CAPACITY ASSURANCE IN HOSTILE NETWORKS

By

Jian Li

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Electrical Engineering – Doctor of Philosophy

2015

ABSTRACT

CAPACITY ASSURANCE IN HOSTILE NETWORKS

By

Jian Li

Linear network coding provides a new communication diagram to significantly increase the network capacity by allowing the relay nodes to encode the incoming messages. However, this communication diagram is fragile to communication errors and pollution attacks. How to combat errors while maintaining the network efficiency is a challenging research problem. In this dissertation, we study how to combat the attacks in both fixed network coding and random network coding.

For fixed network coding, we provide a novel methodology to characterize linear network coding through error control coding. We propose to map each linear network coding to an error control coding. Under this mapping, these two codes are essentially identical in algebraic aspects. Meanwhile, we propose a novel methodology to characterize a linear network coding through a series of cascaded linear error control codes, and to develop network coding schemes that can combat node compromising attacks.

For random network coding, we propose a new error-detection and error-correction (EDEC) scheme to detect and remove malicious attacks. The proposed EDEC scheme can maintain throughput unchanged when moderate network pollution exists with only a slight increase in computational overhead. Then we propose an improved LEDEC scheme by integrating the low-density parity check (LDPC) decoding. Our theoretical analysis, performance evaluation and simulation results using ns-2 simulator demonstrate that the LEDEC scheme can guarantee a high throughput even for heavily polluted network environment.

Distributed storage is a natural application of network coding. It plays a crucial role in the current cloud computing framework in that it can provide a design trade-off between security management and storage. Regenerating code based approach attracted unique attention

because it can achieve the minimum storage regeneration (MSR) point and minimum bandwidth regeneration (MBR) point for distributed storage. Since then, Reed-Solomon code based regenerating codes (RS-MSR code and RS-MBR code) were developed. They can also maintain the MDS (maximum distance separable) property in code reconstruction. However, in the hostile network where the storage nodes can be compromised and the packets can be tampered with, the storage capacity of the network can be significantly affected. In this dissertation, we propose a Hermitian code based minimum storage regenerating (H-MSR) code and a Hermitian code based minimum bandwidth regenerating (H-MBR) code. We first prove that they can achieve the theoretical MSR bound and MBR bound respectively. We then propose data regeneration and reconstruction algorithms for the H-MSR code and the H-MBR code in both error-free networks and hostile networks. Theoretical evaluation shows that our proposed schemes can detect the erroneous decodings and correct more errors in the hostile network than the RS-MSR/RS-MBR code with the same code rate respectively.

Inspired by the novel construction of Hermitian code based regenerating codes, a natural question is how to construct optimal regenerating codes based on the layered structure like Hermitian code in distributed storage. Compared to the Hermitian based code, these codes have simpler structures and are easier to understand and implement. We propose two optimal constructions of MSR codes through rate-matching in hostile networks: 2-layer rate-matched MSR code and m-layer rate-matched MSR code. For the 2-layer code, we can achieve the optimal storage efficiency for given system requirements. Our comprehensive analysis shows that our code can detect and correct malicious nodes with higher storage efficiency compared to the RS-MSR code. Then we propose the m-layer code by extending the 2-layer code and achieve the optimal error correction efficiency by matching the code rate of each layer's MSR code. We also demonstrate that the optimized parameter can achieve the maximum storage capacity under the same constraint. Compared to the RS-MSR code, our code can achieve much higher error correction efficiency. The optimized m-layer code also has better error correction capability than the H-MSR code.

This dissertation is dedicated to my wife Zhang, Ying.

ACKNOWLEDGMENTS

During my Ph.D. study, Dr. Jian Ren has been an excellent advisor and mentor. I would like to thank Dr. Jian Ren for bringing me into the academic area. He not only teaches me solid knowledge in cyber security area, but also helps me establish the methodology of doing serious and meaningful research. Without his support this dissertation would not have happened.

I also would like to express gratitude to the professors in my committee: Dr. Subir Biswas, Dr. Fathi Salem, and Dr. Richard Enbody from department of Computer Science. I would not have achieved current goals without their helpful advice.

I want to thank Dr. Tongtong Li for those enlightening discussions. She has taught me so much and greatly expanded my research visions.

And I cannot imagine what it would be without the great support from my family, my lab mates and my friends. I love you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

KEY TO ABBREVIATIONS

AODV	Ad hoc On-Demand Distance Vector
BEC	Binary Erasure Channel
BPA	Belief Propagation Algorithm
CRC	Cyclic Redundancy Check
DC	Data Collector
EDEC	Error Detection and Error Correction
GF	Galois Field
H-MBR	Hermitian Code Based Minimum Bandwidth Regeneration
H-MSR	Hermitian Code Based Minimum Storage Regeneration
LDPC	Low Density Parity Check
LEDEC	LDPC Based Error Detection and Error Correction
MAC	Media Access Control
MBR	Minimum Bandwidth Regeneration
MSR	Minimum Storage Regeneration
MDS	Maximum Distance Separable
P2P	Peer to Peer
RS	Reed-Solomon
RS-MBR	Reed-Solomon Code Based Minimum Bandwidth Regeneration
RS-MSR	Reed-Solomon Code Based Minimum Storage Regeneration

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

In this dissertation, we have done researches on ensuring the network capacity in two hot research areas: network coding and distributed storage.

1.1 Combating Pollution Attacks in Network Coding

Network coding is a new communication diagram that is designed to improve the throughput and robustness in network environment. The core notation of network coding is that it allows the participating nodes to encode incoming packets at intermediate network nodes in a way that when a sink receives the packets, it can recover the original message. Network coding provides a trade-off between maximum multicast flow rate in directed networks and computational complexity. However, in the context of network coding, all participating nodes must encode the incoming packets according to a fixed coding algorithm. If a packet from an intermediate relay node is corrupted or being tampered, the entire communications may be disrupted. One main purpose of this dissertation is to develop schemes that can combat network pollution and malicious attacks from the network nodes based on error control coding.

1.1.1 Brief Review of Network Coding

Network coding was first introduced in the seminal paper by [1]. [2] formulated the multicast problem in network coding as the max-flow from the source to each receiving node. They proved that linear coding is sufficient to achieve the optimum. This work made the network coding simpler and more practical. [3] have shown that linear codes are sufficient to achieve the multicast capacity by coding on a large enough field. [4] have shown that using of random

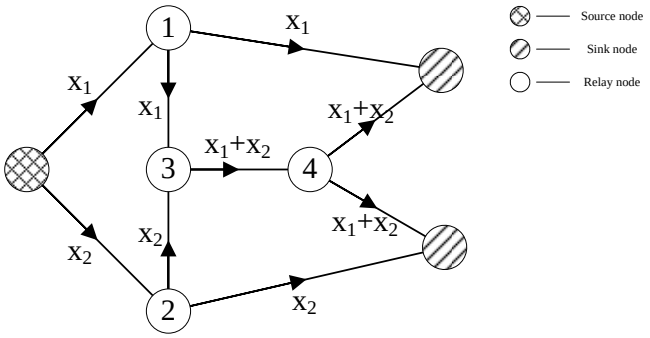

Figure 1.1 A simple example of network coding

linear network coding is a more practical way to design linear codes. [5] have applied the principles of random network coding to the context of peer-to-peer (P2P) content distribution, and have shown that file downloading times can be reduced. Since it has been proved that linear network codes are sufficient to achieve the multicast capacity, we will focus our discussion on linear network coding in this dissertation.

