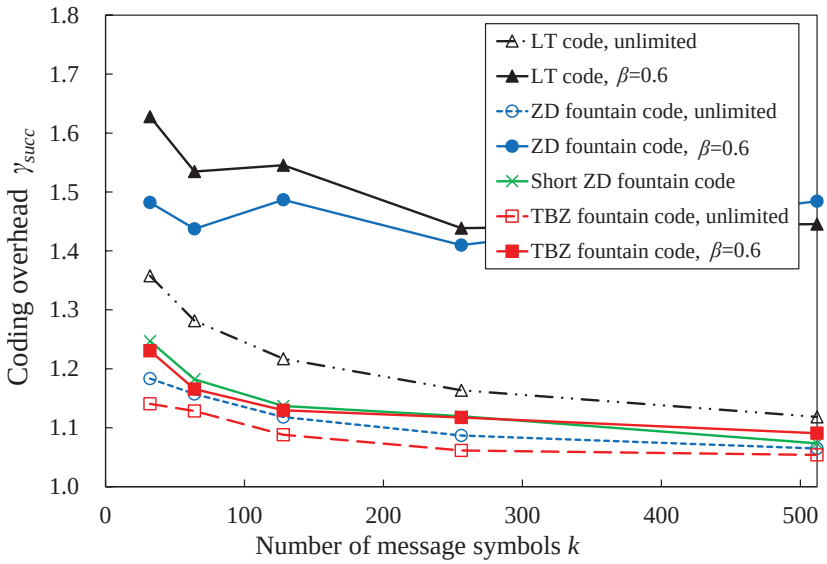

Figure 3.10: Coding overheads of LT codes, ZD fountain codes, and TBZ fountain codes for $k \in \{32, 64, 128, 256, 512\}$ when the available buffer size is unlimited/limited with $\beta = 0.6$.

TBZ fountain codes have improved message symbol recovery ratio without sacrificing coding overhead. In Fig. 3.13, similar simulations are performed for various β. Since LT codes with the ISRR distribution have almost optimal recovery, they use very small amount of buffer to store unreleased code symbols, which show the robustness against β. However, their coding overheads are much higher than those of the TBZ fountain codes.

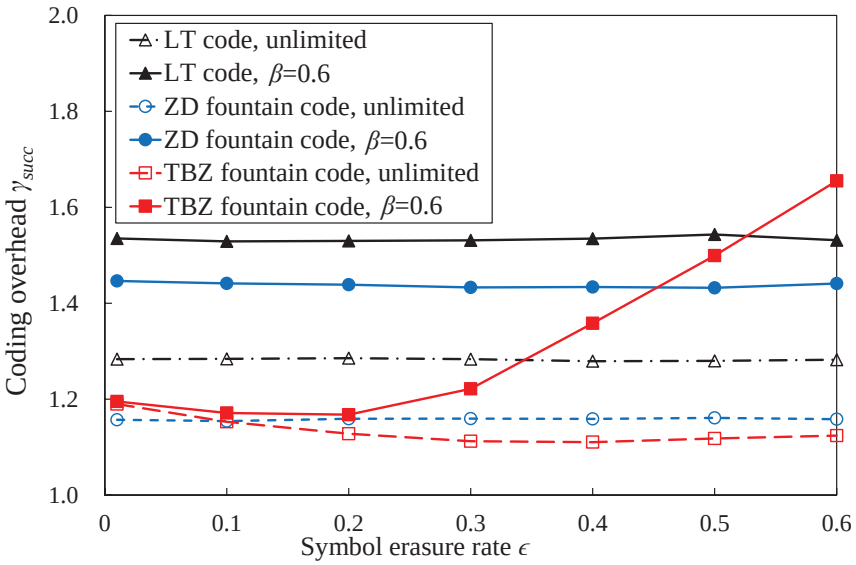

Figure 3.11: Coding overheads of LT codes, ZD fountain codes, and TBZ fountain codes for various ϵ when the available buffer size is unlimited/limited with $\beta = 0.6$, where the target symbol erasure rate is set to 0.2.

Table 3.2: Coding overheads of TBZ fountain codes over memoryless BEC and GE channel when symbol erasure rate is 0.2.

Memoryless BEC		GE channel			
unlimited	$\beta = 0.6$	$\epsilon_{\mathrm{g}} = 0.01, \epsilon_{\mathrm{b}} = 0.39$		$\epsilon_{\mathrm{g}} = 0.1, \epsilon_{\mathrm{b}} = 0.3$	
		unlimited	$\beta = 0.6$	unlimited	$\beta = 0.6$
1.1258	1.2403	1.1487	1.2357	1.1262	1.2414

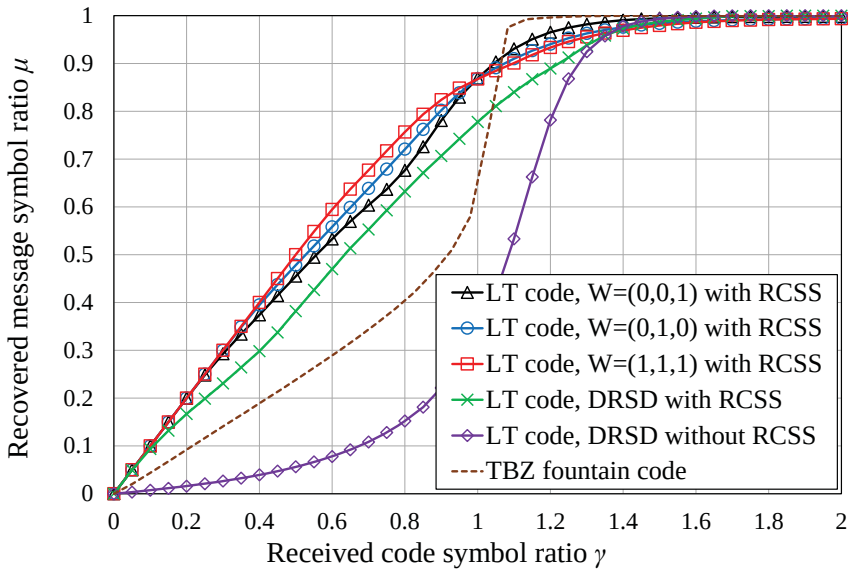

Figure 3.12: Comparison of the proposed TBZ fountain codes and LT codes using the ISSR distributions and the DRSD with/without the RCSS algorithm at the encoder with respect to μ for $k = 100$ and the unlimited buffer size.

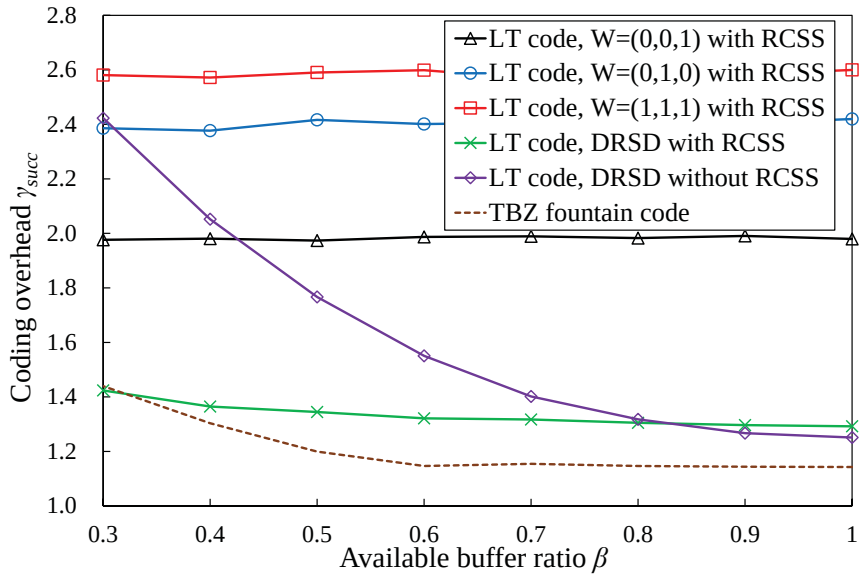

Figure 3.13: Comparison of the proposed TBZ fountain codes and LT codes using the ISSR distributions and the DRSD with/without the RCSS algorithm at the encoder with respect to γ_{succ} when $k = 100$ and the size of buffer is limited to $0.3 \leq \beta \leq 1$.