The main idea of network coding can be illustrated through Figure 1.1. Assume the capacity of all the edges is C, the capacity of this network is $2C$ according to the max-flow min-cut theorem. Only by encoding the incoming packet symbols x_1, x_2 at node 3, this network can achieve the maximum capacity.

1.1.2 Security Problems of Network Coding

For network coding to achieve the expected benefits, all the participating nodes in the network should be free of network pollution and malicious attacks. Suppose under the linear network coding, the sink node receives m packet symbols $y_1, \ldots, y_m$. It decodes the original message symbols $x_1, \ldots, x_l$ by solving a set of linear equations:

$$\mathbf{Bx} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1l} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{m1} & \beta_{m2} & \cdots & \beta_{ml} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}. \tag{1.1}$$

If all the relay nodes encode correctly and $m \geq l$, we can decode all the message symbols successfully. However, if there are adversaries in the network that can modify the contents of the messages and send them to the succeeding relay nodes, the equations above may not be solved successfully. So the communication fails. In addition, for a large scale network, a small error occurring at the beginning of relay may diffuse to most of the messages in the end. This can cause a significant waste of network resources and sometimes can even ruin the whole network communication. This kind of attack is called *pollution attack*.

1.1.3 Existing work on Combating Pollution Attacks in Network Coding

Existing work on pollution elimination can largely be divided into error-detection based schemes and error-correction based schemes. For error-detection based schemes, the errors are normally detected at the intermediate forward nodes, while for error-correction based schemes, the errors are generally corrected at the sink node. While the error-correction based schemes seem to be more appealing, the complexity for encoding and decoding is relatively high. It also comes with a relatively higher computational overhead. In [6–8], classic network error correction codes were studied following the work of [9]. The existence and construction of MDS network error correction codes were also studied in [10–19]. In [20], the authors proposed to use network error correction code to locate the malicious attackers. The decoding of network error correction code was studied in [21]. Another type of error-correction based schemes use rank metric codes to correct the errors in the sink nodes: [22–27]. In [28], Jaggi *et al.* developed a two-part rate-region for their codes based on BEC channel codes. [29–31] studied the theoretical network capacity under pollution attacks.

The error-detection based schemes are attractive in some network scenarios where the network topology is unknown. In [32], Zhen *et al.* proposed a probabilistic key pre-distribution and message authentication codes based scheme against pollution attacks. Their scheme is efficient in the XOR network coding environments. Krohn *et al.* proposed to use homomorphic hash functions [33] to guarantee the correctness of network flow. The main idea is

that each intermediate node will check the correctness of the packets. If a packet fails at an intermediate node verification, it will be discarded. This approach can reduce the communication overhead and can be used in random network coding. However, the computational complexity is still very high. When the network scale is large, computing too many hash values also creates high delay. Similarly, [34] used the cryptographic idea to capture and discard the corrupted packets. Other error-detection based schemes were studied in [35–43]. To address the computational limitations, [44] developed a simple error-detection based *Null Key* scheme. The main idea is to partition the n-dimensional linear space over $GF(q^n)$ into two orthogonal subspaces of dimension k (symbol subspace) and $n-k$ (null key space). Comparing to the homomorphic hash function, the Null Key scheme is much more efficient and has virtually no message delay. For all these schemes above, all corrupted packets will be discarded. In packetized networks, a large message is divided into small packets. If a malicious node can corrupt one fragment (packet) in the whole message, according to the approaches described in the error-detection based schemes, this fragment will be discarded. In this way, the net transmission efficiency can be close to zero.

There are also other approaches to improve the error resilience in network coding [45–53], including designing secure protocols, combining network codes with other codes and implementing secure network codes in specific scenarios.

1.1.4 Summary of the Limitations of existing work on Combating Pollution Attacks in Network Coding

Error-correction based schemes for fixed networks

- High complexity for encoding and decoding.
- High computational overhead.

Error-detection based schemes for random networks:

- High computational overhead and communication delay for some schemes.

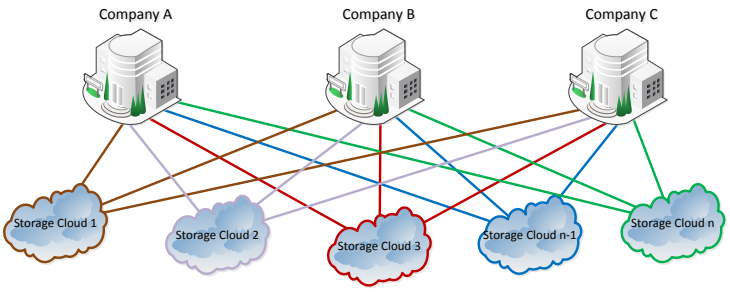

Figure 1.2 Diagram of distributed storage

- Low transmission efficiency if a malicious node continues to corrupt one message packet.

1.2 Distributed Storage in Hostile Networks

Distributed storage is an on-demand network data storage and access paradigm. The distributed data storage architecture model (in Figure 1.2) distributes the database to multiple servers in many locations across the participating network in the storage cloud. Under this model, protected data is distributed on servers in many locations across the participating network. Each location is directly plugged into the Internet. These distributed servers may even be untrusted and unreliable. If an attack is made on the data in one location, or try to jam the communications, only a small amount of backed up data is impacted. In addition, since the potential storage is dispersed to many locations, access to data does not come under the same bandwidth constraints since each location has its own pipe to the Internet. The decentralization also shrinks the footprint for data attacks, since a breach of one data center does not expose all backed up data to the attacker.

1.2.1 Brief Review of Current Algorithms for Distributed Storage

To ensure accessibility of remotely stored data at any time, a typical solution is to store the data across multiple servers or clouds, often in a replicated fashion. Data replication not only lacks flexibility in data recovery, but also requires secure data management for the stored data.

It is well known that security data management is generally very costly and very hard to defend against compromising attacks. Distributed data storage provides an elegant tradeoff between the costly secure data management task and the cheap storage media. The main idea is instead of storing the entire data in one server, we can split the data into n data components. The original data can be recovered only when the required (threshold) number of components, say k, are collected. The original data is information theoretically secure for anyone who can access either an individual component or multiple components when the number of components combined is less than the threshold k. In this case, when the individual components are stored distributively across multiple cloud storage servers, each cloud storage server only needs to assure data integrity and data availability. The costly data encryption and secure key management may no longer be needed any more. The distributed cloud storage can also increase data availability while reducing network congestion that leads to increased resiliency. A popular approach is to employ an (n, k) maximum distance separable (MDS) code such as an Reed-Solomon (RS) code [54, 55]. For RS code, the data is stored in n storage nodes in the network. The data collector (DC) can reconstruct the data by connecting to any k healthy nodes.

While RS code works perfect in reconstructing the data, it lacks scalability in repairing or regenerating a failed node. To deal with this issue, the concept of regenerating code was introduced in [56]. The main idea of the regenerating code is to allow a replacement node to connect to some individual nodes directly and regenerate a substitute of the failed node, instead of first recovering the original data then regenerating the failed component. In this way, the recovery problem of the distributed storage can be viewed as a multicasting problem which can be solved using network coding. Network coding becomes the base of the regenerating code.

Compared to the RS code, regenerating code achieves an optimal tradeoff between bandwidth and storage within the minimum storage regeneration (MSR) and the minimum bandwidth regeneration (MBR) points. RS code based MSR (RS-MSR) code and MBR (RS-

MBR) code [57] have been explicitly constructed. However, the existing research either has no error detection capability, or has the error correction capability limited by the RS code. Moreover, the schemes with error correction capability are unable to determine whether the error correction is successful.

1.2.2 Existing Work on Distributed Storage

When a storage node in the distributed storage network that employing the conventional (n, k) RS code (such as OceanStore [54] and Total Recall [55]) fails, the replacement node connects to k nodes and downloads the whole file to recover the symbols stored in the failed node. This approach is a waste of bandwidth because the whole file has to be downloaded to recover a fraction of it. To overcome this drawback, Dimakis *et al.* [56] introduced the conception of $\{n, k, d, \alpha, \beta, B\}$ regenerating code based on the network coding. In the context of regenerating code, the contents stored in a failed node can be regenerated by the replacement node through downloading γ help symbols from d helper nodes. The bandwidth consumption for the failed node regeneration could be far less than the whole file. A data collector (DC) can reconstruct the original file stored in the network by downloading α symbols from each of the k storage nodes. In [56], the authors proved that there is a tradeoff between bandwidth γ and per node storage α. They find two optimal points: minimum storage regeneration (MSR) and minimum bandwidth regeneration (MBR) points. Currently there are many literatures focusing on the optimal regenerating codes design: [58–69]. In [70, 71] the implementation of the regenerating code were studied.

The regenerating code can be divided into functional regeneration and exact regeneration. In the functional regeneration, the replacement node regenerates a new component that can functionally replace the failed component instead of being the same as the original stored component. [72] formulated the data regeneration as a multicast network coding problem and constructed functional regenerating codes. [73] implemented a random linear regenerating codes for distributed storage systems. [74] proved that by allowing data exchange among the

replacement nodes, a better tradeoff between repair bandwidth γ and per node storage α can be achieved. In the exact regeneration, the replacement node regenerates the exact symbols of a failed node. [75] proposed to reduce the regeneration bandwidth through algebraic alignment. [76] provided a code structure for exact regeneration using interference alignment technique. [57] presented optimal exact constructions of MBR codes and MSR codes under product-matrix framework. This is the first work that allows independent selection of the nodes number n in the network.

1.2.3 Existing Work on Distributed Storage in Hostile Networks

None of these works in Section 1.2.2 considered code regeneration under node corruption or adversarial manipulation attacks in hostile networks. In fact, all these schemes will fail in both regeneration and reconstruction if there are nodes in the storage cloud sending out incorrect responses to the regeneration and reconstruction requests.

In [77], the Byzantine fault tolerance of regenerating codes were studied. In [78], the authors discussed the amount of information that can be safely stored against passive eavesdropping and active adversarial attacks based on the regeneration structure. In [79], the authors proposed to add CRC codes in the regenerating code to check the integrity of the data in hostile networks. Unfortunately, the CRC checks can also be manipulated by the malicious nodes, resulting in the failure of the regeneration and reconstruction. In [80], the authors proposed to add data integrity protection in distributed storage. In [81], the authors proposed an erasure-coded distributed storage based on threshold cryptography. In [82], the authors analyzed the verification cost for both the client read and write operation in workloads with idle periods. In [83], the authors analyzed the error resilience of the RS code based regenerating code in the network with errors and erasures. They provided the theoretical error correction capability. Their result is an extension of the MDS code to the regenerating code and their scheme is unable to determine whether the errors in the network are successfully corrected.

1.2.4 Limitations of Existing Work on Distributed Storage in Hostile Networks

Existing Work on Distributed Storage in Hostile Networks has the following limitations:

- The error detection/correction capability is fixed. If there are a few errors, there will be a waste of bandwidth. If there are too many errors, the error correcting process will fail without being detected.
- The error correction capability is limited by the error correction capability of the MDS codes.

1.2.5 Difference with Existing Work on Secure Network Communication

It is worthwhile to point out that although there are strong connections between regenerating code in distributed storage and general network communication of which security problems have been well studied, our proposed H-MSR/H-MBR codes are different from these security studies of network communication e.g. [84–86] in both principles and scopes. First, unlike all the studies above, the nice error correction capability of the proposed H-MSR/H-MBR codes is due to the underlying Hermitian code [87]. Second, the regenerating codes studied in this work and the general network communication are fundamentally different in that besides the overall data reconstruction the regenerating codes also emphasize the repair of corrupted code components (regeneration), while general network communication only focuses on data reconstruction. None of the researches of general secure network communication studies the regeneration problem. The scope of this work is different from that of those researches.

1.3 Proposed Research Directions

1.3.1 Directions for Combating Pollution Attacks in Network Coding

For fixed networks, instead of designing the network codes with error correction capability, we focus on the characterizing of the network coding so that network coding can be viewed

from the perspective of error control coding. In particular, we will analyze the relationship between the network coding and the error control coding in both unicast and multicast cases. We find the algebraic aspects for these two cases are essentially identical.

After we have proven that each network coding can be transferred into an error control code in a bipartite graph by ignoring the structure of the underlying error control coding, we then transfer a network coding scheme into a series of cascaded error control codes by exploring the inner structure of network coding. This mapping enables us to identify the minimum number of independent error pattern in the corresponding network level and identify the malicious network nodes.

For random networks, we propose a new scheme that combines error-detection and error-correcting (EDEC) to combat network pollution attacks. Original message symbols are first encoded using an (n, k) code then sent out in packets. When an intermediate node detects an error, instead of discarding the packet, the intermediate node will continue to forward it. As long as the errors are within the decoding capability, the sink nodes will be able to recover the corrupted packet.In our LDPC based EDEC (LEDEC) scheme, we treat the packets as LDPC codes at sink nodes and use the belief propagation algorithm (BPA) to decode the LDPC code. In this case, the LEDEC scheme can maintain the throughput unchanged for moderate network pollution. It can also guarantee a high network throughput even for a heavily polluted network environment, while the throughput becomes very low for all error-detection based schemes in this case. In the analyses, we mainly focus on the throughput impact brought by different strategies (discard vs. keep) towards corrupted packets.

1.3.2 Directions for Distributed Storage in Hostile Networks

For distributed storage, we propose Hermitian code based regenerating codes: H-MSR code and H-MBR code. We construct the H-MSR code by combining the Hermitian code and regenerating code at the MSR point, then we construct the H-MBR code by combining the Hermitian code and regenerating code at the MBR point. Then we prove that these codes

can achieve the theoretical MSR bound and MBR bound respectively. We also propose data regeneration and reconstruction algorithms for the H-MSR code and the H-MBR code in both error-free networks and hostile networks.

Moreover, inspired by the nice performance of Hermitian code based regenerating codes, we step forward to further construct optimal regenerating codes which have similar layered structure like Hermitian code in distributed storage. We first propose a simple optimal construction of 2-layer rate-matched MSR code. We conduct both theoretical analysis and performance evaluation to show that this code can achieve the optimal storage efficiency. Then we propose an optimal construction of m-layer rate-matched MSR code. The m-layer code can achieve the optimal error correction efficiency.

1.4 Overview of the dissertation

1.4.1 Major Contributions

The major contributions of this dissertation are as follows:

- Combating pollution attacks in fixed network coding.

    1. We provide a comprehensive analysis and theoretical results on the relationships between the network coding and the error control coding.
    2. We propose a methodology to design efficient network coding schemes that can combat network errors and network pollution.
    3. We develop a methodology to map each network coding into a series of cascaded error control codes.
    4. We provide a novel approach to design efficient network coding schemes that can combat pollution attacks and locate the malicious nodes by utilizing the inner structure of the network code.

- Combating pollution attacks in random network coding.

    1. Our proposed EDEC scheme and the LEDEC scheme can maintain the throughput unimpacted for network environment with moderate malicious attacks with only a slight increase in computational overhead.
    2. The proposed scheme can guarantee a high throughput even for a heavily polluted network environment.
    3. We provide comprehensive throughput analysis of the proposed EDEC scheme and the LEDEC scheme.
    4. We conduct extensive simulations using ns-2 to evaluate the performance of the proposed schemes and compare our schemes with the error-detection based schemes.