Chapter 4

Write-Once Memory Codes Using Rate-Compatible LDGM Codes

NAND flash memory has been widely used in various storage systems due to its outstanding features over hard disk drives (HDDs) such as higher storage density, lower power consumption, and better random read/write behavior. Especially, NAND flash-based solid-state drives (SSDs) have become one of the popular storage device. By injecting charge in a floating gate of a cell, a binary information can be stored, which is called single level cell (SLC) flash memory. In order to increase the storage density, more than one bit data is stored in one cell by precise programming scheme. It is called as multi-level cell (MLC) or triple-level cell (TLC) flash memory, if two or three bits of information are stored in a cell, respectively. Unfortunately, flash memory suffers from wearing out phenomenon due to frequent programming and erasing (P/E) operations, which degrades reliability and the lifetime of the storage device [67]. This drawback becomes more fatal for MLC flash memory, because it has multi-level states in the cells. To solve this problem, strong error correction codes such as con-

catenation of Bose-Chaudhuri-Hocquenghem (BCH) codes [68] and low-density parity check (LDPC) codes [69] are studied to guarantee reliability of flash memory. As an another approach, rewriting scheme for the flash memory was introduced recently to reduce the number of erasure operations. This approach reduces the factor of reliability degradation.

Write-once memory (WOM) codes are originally introduced in [70] for write-once memory such as punch cards or compact disc and their capacities are studied in [71]. Clearly, WOM codes allow multiple writes on the same cell without invoking block erasure operation. Recently, low complexity WOM codes are constructed by polar codes and low-density generator matrix (LDGM) codes in [72] and [40], respectively. In [40] encoding of WOM code is regraded as solving a binary erasure quantization (BEQ) problem by an LDGM code with a simple message passing algorithm. However, the encoder will fail when the probability of initial state in an invalid page is lower than the rate of rewriting code. In that case, the corresponding page cannot be reused for the second write and it needs to be erased.

In this chapter, a rewriting scheme is proposed for the flash memory is by multi-rate BEQ codes, which are implemented by rate-compatible (RC) LDGM codes. By adjusting rate of rewriting code, we can reuse invalid pages more efficiently. RC-LDGM codes are constructed from protographs with desired rates, each of which is optimized by computer search. In addition, we select pages from predetermined candidate pages for the second write. Since the encoder of WOM code is able to recognize whether a new message can be written over the selected pages or not, invalid pages in the block can be reused further if combinations of the selected pages and their ordering are carefully chosen.

4.1 Preliminaries

4.1.1 NAND Flash Memory

First physical characteristics of NAND flash memory is reviewed. In the NAND flash memory, charge can be injected into a floating gate in a cell, called *programming operation* and removed from the cell by *erasure operation*. When charge is captured in the floating gate, the transistor will be turned on if a voltage above threshold voltage is applied at the control gate. If there is no charge in the floating gate, then the transistor cannot be turned on with the same voltage. Thus, states of the cell can be distinguished by measuring current, which is *read operation*.

A NAND flash memory has asymmetry property between program and erasure operations due to its manufactured feature. The flash memory unit for reading and programming operations is called *page*, which is composed of cells. The size of page may vary from 1KB to 16KB [67]. On the other hand, a *block* is the unit of erasure operation and it may contain 64–384 pages. Note that each page in a block must be erased before writing new messages, which causes wearing out the flash memory [73]. Hence, as the number of P/E cycles increases, raw bit error rate (RBER) of the memory device increases, which results in the limitation of device lifetime. Therefore, reducing the number of erasure operations is one of key approaches to extend the lifetime of the flash memory.

4.1.2 Rewriting Schemes for Flash Memory

Recently, rewriting schemes for the flash memory by WOM codes have been researched [74]–[77]. Yadgar *et al.* [74] proposed a rewriting scheme with polar code,

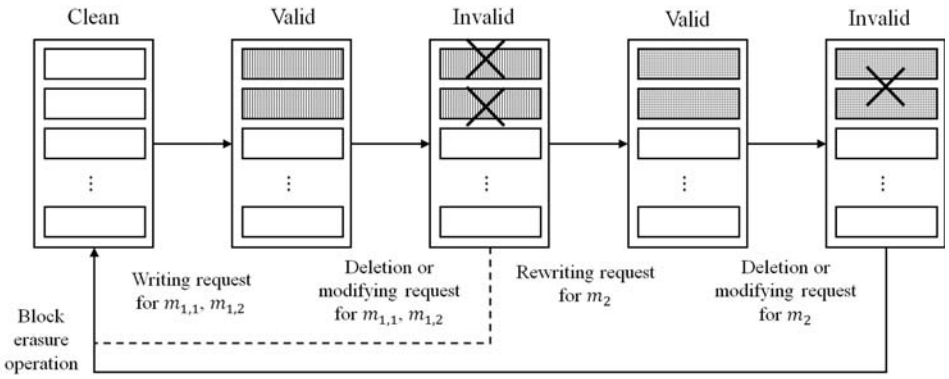

Figure 4.1: Cycles of rewriting scheme in a block.

called the reusable SSD, in which invalid pages are reused for the second write. At the first write, two messages $\mathbf{m}_{1,1}$ and $\mathbf{m}_{1,2}$ are written in two clean pages. When the messages are updated, the erasure operations are not invoked immediately but only the statuses of the corresponding pages are changed to invalid. While the pages wait for block erasure operation in the conventional scheme, the reusable SSD takes advantage of invalid pages for rewriting. In other words, block erasure is not invoked and new message $\mathbf{m}_2$ for the second write is encoded by the WOM encoder and rewritten over two invalid pages. After that, the status of page pair becomes valid again. If we want to update $\mathbf{m}_2$, the status of page pair is changed to invalid and then these pages wait for the block erasure operation. Finally, the block is erased when the number of invalid pages exceed predetermined value. Hence the length of P/E cycle in a block is increased from 3 to 5, which is shown in Fig. 4.1. As a result, it is known that the number of erasures is reduced by 33% in most cases of SLC flash-based SSD.

Generally, the low-high programming is used to avoid programming interference for MLC flash memory, i.e., low page is programmed first and then high page is pro-

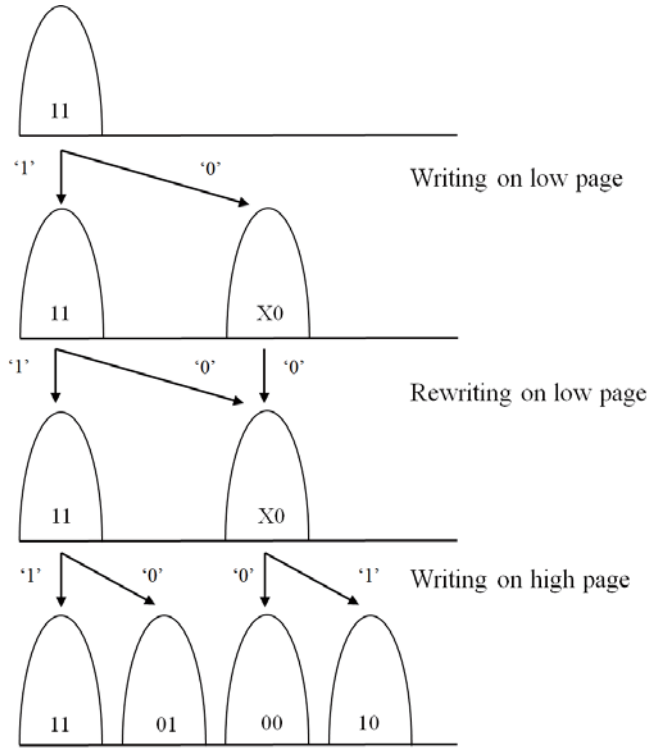

Figure 4.2: The LLH programming scheme for MLC flash memory.

grammed. Unlike the low page, the value of high page which increases the level of cell depends on that of low page. More precisely, if the value of low page is '1' then '0' increases the cell level while '1' programs the cell when the value is '0' at the low page. In addition, some transitions cannot be implemented in MLC flash memory even though they are not violating the WOM constraint with alphabet size four. Thus, adopting WOM codes to both pages in MLC is troublesome. To resolve this drawback, [76] and [77] suggested the low-low-high (LLH) programming which reuses only low pages and the corresponding state transitions of LLH are shown in Fig. 4.2. It is shown that the rewriting with LLH programming scheme gives 20% reduction in erasures.

4.1.3 Construction of Rewriting Codes by BEQ Codes

In this subsection, we briefly overview BEQ problem and the construction of rewriting codes by BEQ codes.

Definition 4.1 *A binary erasure channel function* $BEC : \{0,1\}^n \times \{0,1\}^n \to \{0,1,*\}^n$ *is defined as follows. Let* $\mathbf{z}$ *and* $\mathbf{v}$ *be a binary vector and an erasure location vector, respectively. For* $i = 1, \ldots, n$*, the* i*th component of* $\mathbf{r} = BEC(\mathbf{z}, \mathbf{v})$ *is given as*

$$r_i = BEC(\mathbf{z}, \mathbf{v})_i = \begin{cases} z_i, & \text{if } v_i = 0 \\ *, & \text{if } v_i = 1 \end{cases} \tag{4.1}$$

where $*$ *denotes an erasure.*

Definition 4.2 *Let* $D(\mathbf{r}, \hat{\mathbf{r}})$ *be a distortion function between a binary* n*-tuple vector with erasures* $\mathbf{r}$ *and a binary* n*-tuple vector* $\hat{\mathbf{r}}$*. Then we have*

$$D(\mathbf{r}, \hat{\mathbf{r}}) = \sum_{i=1}^{n} d_i \tag{4.2}$$

where d_i *is given as*

$$d_i = \begin{cases} 0, & \text{if } r_i = * \\ 0, & \text{if } r_i \neq *, r_i = \hat{r}_i \\ 1, & \text{otherwise.} \end{cases} \tag{4.3}$$

In [78], they introduced a binary erasure quantizer $Q(\mathbf{r})$ which is defined by a binary linear code $\mathcal{C}_Q$ with a generator matrix $\mathbf{G}_Q$ and a parity-check matrix $\mathbf{H}_Q$. The erasure quantizer $Q(\mathbf{r})$ compresses the vector $\mathbf{r}$ into the vector $\mathbf{u}$ without any

distortion, i.e., $D(\mathbf{r}, \mathbf{u}\mathbf{G}_Q) = 0$, which results in a valid codeword $\mathbf{u}\mathbf{G}_Q \in \mathcal{C}_Q$. Here, $\mathcal{C}_Q$ is called a BEQ code. It is known that an LDGM code, a dual of LDPC code, can be used as a BEQ code. Clearly, BEQ problem with LDGM code is dual of BEC problem with LDPC code, which can be solved by well known simple message passing algorithm, called belief propagation decoder. Thus, complexity of the binary erasure quantizer is linear.

In [40], they proposed a rewriting code $\mathcal{C}_R$ which is a collection of all cosets of BEQ code $\mathcal{C}_Q$ in $\mathbb{F}_2^n$. In order to rewrite arbitrary k bit message onto the given n cells, a new message vector $\mathbf{m}$ is transformed into n tuple vector $\mathbf{z} = \mathbf{m}\left(\mathbf{H}_Q^{-1}\right)^\intercal$, where $\mathbf{H}_Q^{-1}$ is called a transform matrix and $(\cdot)^\intercal$ denotes the transpose. Then $\mathbf{z}$ is converted to the erased vector $\mathbf{r}$ by the BEC function with given cell state vector (erasure location vector) $\mathbf{v}$. The erased vector $\mathbf{r}$ is compressed via the binary erasure quantizer $Q(\cdot)$ and then the corresponding codeword and $\mathbf{z}$ is added to make a rewriting vector (new cell state vector) $\mathbf{x}$, which is a codeword of WOM code. The block diagrams of rewriting and reconstruction are shown in Fig. 4.3 and Fig. 4.4.

Here, the rates of the first and the second writes are $R_1 = 1$ and $R_2 = k/n$, respectively. Also the sum-rate is $R_{sum} = R_1 + R_2 = 1 + k/n$. It is known that when R_1 is fixed to 1, capacity of two-write WOM code is known as $(1, 0.5)$. Thus, if we use a parity-check matrix of a capacity achieving LDPC code over binary erasure channel as a generator matrix of BEQ code, the rewriting capacity can be asymptotically achieved.

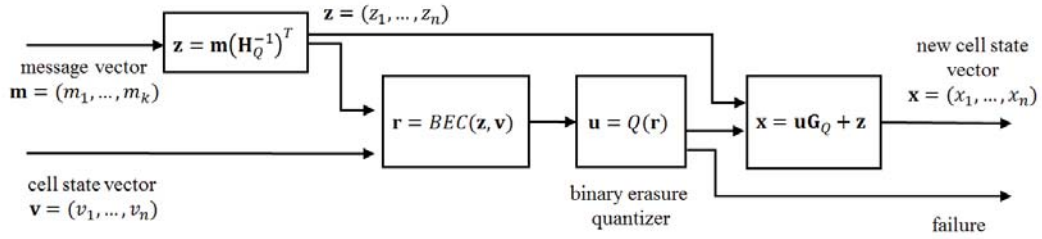

Figure 4.3: Block diagram of rewriting (encoding) process by BEQ code.

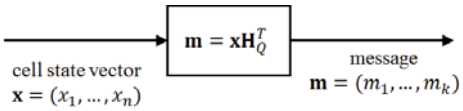

Figure 4.4: Block diagram of reconstruction (decoding) process by BEQ code.

4.2 Proposed Rewriting Codes

4.2.1 System Model

In this chapter, we focus on two writes on SLC flash memory and further extend to MLC flash memory using LLH programming scheme. It is assumed that there are L logical pages in a block and the size of each page is k bits. Let ϵ denote the probability that cell is in initial state, that is, $\Pr[v = 1] = \epsilon$. Further, it is considered that only information bits, systematic part of error correction code, are stored in the logical page and the parity bits are stored in spare page. It is also assumed that the second writing is invoked after sufficient elapsed time from the first write, which means that preprocessing can be done before request of the second write. Moreover, if an LDGM code is applied as a BEQ code in practical system, there exists performance degradation due to its finite length. In other words, the rewriting rate has to be reduced slightly.

Therefore, at least two invalid pages and extra α page is used for each writing request, where $0 < \alpha < 1$.

4.2.2 Multi-rate Rewriting Codes

In [40], only a single LDGM code is used as a linear BEQ code, which implies the fixed rewriting code rate. Thus, when the portion of 1's (initial states) in a cell state vector is below target $\hat{\epsilon}$, the new message cannot be written over the cell state vector. Therefore, those pages will not be used for rewriting and have to be thrown away.

In the communication systems, erasure patterns are randomly generated in BEC and occasionally they cause decoding failures even though error correction codes are used. The encoder is not able to recognize decoding failure until it receives a feedback from the receiver. With the feedback signal, the encoder generates and transmits additional parity bits, called incremental redundancy hybrid automatic repeat request (IR-HARQ). Similarly, when the binary erasure quantizer in the encoder is not able to find any valid codeword in the BEQ code for a given cell state vector, the encoder of WOM code recognizes encoding failure. Then we may apply modified encoding procedure. For example, by reducing the rewriting code rate, the encoder can further utilize invalid pages.