- Distributed storage in hostile networks — Hermitian code based regenerating codes

    1. Theoretical evaluation shows that our proposed schemes can detect the erroneous decodings while other existing work cannot.
    2. Our proposed schemes can correct more errors in the hostile network than the RS-MSR/RS-MBR codes with the same code rates.
    3. Our analysis also demonstrates that the proposed H-MSR/H-MBR codes have lower complexity than the RS-MSR/RS-MBR codes in both codes regeneration and codes reconstruction.

- Distributed storage in hostile networks — Optimal construction of regenerating codes through rate-matching

    1. Our proposed optimal construction of 2-layer rate-matched MSR code can achieve the optimal storage efficiency, which is higher than the RS-MSR code proposed in [83].

2. Our proposed optimal construction of m-layer rate-matched MSR code can achieve the optimal error correction efficiency, which is higher than the code proposed in [83] and the H-MSR code proposed in [88]. Furthermore, the m-layered code is easier to understand and has more flexibility than the H-MSR code.

1.4.2 Structure

The dissertation is structured as follows.

Chapter 2 introduces the preliminary for this dissertation. Some basic concepts and properties of network coding, error control coding, regenerating code and hermitian code are presented.

Chapter 3 is mainly about combating pollution attacks for fixed networks. The first section studies the relationship between network coding and error control coding. The second section characterizes network coding using cascaded error control coding.

Chapter 4 is mainly about combating pollution attacks for random networks. The system/adversarial models and assumptions are presented in the first section. The proposed EDEC scheme and performance analysis are described in the second section. The third section presents the LDPC decoding and analysis of the LEDEC scheme.

Chapter 5 is mainly about the Hermitian code based regenerating codes in distributed storage in hostile networks. After the system/adversarial models and assumptions are presented, our proposed H-MSR code is described and analyzed in the second section. The proposed H-MBR code is described and analyzed in the third section. Performance analysis is conducted in the fourth section.

Chapter 6 is mainly about the optimal construction of regenerating code through rate-matching. The first section presents the system/adversarial models and assumptions. The second section proposes two component codes for the rate-matched MSR codes. The third section proposes and analyzes the 2-layer rate-matched MSR code. Then the fourth section proposes and analyzes the m-layer rate-matched MSR code.

Chapter 7 summarizes the dissertation.

CHAPTER 2

PRELIMINARY

In this chapter, we will present some basic concepts and properties of network coding, error control coding, regenerating code and hermitian code.

2.1 Network Coding

In this dissertation, we adopt the notations of [3]. A network is equivalent to a directed graph $G = (V, E)$, where V represents the set of vertices corresponding to the network nodes (source nodes, relay nodes and sink nodes) and E represents all the directed edges between vertices corresponding to the communication link. The start vertex v of an edge e is called the tail of e and written as $v = tail(e)$, while the end vertex u of an edge e is called the head of of e and written as $u = head(e)$. We define the capacity of an edge as the number of bits that can be transmitted through the edge in one time unit. So the capacity should be non-negative integers. In this dissertation, we normalize the capacity of one edge to 1. If a channel between two nodes has capacity C larger than 1, we model this channel as C multiple edges each with capacity 1. We assume the network is delay-free [3], that is all the edges in the graph have zero delay. And the network is acyclic, that is all the vertices in the graph can be organized in an ancestral ordering.

For a source node u, there is a set of discrete random processes to be sent. Each of the random process can be represented by a binary vector of length m, that is every symbol sequence of the random process is from the finite field $\mathcal{F}_2 m$. We write the set of the processes as $\mathcal{X}(u) = \{X(u,1), X(u,2), \cdots, X(u,\mu(u))\}$, in which $\mu(u)$ is the number of random processes in node u. Since we normalize the capacity of each edge to 1, it is reasonable to normalize the rate of the random process $X(u,i)$ $(1 \leq i \leq \mu(u))$ to 1.

We can write a link e between r_1 and r_2 as $e = (r_1, r_2)$. The random process $Y(e)$ on the link e is the function of all the $Y(e')$ from links e' (such that $head(e') = r_1$) and the random processes $\mathcal{X}(r_1)$ from node r_1. In $\mathcal{F}_{2^m}$ linear network coding [2], $Y(e)$ can be written as:

$$Y(e) = \sum_{l=1}^{\mu(r_1)} \alpha_{l,e} X(v,l) + \sum_{e':head(e')=r_1} \beta_{e',e} Y(e'), \tag{2.1}$$

in which the encoding coefficients $\alpha_{l,e}, \beta_{e',e} \in \mathcal{F}_{2^m}$.

For a sink node v, there is also a set of discrete random processes to be observed. We write the set of the processes as $\mathcal{Z}(v) = \{Z(v,1), Z(v,2), \cdots, Z(v,\lambda(v))\}$, in which $\lambda(v)$ is the number of random processes observed in node v. In linear network coding, $Z(v,j)$ can be written as:

$$Z(v,j) = \sum_{e':head(e')=v} \gamma_{e',j} Y(e'), \tag{2.2}$$

in which the encoding coefficients $\gamma_{e',j} \in \mathcal{F}_{2^m}$.

A connection between a source node u and a sink node v can be written as $C = (u, v, \mathcal{X}(u))$. From the assumptions and deductions above, the rate of this connection $R(C)$ is equal to $|\mathcal{X}(u)|$, where $|x|$ is the cardinality of the set x. As long as we can retrieve $\mathcal{X}(u)$ from $\mathcal{Z}(v)$, we say that this connection is possible. Because we apply the linear encoding in the network, we can find the system transfer matrix M between input $\underline{x}$ and output $\underline{z}$. If we write $\underline{x} = (X(u,1), X(u,2), \cdots, X(u,\mu(u)))$ and $\underline{z} = (Z(v,1), Z(v,2), \cdots, Z(v,\lambda(v)))$, we have $\underline{z} = \underline{x}M$.

In this dissertation, since we are only concerned about this relationship in each single encoding period, we simply write the random processes $X(u,i), Y(e), Z(v,j)$ as random numbers $x_{u,i}, y_e, z_{v,j}$.

2.2 Error Control Coding

In this section, we present the preliminary for error detection, which is the base of the Null Key scheme and the proposed EDEC scheme. Error correction, which is also the base of the

proposed EDEC scheme, is also presented.

2.2.1 Error Detection

Suppose the original message symbols are in the k-dimensional linear space over $GF(2^k)$. After we encode the symbols using a generating matrix $\mathbf{G}_{k\times n}$ from an (n, k) block code, the encoded codewords will form a linear subspace over $GF(2^n)$ of dimension k. So there will be another $n-k$ dimensional subspace over the n dimensional space, which is orthogonal to the codewords subspace. If we denote a valid codeword by $\mathbf{c}$ and the bases for the $n-k$ dimensional subspace by $\mathbf{h}_1, \ldots, \mathbf{h}_{n-k}$, we have $< \mathbf{c}, \mathbf{h}_i >= 0,\ 1 \leq i \leq n-k$, where $< \cdot, \cdot >$ represents the inner product.