Here, a new encoding scheme using $\beta + 1$ multi-rate LDGM codes as BEQ codes is proposed, that is, $\mathcal{C}_Q^{(0)}, \mathcal{C}_Q^{(1)}, \cdots, \mathcal{C}_Q^{(\beta)}$, where β is nonnegative integer. For $i = 0, \ldots, \beta$, the rewriting code rate of $\mathcal{C}_Q^{(i)}$ is $R_{2,i}$. For simplicity, it is assumed that $R_{2,\beta} < \cdots < R_{2,0}$. First, a cell state vector is divided into $\beta + 3$ parts, i.e., a cell

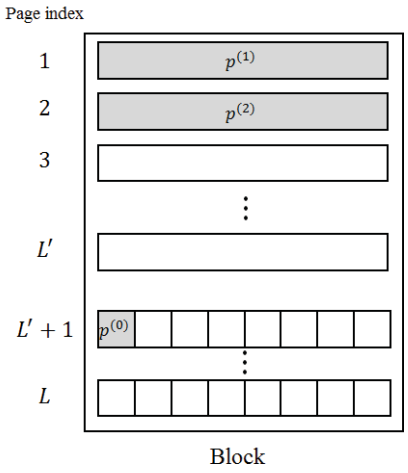

Figure 4.5: An example of cell state vector $\mathbf{v} = \left(p^{(0)}, p^{(1)}, p^{(2)}\right)$.

state vector which contains invalid pages can be represented as

$$\mathbf{v} = \left(p^{(0)}, p^{(1)}, p^{(2)}, \ldots, p^{(\beta+2)}\right) \tag{4.4}$$

where sizes of $p^{(0)}$ and $p^{(i)}$ are α page and one page, respectively for $i = 1, \ldots, \beta+2$. An example of cell state vector with $\alpha = 1/8$ and $\beta = 0$ is shown in Fig. 4.5.

Definition 4.3 *If the encoder succeeds rewriting over the given cell state vector, then the cell state vector is said to be rewritable.*

Now, one of performance criteria for SSD with WOM code, called writing efficiency, is defined as follows.

Definition 4.4 *Assume that L pages are in a block and t writes are allowed. Then writing efficiency is defined as*

$$\eta = \frac{\sum_{i=1}^{t} W^{(i)}}{L} \tag{4.5}$$

where $W^{(i)}$ is the expected number of one-page messages written at the ith write.

In this scenario, only two writes are used, i.e., $t = 2$, and $\beta + 1$ multi-rate BEQ codes are applied at the second writing. Thus, we have

$$W^{(2)} = \sum_{i=0}^{\beta} W_i \tag{4.6}$$

where W_i is the expected number of one-page messages written by the ith BEQ code $\mathcal{C}_Q^{(i)}$ at the second writing. Note that the best case is when all rewritings are successful with the highest rate. In other words, only $2 + \alpha$ pages are required for every one-page message. As a result, the number of maximum rewriting of one-page messages is $W_{max} = \left\lfloor \frac{L}{2+\alpha} \right\rfloor$ and we have

$$\sum_{i=0}^{\beta} W_i(2 + i) + \lceil \alpha W_{max} \rceil \leq L. \tag{4.7}$$

4.2.3 Page Selection for Rewriting

In this subsection, a new page selection scheme is proposed. While high quality of channels cannot be chosen in conventional communication systems, rewritable cell state vectors can be selected from a number of combinations since there are at most L invalid pages in a block. Thus, by carefully selecting cell state vectors, the writing efficiency can be enhanced.

Assume that the status of the lth page is changed from valid to invalid, for $1 \leq l \leq L'$, where $L' = L - \lceil \alpha W_{max} \rceil$. Then it can be checked whether the lth page is a rewritable candidate or not for the second write.

Remark 4.5 *A sufficient condition for a cell state vector to be rewritable is that the cell state vector does not contain any local stopping sets in each part $p^{(i)}$.*

Above condition can be easily checked by the message passing algorithm. First, we construct a temporary cell state vector $\mathbf{v}_{\langle l \rangle, i}$ for the lth page given as

$$\mathbf{v}_{\langle l \rangle, i} = \left(\mathbf{1}_{\alpha k}, p_i^{(1)}, p_i^{(2)}, \ldots, p_i^{(2+\beta)} \right) \tag{4.8}$$

where $p_i^{(j)}$ is a k-tuple binary vector and

$$p_i^{(j)} = \begin{cases} p_{l,old}, & \text{if } j = i \\ \mathbf{1}_k, & \text{otherwise} \end{cases} \tag{4.9}$$

for $i, j = 1, \ldots, 2 + \beta$ and $p_{l,old}$ is the k-tuple binary vector written at the first write in the lth page. Here, the corresponding erased vector is given as

$$\mathbf{r}_{\langle l \rangle, i} = BEC(\mathbf{0}_n, \mathbf{v}_{\langle l \rangle, i}) \tag{4.10}$$

where $n = (\alpha + 2 + \beta)k$ and $\mathbf{1}_n$ and $\mathbf{0}_n$ denote all one and all zero vectors with length n. Then the quantization algorithm is applied to each erased vector $\mathbf{r}_{\langle l \rangle, i}$. Finally, a $(2 + \beta)$-tuple vector $\mathbf{q}_{\langle l \rangle} = (q_{\langle l \rangle, 1}, \ldots, q_{\langle l \rangle, 2+\beta})$ is given as

$$q_{\langle l \rangle, i} = \begin{cases} 0, & \text{if } Q\left(\mathbf{r}_{\langle l \rangle, i} \right) = \text{Failure} \\ 1, & \text{otherwise} \end{cases} \tag{4.11}$$

for $i = 1, \ldots, (2 + \beta)$ and the quantization $Q\left(\mathbf{r}_{\langle l \rangle, i} \right)$is performed with BEQ code $\mathcal{C}_Q^{(2+\beta)}$ and represented in Algorithm 4.1.

Let θ be the maximum trial number for each BEQ code $\mathcal{C}_Q^{(i)}$ for rewriting, where $i = 0, \ldots, \beta$. Initially, i is set to 0 and then $p^{(0)}$ and $p^{(i)}$ are randomly selected from the predetermined fragmented page and one-page invalid pages, respectively. For simplicity, it is assumed that the indices of $p^{(1)}$ and $p^{(2)}$ are given as l_1 and l_2. From Lemma

Algorithm 4.1 Quantization with BEQ Code [40]

Input: $\mathbf{m}, \mathbf{v}, \mathcal{C}_Q$

$\mathbf{z} \leftarrow \mathbf{m}\left(\mathbf{H}_Q^{-1}\right)^{\mathsf{T}}$

$\mathbf{r} \leftarrow BEC(\mathbf{z}, \mathbf{v})$

$\mathbf{u} \leftarrow Q(\mathbf{r})$

if $D(\mathbf{r}, \mathbf{u}\mathbf{G}_Q) = 0$ **then**

    $\mathbf{x} \leftarrow \mathbf{u}\mathbf{G}_Q + \mathbf{z}$

    return $\mathbf{x}$

else

    return Failure

end if

4.5, the selected page $p^{(j)}$ should satisfy $q_{\langle l_j \rangle, j} = 1$ for $j = 1, 2$. Finally, the BEC function and quantization with $\mathcal{C}_Q^{(i)}$ are applied to the constructed cell state vector $\mathbf{v}$. A valid output is returned if quantization is successful. On the other hand, if the quantizer fails, fragments of cell state vector are selected again and the previous steps are repeated. In other words, the encoder increases i by 1 and repeats above procedures. The proposed scheme is described precisely in Algorithm 4.2.