Let $\mathbf{H}_{(n-k)\times n} = [\mathbf{h}_1, \ldots, \mathbf{h}_{n-k}]^T$, then $\mathbf{H}$ forms the parity-check matrix of the codewords and we have

$$\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}. \tag{2.3}$$

Suppose $\mathbf{r} = \mathbf{c} + \mathbf{e}$ is a received codeword, where $\mathbf{e}$ is an n-tuple error generated by a malicious node. For the received word $\mathbf{r}$, according to equation (2.3), $\mathbf{c}$ is orthogonal to $\mathbf{H}$, therefore, we have:

$$\mathbf{r} \cdot \mathbf{H}^T = (\mathbf{c} + \mathbf{e}) \cdot \mathbf{H}^T = \mathbf{c} \cdot \mathbf{H}^T + \mathbf{e} \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T. \tag{2.4}$$

For a received word $\mathbf{r}$, there are two possibilities: (i) $\mathbf{e}$ is a codeword generated by $\mathbf{G}$ but different from the original codeword. In this case, though $\mathbf{r}$ contains error, however, because $\mathbf{r} \cdot \mathbf{H}^T = 0$, the error is undetectable using conventional error control coding techniques; (ii) $\mathbf{e}$ contains a nonzero projection to the orthogonal parity check subspace, then $\mathbf{r} \cdot \mathbf{H}^T \neq 0$. In this case, we can detect that the received word contains error.

In network coding, suppose $\mathbf{c}_1, \ldots, \mathbf{c}_i$ are valid codewords, $\mathbf{c} = \sum_j \mathbf{c}_j \cdot b_j$ is a linear combination of the codewords $\mathbf{c}_1, \ldots, \mathbf{c}_i$, where b_j is the network encoding coefficients and

has the value 0 or 1. It can be easily verified that

$$\mathbf{c} \cdot \mathbf{H}^T = \sum_j \mathbf{c}_j \cdot \mathbf{H}^T \cdot b_j = \mathbf{0}. \tag{2.5}$$

Equation (2.5) is the theoretical foundation for error control coding to be used in network coding. The rows of $\mathbf{H}$ are called *Null Keys* in [44]. By checking packet symbols at every node, there is a high probability that the Null Key scheme can detect the polluted packets after a few hops of transmission. However, the 'check-and-dump' strategy may result in a very low communication efficiency under continuous network pollution and packet corruption attacks.

2.2.2 Error Correction

Equation (2.4) is called the *syndrome* of error pattern $\mathbf{e}$, denoted as $\mathbf{s}$. It is clear that $\mathbf{r}$ is a codeword if and only if $\mathbf{s} = 0$. The task of maximum likelihood decoding is to find the minimum weight error pattern $\mathbf{e}$ such that $\mathbf{r} \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T$. In this case, the received $\mathbf{r}$ is corrected to $\mathbf{r} + \mathbf{e} = \mathbf{c}$.

In linear network coding, although the packet symbols are not the original ones sent from source nodes, we can still perform error correction using equation (2.4).

Suppose $\mathbf{r}_1, \ldots, \mathbf{r}_i$ are the received codewords from i incoming edges, $\mathbf{e}$ is the error vector added to the network coding $\mathbf{r} = \sum_j \mathbf{r}_j \cdot b_j$. If the error is within the correction capability of the (n, k) code, the syndrome will still be $\mathbf{r}\cdot\mathbf{H}^T = \mathbf{e}\cdot\mathbf{H}^T + \sum_j \mathbf{r}_j \cdot \mathbf{H}^T \cdot b_j = \mathbf{e}\cdot\mathbf{H}^T + \mathbf{0} = \mathbf{e}\cdot\mathbf{H}^T$. Then we can correct the error using syndrome decoding.

In the proposed EDEC scheme, the corrupted packets detected at the intermediate nodes will not be dumped. Both the intact and corrupted packets will be gathered by the sink nodes. The sink nodes can correct the corrupted packets and have a higher communication efficiency than the error-detection based schemes.

2.2.3 Some Properties of Error Control Codes

The error detection and correction capabilities are decided by the (n, k) code structure. We can adopt appropriate error control codes according to the pollution levels of the network. In this section, some properties of error control codes will be presented.

Theorem 2.1. (Singleton bound [89]) *For an (n, k) block code with the minimum distance d, the following relationship holds: $k + d \geq n + 1$.*

The minimum distance d is defined as the minimum hamming distance for any two distinct codewords $\mathbf{x}$ and $\mathbf{y}$ of $\mathbf{C}$: $d_{min} = min\{d(\mathbf{x}, \mathbf{y}) | \forall \mathbf{x}, \mathbf{y} \in \mathbf{C}\}$, where $d(\mathbf{x}, \mathbf{y})$ is the number of positions at which the corresponding bits are different between $\mathbf{x}$ and $\mathbf{y}$. For an (n, k) block code with minimum distance d, if we delete the first $d - 1$ bits of every codewords, all the codewords are still distinct. So there are at most $2^{n-(d-1)}$ codewords. The total number of original message symbols is at most 2^k and it can not be bigger than the number of possible codewords: $2^k \leq 2^{n-(d-1)}$.

Theorem 2.2. ([89]) *For an (n, k) block code with the minimum distance d, it can detect all the $d - 1$ or less errors, or it can correct all the $\left\lfloor \frac{d-1}{2} \right\rfloor$ or less errors.*

According to the definition of minimum hamming distance, all codewords within the distance $d - 1$ or shorter of a valid codeword are invalid. So all the $d - 1$ or less errors can be detected. Suppose $\mathbf{x}$ and $\mathbf{y}$ are two valid codewords with the minimum hamming distance and $\mathbf{z}$ is a corrupted version of codeword $\mathbf{x}$ with t errors. That is $d(\mathbf{x}, \mathbf{z}) = t$. If we want to correct $\mathbf{z}$ to $\mathbf{x}$, we must have $d(\mathbf{y}, \mathbf{z}) > t$. By using the triangle inequity, we have $d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, \mathbf{z}) = d - t$. We can ensure $d(\mathbf{y}, \mathbf{z}) > t$ by making $d - t > t$. That is $2t < d$, $t_{max} = \left\lfloor \frac{d-1}{2} \right\rfloor$.

Theorem 2.3. ([89])*A linear code is capable of correcting λ or fewer errors and simultaneously detecting τ $(\tau > \lambda)$ or fewer errors if its minimum distance $d_{min} \geq \lambda + \tau + 1$.*

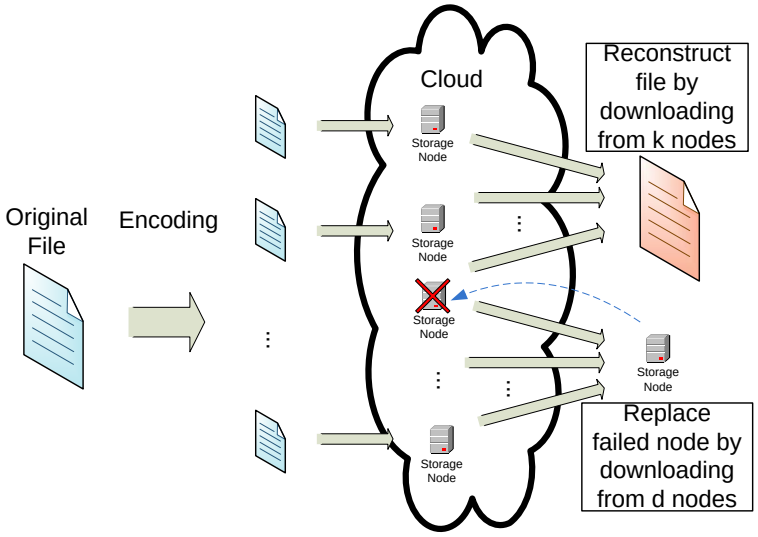

Figure 2.1 Illustration of regenerating code

Proof. Since $\tau > \lambda$, we have $d_{min} \geq \lambda + \tau + 1 > 2\lambda + 1$, $\lambda < \left\lfloor \frac{d-1}{2} \right\rfloor$. So we can correct λ or fewer errors. Suppose $\mathbf{x}$ and $\mathbf{y}$ are two valid codewords with the minimum hamming distance d_{min} and $\mathbf{z}$ is a corrupted version of codeword $\mathbf{x}$ with τ errors. In order to avoid the wrong correction, we must make sure that $d(\mathbf{y}, \mathbf{z}) > \lambda$. According to the triangle inequity, we have $d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{y}) - d(\mathbf{x}, \mathbf{z}) = d_{min} - \tau$. We can ensure $d(\mathbf{y}, \mathbf{z}) > \lambda$ by making $d_{min} - \tau > \lambda$. That is $d_{min} \geq \tau + \lambda + 1$. □