Algorithm 4.2 Procedure of the Second Writing

Input: $\mathbf{m}_{new}, \alpha, \beta, \theta, L, \left\{\mathcal{C}_Q^{(0)}, \ldots, \mathcal{C}_Q^{(\beta)}\right\}, \left(\mathbf{q}_{\langle 1\rangle}, \ldots, \mathbf{q}_{\langle L'\rangle}\right)$

$isSucc \leftarrow$ false
for $i = 0$ to β **do**
  for $t = 1$ to θ **do**
    Randomly select l_0 from the fragments and $p^{(0)} \leftarrow p_{\langle l_0\rangle}$
    for $j = 0$ to $2 + i$ **do**
      Randomly select l_j satisfying $q_{\langle l_j\rangle,j} = 1$ and $p^{(j)} \leftarrow p_j$
    end for
    $\mathbf{v} \leftarrow \left(p^{(0)}, p^{(1)}, \ldots, p^{(i+2)}\right)$
    $\mathbf{x} \leftarrow$ output of Algorithm 4.1 with $\mathbf{m}_{new}$, $\mathbf{v}$, $\mathbf{G}_Q^{(i)}$ as inputs
    if ($\mathbf{x} \neq$ Failure) **then**
      $isSucc \leftarrow$ true
      break
    end if
  end for
  if ($isSucc =$ true) **then**
    return $\mathbf{x}$
  end if
end for
return Failure

4.3 RC-LDGM Codes

It is clear that a generator matrix of LDGM code is the same as the parity-check matrix of LDPC code. Thus, construction of good LDGM code is equivalent to that of LDPC code, which has been intensively studied for decades. Especially, protograph-based construction, a subclass of multi-edge-type LDPC codes, is introduced and shown that it has excellent error correcting performance and low complexity [79]. In order to implement structured LDPC codes with multi-rate, RC-LPDC codes have attracted attention. More than two codes have rate-compatible property if codewords of the higher rate code are embedded in the codewords of the lower rate codes. Recently, construction of protograph-based RC-LDPC codes by extending and puncturing approaches is studied in [80]–[83]. Generally, it is done by exhaustive computer search, where limiting search space is major concern. In [83], raptor-like structure is adopted and shown that the corresponding RC-LDPC codes have outstanding performance while providing wide rate-compatibility.

Similarly, we construct protograph-based raptor-like RC-LDGM codes using computer search with PEXIT analysis over BEC, which is EXIT analysis for protograph [84]. The protograph which has the best BP threshold is selected from the all candidates. First, we start with the base matrix of well designed R4JA code with rate $1/2$ [85] represented by

$$\mathbf{B}_{\mathrm{R4JA}} = \begin{bmatrix} 2 & 2 & 1 & 1 \\ 1 & 1 & 3 & 1 \end{bmatrix} \tag{4.12}$$

and the corresponding protograph is shown in Fig. 4.6. By setting the maximum par-

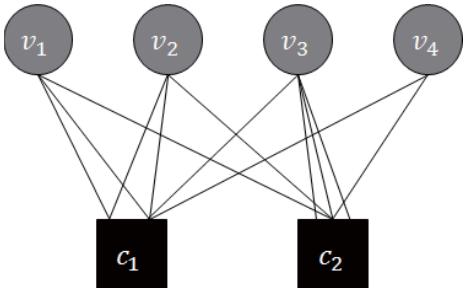

Figure 4.6: The protograph for the base matrix of R4JA code.

allel edges to p, the base matrix for the desired rate $1/3$ is given as

$$\mathbf{B}_{1/3} = \left[\begin{array}{cccc:cc} 2 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 3 & 1 & 0 & 0 \\ \hdashline x_1 & x_2 & x_3 & x_4 & 1 & 0 \\ x_5 & x_6 & x_7 & x_8 & 0 & 1 \end{array}\right] \tag{4.13}$$

where $0 \leq x_i \leq p$ for $i = 1, 2, \ldots, 8$. Using the computer search, $x_1 = 1, x_3 = 3, x_7 = 3$, and otherwise $x_i = 0$ are found when $p = 3$. Similarly, the protograph of code with the lowest rate for $\beta = 2$, $\mathbf{B}_{1/4}$, is constructed successively, which is represented as

$$\mathbf{B}_{1/4} = \left[\begin{array}{ccc:c} \mathbf{B}_{\mathrm{R4JA}} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_1 & \mathbf{I}_2 & \mathbf{0} \\ \hdashline \mathbf{B}_2 & \mathbf{0}_2 & \mathbf{I}_2 \end{array}\right]. \tag{4.14}$$

Here, $\mathbf{B}_2$ is obtained as

$$\mathbf{B}_2 = \begin{bmatrix} 1 & 0 & 3 & 0 \\ 1 & 0 & 3 & 0 \end{bmatrix}. \tag{4.15}$$

Then, the first step of lifting algorithm is applied to avoid parallel edges in the base matrix. For example, it becomes 24×32 matrix $\mathbf{B}'_{1/4}$ when lifting factor 4 is used. Note

that the required set of rates for the proposed rewriting scheme is $\left\{R_{2,0}, \cdots, R_{2,\beta}\right\} = \left\{\frac{1}{2+\alpha}, \cdots, \frac{1}{2+\beta+\alpha}\right\}$. Thus, $\mathbf{B}'_{1/4}$ needs to be extended again. With the protograph having $R_{2,0}$ is constructed based on the lifted version of $\mathbf{B}_{\mathrm{R4JA}}$. The desired protograph $\mathbf{B}'_{8/17}$ for $\beta = 0$ is represented as

$$\mathbf{B}'_{8/17} = \left[\begin{array}{c:ccccc} 1 & x_1 & x_2 & \cdots & x_{16} \\ \hdashline x_{17} & & & & \\ x_{18} & & & & \\ \vdots & & \mathbf{B}'_{\mathrm{R4JA}} & & \\ x_{24} & & & & \end{array}\right] \tag{4.16}$$

where $0 \leq x_i \leq 1$ for $i = 1, \ldots, 24$ and the computer search is applied. Finally, the second step of lifting algorithm, called circulant progressive edge growth (CPEG) algorithm [86], is applied to guarantee the girth of code.

4.4 Numerical Analysis

In this chapter, the parameters for the proposed scheme are set as $k = 8192$, $L = 128$, $\alpha = 1/8$, and $\beta = 2$. The parity-check matrices of RC-LDPC codes, $\mathbf{H}_A$, $\mathbf{H}_B$, and $\mathbf{H}_C$, are constructed from Section 4.3, each of which has the lifting factor as 1024 for CPEG and the girth larger than or equal to 8. The rates and thresholds of constructed LDPC codes, $\mathcal{C}_A, \mathcal{C}_B$, and $\mathcal{C}_C$, are shown in Table 4.1. In addition, BER performance of each LDPC code is shown in Fig. 4.7. These parity-check matrices are used as the generator matrices of BEQ codes, i.e., $\mathbf{G}_Q^{(0)} = \mathbf{H}_A$, $\mathbf{G}_Q^{(1)} = \mathbf{H}_B$, and $\mathbf{G}_Q^{(2)} = \mathbf{H}_C$.

The performance of writing efficiency of the proposed scheme is compared with

Table 4.1: Constructed RC-LDPC codes

Code	$\mathcal{C}_A$	$\mathcal{C}_B$	$\mathcal{C}_C$
Rate	0.471	0.320	0.242
Threshold	0.470	0.642	0.734

those of the conventional schemes, i.e., without WOM code, with Rivest-Shamir $\langle 4 \rangle^2/3$ WOM code [70], and with WOM code which has fixed rewriting rate [40]. The simulation results are shown in Fig. 4.8. It is shown that the proposed scheme outperforms SSD with simple Rivest-Shamir WOM code when $\epsilon > 0.52$.

In addition, it is verified that the page selection scheme, i.e., constructing rewritable cell state vector, improves rewriting capability. The simulation result of the proposed scheme with various θ for $\epsilon = 0.53$ is shown in Table 4.2. As θ increases, the writing efficiency also increases. Therefore, more write request can be done before invoking the block erasure operation at a cost of increasing iteration number, which causes higher computational complexity. However, it is assumed that there is enough time for preparing the second writes. Thus, we can reduce the erasure operation in SSD storage device by carefully choosing θ.

Furthermore, when the proposed rewriting scheme is implemented for MLC using LLH programming, we have $\eta_{\mathrm{MLC}} = 1.234$ for the best case. Even though the gain for rewriting scheme is decreased, the proposed scheme still has advantage over the conventional scheme which does not use WOM codes.

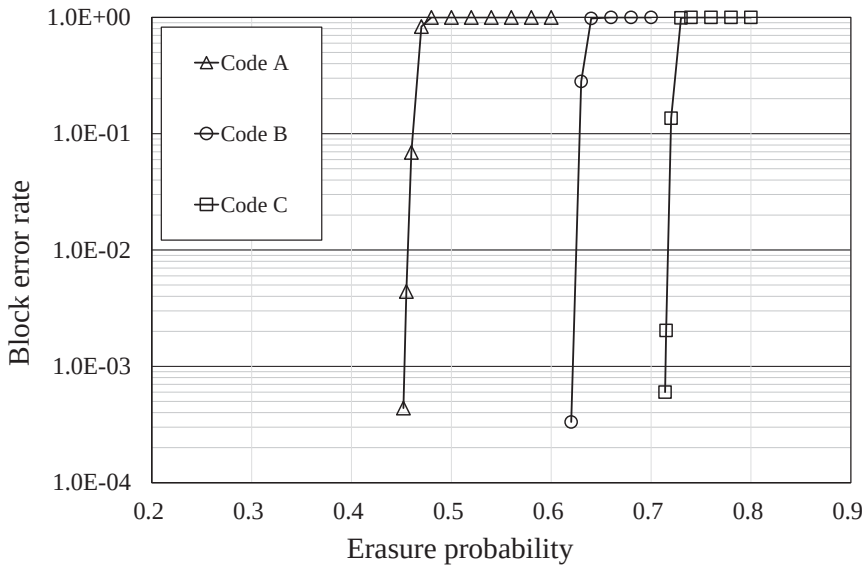

Figure 4.7: Block error rate of constructed LDPC codes, $\mathcal{C}_A$, $\mathcal{C}_B$, and $\mathcal{C}_C$ in BEC.

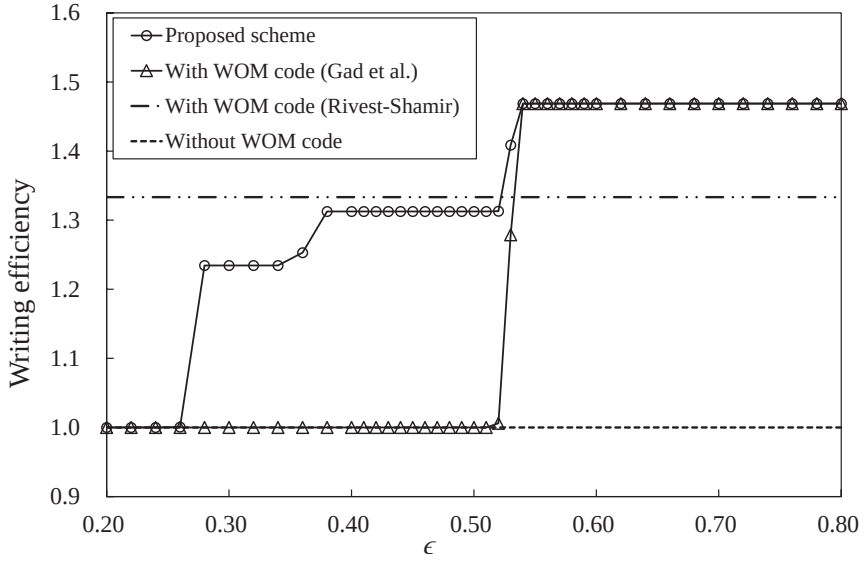

Figure 4.8: Comparison of writing efficiency of the proposed scheme with those of the conventional schemes.

Table 4.2: Writing efficiency η with various θ for $L = 128$

θ	W_0	W_1	Iter.	η
1	19.17	26.90	1.59	1.3599
3	32.38	18.06	2.56	1.3941
5	37.96	14.36	3.35	1.4088
10	42.37	11.40	5.07	1.4201
20	45.37	9.42	7.38	1.4281

* W_2 for all θ is equal to zero.

Chapter 5

Conclusions

In this dissertation, research on the sparse codes and their applications are presented. Specially, fountain and WOM codes which are constructed by sparse graphs are studied.

In Chapter 2, some preliminaries for spare codes are overviewed. In order to utilize their excellent properties, a simple message passing algorithm is also introduced.

In Chapter 3, fountain codes for receivers with limited hardware are proposed. Generally, well designed fountain codes with respect to coding overhead such as LT codes and ZD fountain codes have poor ISRR. Thus, they have to store lots of unrecovered code symbols in the buffer during decoding process. In this chapter, two new classes of fountain codes based on the batched zigzag coding for receivers with small buffer size, called BZ fountain codes and TBZ fountain codes are proposed. A method to obtain code symbol degree distributions for the proposed codes via the ripple size evolution is also proposed. By carefully generating batches of size larger than one to be zigzag decodable, the proposed fountain codes have improved intermediate recov-

ery at low symbol erasure rates. Consequently, the coding overhead performance of the proposed codes is little affected by the restriction of buffer size. It is verified that the proposed codes outperform the conventional fountain codes via numerical analysis with respect to intermediate recovery and coding overhead when message symbol is short, symbol erasure rate is low, and the buffer size is limited.

In Chapter 4, WOM codes with message passing algorithm are introduced. While NAND flash memory has many advantages such as high storage density and low latency, it can be worn out due to frequent erasure operations. Thus, reducing the number of erasure operations can be one of key approach to extend lifetime of NAND flash-based SSD. In this chapter, a rewriting scheme with multi-rate BEQ codes is proposed. Also by selecting invalid pages and ordering those pages appropriately to make cell state vectors rewritable, writing efficiency is improved. It is verified via numerical analysis that SSD with the proposed rewriting scheme outperforms SSD without and with conventional WOM codes with respect to writing efficiency.

Bibliography

[1] C. E. Shannon, “A mathematical theory of communications,” *Bell Syst. Tech. J.* vol. 27, pp. 379–423, pp. 623–656, Jul., Oct. 1948.

[2] R. G. Gallager, “Low density parity check codes,” *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting codes and decoding: Turbo codes,” in *Proc. IEEE Int. Conf. Commun.*, May 1993, pp. 1064–1070.

[4] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[5] T. J. Richardson and R. L. Urbanke, “The capacity of low-density parity-check codes under message passing decoding,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599-618, Feb. 2001.

[6] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, “Improved low-density parity-check codes using irregular graphs,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[7] S.-Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[8] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

[9] IEEE, "IEEE standards for local and metropolitan area network. Part 16: Air interface for fixed broadband wireless access systems," *IEEE Std 802.16-2009*, May 2009.

[10] IEEE, "IEEE standards for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. Amendment 5: Enhancement for higher throughput," *IEEE Std 802.11n-2009*, Oct. 2009.

[11] IEEE, "IEEE standards for information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specification. Amendment 1: Physical layer and management parameters for 10 Gb/s operation, type 10GBASE-T," *IEEE Std 802.3an-2006*, Sep. 2006.

[12] ETSI, "Digital video broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services,

news gathering and other broadband satellite applications (DVB-S2)," *EN 302 307 v1.2.1*, Aug. 2009.

[13] L. Ping, S. Chan, and K. L. Yeung, "Iterative decoding of multi-dimensional concatenated single parity check codes," in *Proc. IEEE Int. Conf. Commun. (ICC)* Jun. 1998, pp. 131–135.

[14] T. R. Oenning and J. Moon, "A low-density generator matrix interpretation of parallel concatenated single bit parity codes," *IEEE Trans. Magn.*, vol. 37, pp. 737–741, Mar. 2001.

[15] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[16] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding 5th IMA Conf.*, C. Boyd, Ed., *Lecture Notes in Computer Science*, no. 1025. Berlin, Germany: Springer, 1995, pp. 100–111.

[17] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data", in *Proc. ACMSIGCOMM*, Jan. 1998, pp. 56–67.

[18] M. Luby, "LT codes," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, Nov. 2002, pp. 271–282.

[19] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.

[20] D.J.C. MacKay, G. Mitchison, and P. L. McFadden, "Sparse-graph codes for quantum error correction," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2315-2330, Oct. 2004.

[21] I. B. Djordjevic , "Quantum LDPC codes from balanced incomplete block designs," *IEEE Commun. Lett.*, vol 12, no. 5, pp. 389–391, May. 2008.