2.3 Regenerating Code

Regenerating code introduced in [56] is a linear code over GF_q with a set of parameters $\{n, k, d, \alpha, \beta, B\}$. A file of size B is stored in n storage nodes, each of which stores α symbols. A replacement node can regenerate the contents of a failed node by downloading β symbols from each of d randomly selected storage nodes. So the total bandwidth needed to regenerate a failed node is $\gamma = d\beta$. The data collector (DC) can reconstruct the whole file by downloading α symbols from each of $k \leq d$ randomly selected storage nodes. An illustration of regenerating code is shown in Figure 2.1.

In [56], the following theoretical bound was derived:

$$B \leq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\}. \tag{2.6}$$

From equation (2.6), a tradeoff between the regeneration bandwidth γ and the storage requirement α was derived. γ and α cannot be decreased at the same time. There are two special cases: minimum storage regeneration (MSR) point in which the storage parameter α is minimized;

$$(\alpha_{MSR}, \gamma_{MSR}) = \left(\frac{B}{k}, \frac{Bd}{k(d-k+1)}\right), \tag{2.7}$$

and minimum bandwidth regeneration (MBR) point in which the bandwidth γ is minimized:

$$(\alpha_{MBR}, \gamma_{MBR}) = \left(\frac{2Bd}{2kd-k^2+k}, \frac{2Bd}{2kd-k^2+k}\right). \tag{2.8}$$

2.4 Hermitian Code

A Hermitian curve $\mathcal{H}(q)$ over $GF(q^2)$ in affine coordinates is defined by:

$$\mathcal{H}(q) : y^q + y = x^{q+1}. \tag{2.9}$$

The genus of $\mathcal{H}(q)$ is $\varrho = (q^2-q)/2$ and there are q^3 points that satisfy equation (2.9), denoted as $P_{0,0}, \cdots, P_{0,q-1}, \cdots, P_{q^2-1,0}, \cdots, P_{q^2-1,q-1}$ (See Table 2.1), where $\theta_0, \theta_1, \cdots, \theta_{q-1}$ are the q solutions to $y^q + y = 0$ and ϕ is a primitive element in $GF(q^2)$. $L(mQ)$ is defined as:

$$\begin{aligned} L(mQ) &= \{f_0(x) + yf_1(x) + \cdots + y^{q-1}f_{q-1}(x)| \\ &\quad \deg f_j(x) < \kappa(j), j = 0, 1, \cdots, q-1\}, \end{aligned} \tag{2.10}$$

where

$$\kappa(j) = \max\{t|tq + j(q+1) \leq m\} + 1, \tag{2.11}$$

for $m \geq q^2-1$. A codeword of the Hermitian code [87] $\mathcal{H}_m$ is defined as $(\varrho(P_{0,0}), \cdots, \varrho(P_{0,q-1}), \cdots, \varrho(P_{q^2-1,0}), \cdots, \varrho(P_{q^2-1,q-1}))$, where $\varrho \in L(mQ)$. The dimension of the message before encoding can be calculated as $\dim(\mathcal{H}_m) = \sum_{j=0}^{j=q-1}(\deg f_j(x) + 1)$. A soft-decision list

decoding algorithm for Hermitian codes was proposed in [90]. In [87], a novel approach for decoding Hermitian codes with burst errors was proposed. Some good properties of Hermitian codes were studied in [91].

Table 2.1 $q\hat{}3$ rational points of the Hermitian curve

$P_{0,0} = (0, \theta_0)$	$P_{1,0} = (1, \phi + \theta_0)$	$\cdots$	$P_{q^2-1,0} = (\phi^{q^2-2}, \phi^{(q^2-2)(q+1)+1} + \theta_0)$
$P_{0,1} = (0, \theta_1)$	$P_{1,1} = (1, \phi + \theta_1)$	$\cdots$	$P_{q^2-1,1} = (\phi^{q^2-2}, \phi^{(q^2-2)(q+1)+1} + \theta_1)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$P_{0,q-1} = (0, \theta_{q-1})$	$P_{1,q-1} = (1, \phi + \theta_{q-1})$	$\cdots$	$P_{q^2-1,q-1} = (\phi^{q^2-2}, \phi^{(q^2-2)(q+1)+1} + \theta_{q-1})$

CHAPTER 3

COMBATING POLLUTION ATTACKS FOR FIXED NETWORKS

In this chapter, We find that network coding and error control coding are essentially identical in algebraic aspects. We will provide a novel methodology to characterize linear network coding through error control coding for fixed network coding. Our main idea is to represent each linear network coding with an error control coding. We will provide comprehensive theoretical analysis on the relationships between linear network coding and error control coding in both unicast and multicast scenarios.

Meanwhile, our research provides a new approach to understand network coding schemes and a novel methodology to develop network coding schemes that can combat node compromising attacks and locate the malicious nodes. We will characterize a linear network coding through a series of cascaded linear error control codes. This representation enables us to determine the independent source of errors in the cascaded network level. It could lead to a successful decoding of the original message and could help locating the malicious network nodes. We will provide comprehensive theoretical analysis on network coding in both unicast and multicast scenarios.

3.1 Characterization of Linear Network Coding for Pollution Attacks

3.1.1 Models and Assumptions

In this section, our main idea is to characterize and classify network coding according to the underlying error control coding. We only need to limit our consideration to linear network codes in $\mathcal{F}_2$, which makes the corresponding error control codes simple binary block codes. In this case the encoding coefficients can only be 0 or 1 and the addition operation equals

exclusive or.

3.1.2 An Illustrative Example

In this section, we will illustrate our main idea using the classic example [1] shown in Figure 3.1. In this example, source node 1 multicasts two symbols $x_{1,1}, x_{1,2}$ to sink nodes 6 and 7. By encoding at node 4, both nodes 6 and 7 can retrieve the two symbols successfully. To explain our main idea, we will only focus on the communication between node 1 and node 6 (the shaded area in Figure 3.1). The analysis is similar to the communication between node 1 and node 7. In this communication, symbol $x_{1,1}$ is passed directly through the path $e_1 - e_5$. So we can merge the edges e_1 and e_5 together in Figure 3.2(a): node 1 send $x_{1,1}$ directly to node 6 in the equivalent bipartite graph. Meanwhile, in Figure 3.2(b), $x_{1,1}$ and $x_{1,2}$ are passed separately to node 4 through $e_1 - e_3$ and $e_2 - e_4$, then $x_{1,1} + x_{1,2}$ is passed through $e_6 - e_8$ after being encoded at node 4. So we can merge the edges e_1, e_3 together, e_2, e_4 together and e_6, e_8 together in the first step of Figure 3.2(b): node 1 sends $x_{1,1}, x_{1,2}$ directly to node 4 and node 4 sends $x_{1,1} + x_{1,2}$ directly to node 6. In the second step, we can ignore node 4 and put the operation $x_{1,1} + x_{1,2}$ in node 6: node 1 sends $x_{1,1}, x_{1,2}$ directly to node 6 and node 6 adds the two symbols together in the equivalent bipartite graph.