[22] P. Tan and Jing Li, "Efficient quantum stabilizer codes: LDPC and LDPC-convolutional constructions," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 476–491, Dec. 2009.

[23] A. Thangaraj, S. Dihidar, A. R. Calderbank, S. W. McLaughlin, and J.-M. Merolla, "Applications of LDPC codes to the wiretap channel," *IEEE Trans. Inf. Theory*, vol. 53, no. 8, pp. 2933–2945, Aug. 2007.

[24] M. Baldi, M. Bianchi, and F. Chiaraluce, "Coding with scrambling, concatenation, and HARQ for the AWGN wire-tap channel: A security gap analysis," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 3, pp. 883–894, Jun. 2012.

[25] D. Klinc, J. Ha, S. W. McLaughlin, J. Barros, and B.-J. Kwak, "LDPC codes for the Gaussian wiretap channel," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 3, pp. 532–540, Sep. 2011

[26] M. Baldi, M. Bodrato, and F. Chiaraluce, "Security and complexity of the McEliece cryptosystem based on quasi-cyclic low-density parity-check codes," *IET Information Security*, vol. 7, no. 3, pp. 212–220, Sep. 2013.

[27] A.D. Liveris, Z. Xiong, and C. N. Georghiades, "Compression of binary sources with side information at the decoder using LDPC codes," *IEEE Commun. Lett.*, vol. 6, no. 10, pp. 440-442, Oct. 2002.

[28] Y. Matsunaga and H. Yamamoto, "A coding theorem for lossy data compression by LDPC codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 9, pp. 2225–2229, Sep. 2003.

[29] G. Caire, S. Shamai, and S. Verdu, "Universal data compression with LDPC codes," in *Proc. 3rd Int. Symp. Turbo Codes and Related Topics*, Sep. 2003, pp. 55–58.

[30] C.-F. Lan, A. D. Liveris, K. Narayanan, Z. Xiong, and C. Georghiades, "Slepian-Wolf coding of multiple M-ary sources using LDPC codes," in *Proc. Data Compression Conference (DCC)*, Mar. 2004, p. 549.

[31] S. Miyake, "Lossy data compression over Z_q by LDPC code," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2006, pp. 813–816.

[32] A. G. Dimakis, R. Smarandache, and P. O. Vontobel, "LDPC codes for compressed sensing," *IEEE Trans. Inf. Theory*, vol. 58, no. 5, pp. 3093–3114, May 2012.

[33] F. Zhang and H. D. Pfister, "Verification decoding of high-rate LDPC codes with applications in compressed sensing," *IEEE Trans. Inf. Theory*, vol. 58, no. 8, pp. 5042–5058, Aug. 2012.

[34] W. Z. Lu, K. Kpalma, and J. Ronisn, "Sparse binary matrices of LDPC codes for compressed sensing," in *Proc. Data Compression Conference (DCC)*, Apr. 2012, p. 405.

[35] L. Li and D. Qu, "Joint decoding of LDPC code and phase factors for OFDM systems with PTS PAPR reduction," *IEEE Trans. Veh. Technol.*, vol. 62, no. 1., pp. 444–449, Jan. 2013

[36] L. Li, D. Qu, and T. Jiang, "Partition optimization in LDPC-coded OFDM systems with PTS PAPR reduction," *IEEE Trans. Veh. Technol.*, vol. 63, no. 8, pp. 4108–4113, Oct. 2014.

[37] D. Qu, L. Li, and T. Jiang, "Invertible subset LDPC code for PAPR reduction in OFDM systems with low complexity," *IEEE Trans. Wireless Commun.*, vol. 13, no. 4, pp. 2204–2213, Apr. 2014.

[38] S. Shu, D. Qu, L. Li, and T. Jiang, "Invertible subset QC-LDPC codes for PAPR reduction of OFDM signals," *IEEE Trans. Broadcast.*, vol. 61, no. 2, pp. 290–298, Jun. 2015.

[39] S. Kumar, A. Vem, K. Narayanan, and H. D. Pfister, "Spatially-coupled codes for write-once memories," in *Proc. Allerton Conf. Commun., Control, Comput.*, Oct. 2015, pp. 125–131.

[40] E. E. Gad, W. Huang, Y. Li, and J. Bruck, "Rewriting flash memories by message passing," in *Proc. Int. Symp. on Information Theory (ISIT)*, Jul. 2015, pp. 646–650.

[41] J. H. Sørensen, P. Popovski, and J. Østergaard, "Design and analysis of LT codes with decreasing ripple size," *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3191–3197, Nov. 2012.

[42] H. Zhu, G. Zhang, and G. Li, "A novel degree distribution algorithm of LT codes," in *Proc. IEEE Int. Conf. Commun. Technol.*, Nov. 2008, pp. 221–224.

[43] K.-K. Yen, Y.-C. Liao, C.-L. Chen, and H.-C. Chang, "Modified robust soliton distribution (MRSD) with improved ripple size for LT codes," *IEEE Commun. Lett.*, vol. 17, no. 5. pp. 976–979, May 2013.

[44] Y. Zhao, F. C. M. Lau, Z. Zhu, and H. Yu, "Scale-free Luby transform codes," *Int. J. Bifurcation and Chaos*, vol. 22, no. 4, pp. 1250094-1–11, Apr. 2012.

[45] T. Nozaki, "Fountain codes based on zigzag decodable coding," in *Proc. Int. Symp. Inf. Theory Appl. (ISITA)*, Oct. 2014, pp. 274–278.

[46] ——, "Zigzag decodable fountain codes," [Online]. Available: http://arxiv.org/abs/1605.09125v1

[47] J. Qureshi, C. H. Foh, and J. Cai, "Primer and recent developments on fountain codes," *Recent Adv. Commun. Netw. Technol.*, vol. 2, pp. 2–11, 2013.

[48] S. Gollakota and D. Katabi, "Zigzag decoding: Combating hidden terminals in wireless networks," in *Proc. ACM SIGCOMM Conf. Data. Commun.*, Oct. 2008, pp. 159–170.

[49] C. W. Sung and X. Gong, "A zigzag-decodable code with the MDS property for distributed storage systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp. 341–345.

[50] J. Chen, H. Li, H. Hou, B. Zhu, T. Zhou, L. Lu, and Y. Zhang, "A new zigzag MDS code with optimal encoding and efficient decoding," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2014, pp. 1–6.

[51] H. Hou, K. W. Shum, M. Chen, and H. Li, "BASIC regenerating code: Binary addition and shift for exact repair," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2013, pp.1621–1625.

[52] J. Qureshi, C. H. Foh, and J. Cai, "Optimal solution for the index coding problem using network coding over GF(2)," in *Proc. IEEE Conf. Sensor, Mesh, Ad Hoc Commun., Netw. (SECON)*, Jun. 2012, pp. 209–217.

[53] C. Zhou, P. Mei, L. Huang, K. Liu, and Y. Wen, "An electronic shelf label system based on WSN," in *Proc. Int. Conf. Syst. Eng. Modeling (ICSEM)*, 2013, pp. 913–916.

[54] K. Alsmearat, M. Al-Ayyoub, and M. B. Yasseinz, "A new broadcast scheme for sensor networks," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Applicat. (AICCSA)*, Nov. 2014, pp. 824–828.

[55] R. Kumar, A. Paul, U. Ramachandran, and D. Kotz, "On improving wireless broadcast reliability of sensor networks using erasure codes," in *Proc. Second Int. Conf. Mobile Ad-hoc Sensor Netw.*, Dec. 2006, pp. 155–170.

[56] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quarter 2015.

[57] S. Sanghavi, "Intermediate performance of rateless codes," in *Proc. IEEE Inf. Theory Workshop*, Sep. 2007, pp. 478–482.

[58] S. Kim and S. Lee, "Improved intermediate performance of rateless codes," in *Proc. Int. Conf. Adv. Commun. Technol.*, Feb. 2009, pp. 1682–1686.