Using the processes shown in Figure 3.2, we transfer this network coding problem into a bipartite graph shown in Figure 3.3. In this way we can get the explicit relationship between symbols of node 1 and symbols of node 6. If we view $x_{1,1}, x_{1,2}$ as original message and $z_{1,1}, z_{1,2}$ as the codeword in an error control code, we can view this network code as a $(2, 2)$ error control code with the generator matrix $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$.

Although in this example, there is no redundancy in the $(2, 2)$ error control code and this code cannot detect or correct errors, it is sufficient to show that network code can be characterized using error control code.

In the examples below, we will show that network codes with redundancies can be trans-

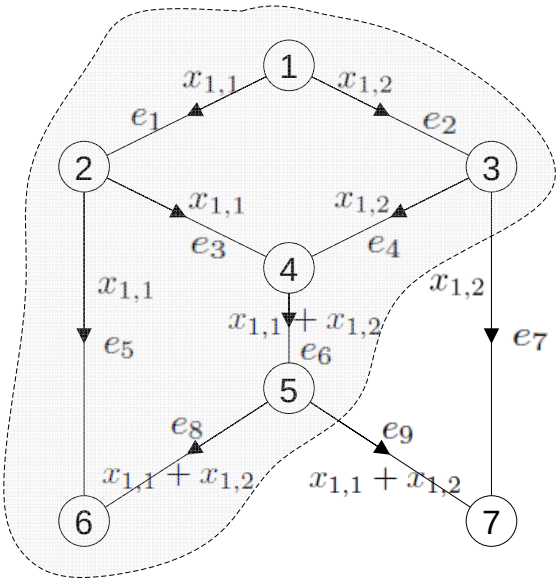

Figure 3.1 Example for illustrating the main idea

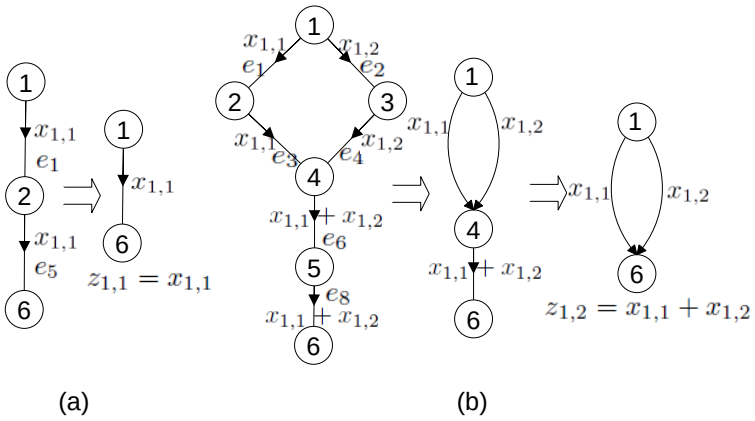

Figure 3.2 The processes of transferring network code into bipartite graph

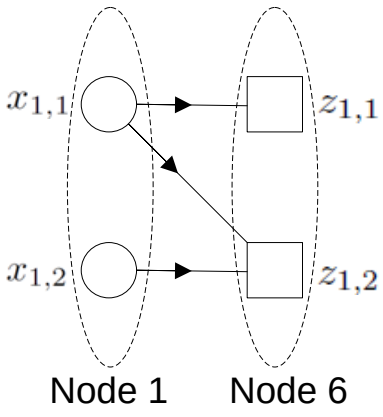

Figure 3.3 The corresponding bipartite graph of Figure 3.1

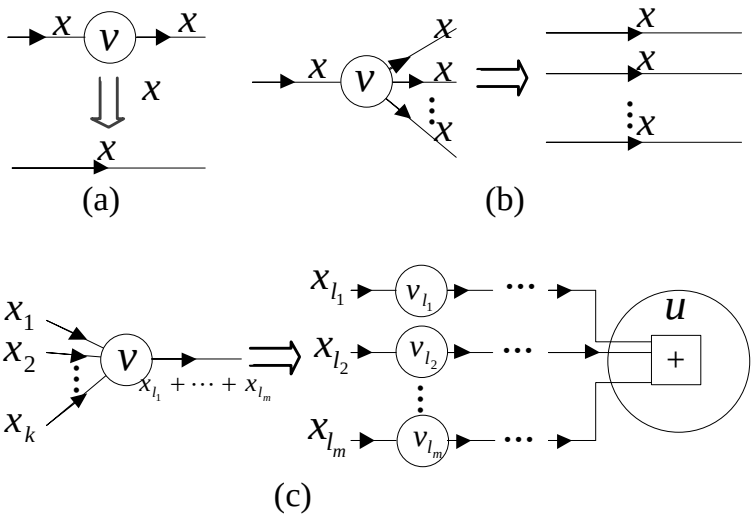

Figure 3.4 Equivalence of three kinds of nodes in network coding

ferred into error control codes. Meanwhile, the redundancies of network codes can be added according to the error control codes. And we can characterize network coding using error control coding.

3.1.3 Relationship between Network Coding and Error Control Coding in Point-to-Point Communication

In this part, we will formally state the relationship between network coding and error control coding in the point-to-point communication. The sufficiency is studied first then the necessity.

3.1.3.1 The Sufficiency

Theorem 3.1. *Every network code scheme can be represented by an error control code.*

Proof. All the nodes in network coding can be categorized into three types: simple forward, multicast and code then forward (shown in Figure 3.4). So we can transfer the graph representing the network coding using three operations accordingly:

1. **Simple forward**: These nodes do not encode the incoming symbols. They simply

forward whatever message they have received. In this situation, we can replace the nodes with direct links.

2. **Multicast**: Like simple forward, these nodes do not encode the incoming symbols either. They simply multicast the message that they have received. In this situation, we can replace the nodes with multiple direct links.

3. **Code then forward**: These nodes produce the linear combination of the incoming symbols $x_1, x_2, \cdots, x_k$. According to the encoding coefficients, m out of k received symbols will be added together to form the new symbol $x_{l_1} + x_{l_2} + \cdots + x_{l_m}$ to be forwarded. We can view this kind of node as m parallel nodes $v_{l_1}, v_{l_2}, \cdots, v_{l_m}$, each of which has only one input. The symbols $x_{l_1}, x_{l_2}, \cdots, x_{l_m}$ will be directly forwarded to the sink node u. And the sink node will complete the addition operation. Therefore we can transfer code then forward nodes by splitting multiple inputs into multiple simple forward nodes. Then we can further simplify the multiple simple forward nodes as in '1)'.