[59] A. Talari and N. Rahnavard, "On the intermediate symbol recovery rate of rateless codes," *IEEE Trans. Commun.*, vol. 60, no. 5, pp. 1237–1242, May 2012.

[60] A. Beimel, S. Dolev, and N. Singer, "RT oblivious erasure correcting," *IEEE/ACM Trans. Netw.*, vol. 15, no. 6, pp. 1321–1332, Dec. 2007.

[61] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, Oct. 2006, pp. 255–266.

[62] N. Thomos, R. Pulikkoonattu, and P. Frossard, "Growth codes: Intermediate performance analysis and application to video," *IEEE Trans. Commun.*, vol. 61, no, 11, pp. 4710–4721, Nov. 2013.

[63] Y. Cassuto and A. Shokrollahi, "Online fountain codes with low overhead," *IEEE Trans. Inf. Theory*, vol. 61, no. 6, pp. 3137–3149, Jun. 2015.

[64] S. Yang and R. W. Yeung, "Batched sparse codes," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5322–5346, Sep. 2014.

[65] X. Wang, A. Willig, and G. Woodward, "Improving fountain codes for short message lengths by adding memory," in *Proc. IEEE Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process.*, Apr. 2013, pp. 189–194.

[66] K. F. Hayajneh, S. Yousefi, and M. Valipour, "Improved finite-length Luby-transform codes in the binary erasure channel," *IET Commun.*, vol. 9, no. 8, pp. 1122–1130, May 2015.

[67] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 24–33.

[68] H. Choi, W. Liu, and W. Sung, "VLSI implementation of BCH error correction for multilevel cell NAND flash memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 843–847, May 2010.

[69] J. Kim and W. Sung, "Rate-0.96 LDPC decoding VLSI for soft-decision error correction of NAND flash memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 5, pp. 1004–1015, May 2014.

[70] R. L. Rivest and A. Shamir, "How to reuse a write-once memory," *Inf. Contr.*, vol. 55, no. 1–3, pp. 1–19, Dec. 1982

[71] C. Heegard, "On the capacity of permanent memory," *IEEE Trans. Inf. Theory*, vol. 31, no. 1, pp. 34–42, Jan. 1985.

[72] D. Burshtein and A. Strugatski, "Polar write once memory codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 5088–5101, Aug. 2013.

[73] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *J. Syst. Archit.*, vol. 55, no. 5–6, pp. 332–343, May 2009.

[74] G. Yadgar, E.Yaakobi, and A. Schuster "Write once, get 50% free: Saving SSD erase costs using WOM codes," in *Proc. File and Storage Technologies (FAST)*, Feb. 2015, pp. 257–271.

[75] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Writing cosets of a convolutional code to increase the lifetime of flash memory," in *Proc. Allerton Conf. Commun., Control, Comput.*, Oct. 2012, pp.308–318.

[76] F. Margaglia and A. Brinkmann, "Improving MLC flash performance and endurance with extended P/E cycles," in *Proc. Mass Storage Systems and Technologies (MSST)*, May 2015, pp. 1–12.

[77] F. Margaglia, G, Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann, "The devil is in the details: Implementing flash page reuse with WOM codes," in *Proc. USENIX Conf. on File and Storage Technologies (FAST)*, Feb. 2016, pp. 95–109.

[78] E. Martinian and J. S. Yedidia, "Iterative quantization using codes on graphs," in *Proc. Allerton Conf. Commun., Control, Comput.*, Oct. 2003.

[79] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," *Proc. IPN Progr. Rep. 42-154*, JPL, Aug. 2003.

[80] D. Divsalar, S. Donlinar, C. R. Jones, and K. Andrews, "Capacity-approaching protograph codes," *IEEE J. Sel. Areas Commun.*, vol. 27, no. 6, pp. 876–888, Aug. 2009.

[81] T. Nguyen, A. Nosratinia, and D. Divsalar, "The design of rate-compatible protograph LDPC codes," *IEEE Trans. Commun.*, vol. 60, no. 10, pp. 2841–2850, Oct. 2012.

[82] T. V. Nguyen and A. Nosratinia, "Rate-compatible short-length protograph LDPC codes," *IEEE Commun. Lett.*, vol. 17, no. 5, pp. 948–951, May 2013.

[83] T.-Y. Chen, K. Vakilinia, D. Divsalar, and R. D. Wesel, "Protograph-based raptor-like LDPC codes," *IEEE Trans. Commun.*, vol. 63, no. 5, pp. 1522–1532, May 2015.

[84] G. Liva and M. Chiani, "Protograph LDPC code design based on EXIT chart analysis," in *Proc. IEEE GLOBECOM*, Nov. 2007, pp. 3250–3254.

[85] D. Divsalar, C. Jones, S. Dolinar, J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," in *Proc. IEEE GLOBECOM*, Nov. 2005, pp. 1152–1156.

[86] Z. Li and B. V. Kumar, "A class of good quasi-cyclic low-density parity check codes based on progressive edge growth graph," in *Proc. Conf. Rec. 38th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2004, pp. 1990-1994.

초 록

본 논문은 저밀도 부호의 응용에 관한 것이며, 다음과 같은 두 가지 연구 결과를 포함하고 있다.

- 파운틴 부호
  - 묶음 지그재그 파운틴 부호 제안
  - 이단 묶음 지그재그 파운틴 부호 제안
- 일회 쓰기 가능 메모리 (WOM) 부호
  - 비율 호환형 (RC) 저밀도 생성 행렬 (LDGM) 부호를 이용한 WOM 부호 구현

먼저, 묶음 지그재그 파운틴 부호와 이단계 묶음 지그재그 파운틴 부호라고 불리는 두 가지 파운틴 부호 종류가 심볼 소실 채널을 위해 제안된다. 부호 심볼의 길이를 약간 늘임으로써 제안한 부호의 각각의 묶음에 포함되어 있는 메세지 심볼들이 낮은 복잡도를 가지는 지그재그 복호 알고리즘에 의해 복원될 수 있다. 따라서 제안하는 부호는 복호 과정동안 낮은 버퍼 사용량을 보인다. 이러한 특징은 하드웨어 자원의 제한이 있는 수신기로 구성된 브로드캐스팅 채널에 적합하다. 또한 묶음으로부터

해제되는 부호 심볼을 고려하는 리플 크기 진화 기법을 통해 제안한 부호의 부호 심볼의 차수 분포를 구하는 기법을 제안한다. 메세지 길이가 짧고, 심볼 소실 확률이 낮으며 사용가능한 버퍼 크기가 제한되어 있을 때, 제안하는 부호가 루비 변형 부호와 지그재그 파운틴 부호에 비해 중간 복호 성능과 부호 오버헤드 측면에서 우수한 성능을 가짐을 보인다.

두 번째로, WOM 부호를 저밀도 부호인 LDGM 부호를 이용해 구현하는 것에 대해 소개한다. 최근에 삭제 과정의 횟수를 줄여 메모리 장치의 수명을 늘리기 위해 WOM 부호가 낸드 플래시 기반 솔리드 스테이트 드라이브 (SSD)에 적용되었다. 본 논문에서는 SSD를 위한 다수의 이진 소실 양자화 (BEQ) 부호를 기반으로 새로운 다시 쓰기 기법을 제안하며 사용되는 BEQ 부호는 RC-LDGM 부호로 생성된다. 나아가 RC-LDGM 부호와 페이지 선택 기법을 함께 사용함으로써 쓰기 효율이 증가한다. 제안하는 다시 쓰기 기법을 사용하는 SSD는 단일 레벨 셀 (SLC) 및 다 레벨 셀 (MLC) 플리시 메모리에서 WOM 부호를 사용하지 않는 SSD와 기존의 WOM 부호를 사용하는 SSD보다 성능이 우수함을 확인한다.

주요어: 파운틴 부호, 저밀도 생성 행렬 부호, 저밀도 패리티 체크 부호, 낸드 플래시 메모리, 솔리드 스테이트 드라이브, 일회 쓰기 가능 메모리 부호.

학번: 2011-20923