Because the network coding is linear, the three operations are commutative and have the superposition property. We can always perform the operations to all of the intermediate nodes in the network and replace the nodes with simple links. At last we can get a bipartite graph consisting of only symbols in the source node and encoded symbols in the sink node, which can be represented using an error control code corresponding to the bipartite graph. □

Take the network code in Figure 3.5 as an example. The source node 1 transmits three symbols $x_{1,1}, x_{1,2}, x_{1,3}$ to sink node 4 in this network code. And sink node 4 can receive 6 encoded symbols, which indicates that there are redundancies in this network coding. Following the operations mentioned in the proof of Theorem 3.1, we can get the corresponding bipartite graph shown in Figure 3.6, which indicates this is a $(6, 3)$ error control code. The

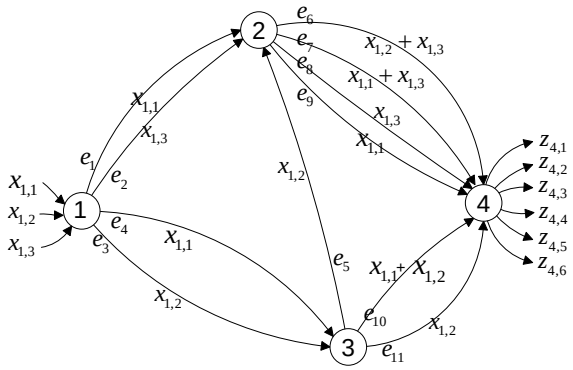

Figure 3.5 An example of point-to-point network coding

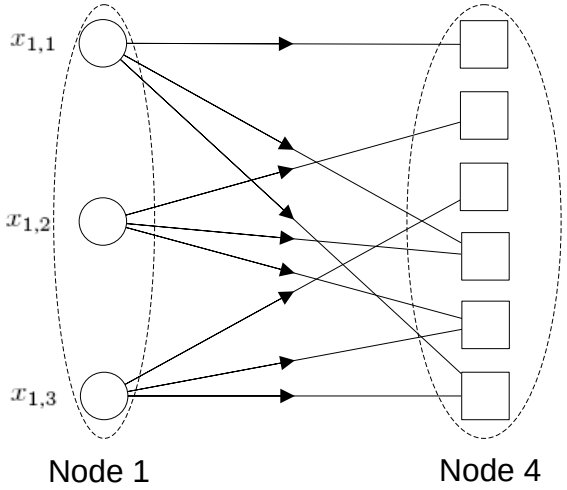

Figure 3.6 The corresponding bipartite graph of Figure 3.5

generator matrix is:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{3.1}$$

This code has a minimum hamming distance 3. So it can detect and correct 1 bit error.

The sufficiency can also be validated by the system transfer matrix M. To show this, we will still use the network topology shown in Figure 3.5, but with different encoding

coefficients. The symbols on each edge can be written as:

$$\begin{aligned}
y_{e_1} &= \alpha_{1,e_1}x_{1,1} + \alpha_{2,e_1}x_{1,2} + \alpha_{3,e_1}x_{1,3} \\
y_{e_2} &= \alpha_{1,e_2}x_{1,1} + \alpha_{2,e_2}x_{1,2} + \alpha_{3,e_2}x_{1,3} \\
y_{e_3} &= \alpha_{1,e_3}x_{1,1} + \alpha_{2,e_3}x_{1,2} + \alpha_{3,e_3}x_{1,3} \\
y_{e_4} &= \alpha_{1,e_4}x_{1,1} + \alpha_{2,e_4}x_{1,2} + \alpha_{3,e_4}x_{1,3} \\
y_{e_5} &= \beta_{e_3,e_5}y_{e_3} + \beta_{e_4,e_5}y_{e_4} \\
y_{e_6} &= \beta_{e_1,e_6}y_{e_1} + \beta_{e_2,e_6}y_{e_2} + \beta_{e_5,e_6}y_{e_5} \\
y_{e_7} &= \beta_{e_1,e_7}y_{e_1} + \beta_{e_2,e_7}y_{e_2} + \beta_{e_5,e_7}y_{e_5} \\
y_{e_8} &= \beta_{e_1,e_8}y_{e_1} + \beta_{e_2,e_8}y_{e_2} + \beta_{e_5,e_8}y_{e_5} \\
y_{e_9} &= \beta_{e_1,e_9}y_{e_1} + \beta_{e_2,e_9}y_{e_2} + \beta_{e_5,e_9}y_{e_5} \\
y_{e_{10}} &= \beta_{e_3,e_{10}}y_{e_3} + \beta_{e_4,e_{10}}y_{e_4} \\
y_{e_{11}} &= \beta_{e_3,e_{11}}y_{e_3} + \beta_{e_4,e_{11}}y_{e_4}.
\end{aligned} \tag{3.2}$$

The symbols at the sink node can be written as:

$$z_{4,j} = \sum_{i=6}^{i=11} \gamma_{e_i,j}y_{e_i}, (1 \le j \le 6). \tag{3.3}$$

Define matrices A, B as in [3]:

$$A = \begin{bmatrix} \alpha_{1,e_1} & \alpha_{1,e_2} & \alpha_{1,e_3} & \alpha_{1,e_4} \\ \alpha_{2,e_1} & \alpha_{2,e_2} & \alpha_{2,e_3} & \alpha_{2,e_4} \\ \alpha_{3,e_1} & \alpha_{3,e_2} & \alpha_{3,e_3} & \alpha_{3,e_4} \end{bmatrix}, \tag{3.4}$$

$$B = \begin{bmatrix} \gamma_{1,e_1} & \cdots & \gamma_{1,e_6} \\ \vdots & \ddots & \vdots \\ \gamma_{6,e_1} & \cdots & \gamma_{6,e_6} \end{bmatrix}. \tag{3.5}$$

We also define a matrix

$$\Gamma = \begin{bmatrix} \beta_{1,6} & \beta_{1,7} & \beta_{1,8} & \beta_{1,9} & 0 & 0 \\ \beta_{2,6} & \beta_{2,7} & \beta_{2,8} & \beta_{2,9} & 0 & 0 \\ \beta_{5,6}\beta_{3,5} & \beta_{5,7}\beta_{3,5} & \beta_{5,8}\beta_{3,5} & \beta_{5,9}\beta_{3,5} & \beta_{3,10} & \beta_{3,11} \\ \beta_{5,6}\beta_{4,5} & \beta_{5,7}\beta_{4,5} & \beta_{5,8}\beta_{4,5} & \beta_{5,9}\beta_{4,5} & \beta_{4,10} & \beta_{4,11} \end{bmatrix}, \tag{3.6}$$

here β_{e_i,e_j} is written as $\beta_{i,j}$ for short. Then the system matrix is:

$$M = A \cdot \Gamma \cdot B^T. \tag{3.7}$$

In this example, the sizes of matrices A, B are 3×4 and 6×6. Thus the size of transfer matrix is 3×6. The original symbols $x_{1,1}, x_{1,1}, x_{1,3}$ can be seen as an original message of length 3. And the received symbols at sink node can be seen as a codeword of length 6. It is appropriate that we identify the 3×6 transfer matrix M with the generator matrix G of a $(6, 3)$ error control code. Hence Theorem 3.1 is verified from the perspective of transfer matrix.

3.1.3.2 The Necessity

We have proved that any network code can be viewed as an error control code, now we will consider the reverse problem. For a point-to-point communication, a network code is feasible only if it can successfully deliver all the desired symbols from the source node to the sink node. Theorem 3.2 shows the criterion of a feasible network code.

Theorem 3.2. *For a linear network with source node u, sink node v and a desired connection $C = (u, v, \mathcal{X}(u))$, the point-to-point connection C is possible if and only if the determinant of the $R(C) \times R(C)$ transfer matrix M is nonzero.*

The proof of this theorem can be found in [3].

Since there are no redundancies in the network code in Theorem 3.2, the dimension of symbols in source node is $R(C) = |\mathcal{X}(u))|$, and the dimension of received symbols in sink node is also exactly $R(C)$. Therefore, the size of the transfer matrix M is $R(C) \times R(C)$.

Theorem 3.3. *For a linear network with source node u, sink node v and a desired connection* $C = (u, v, \mathcal{X}(u))$*, A (n,k) error control code with the* $k \times n$ *generator matrix G can be seen as a feasible network code in the point-to-point connection C if we have the relationship:* $k \geq R(C)$.

Proof. If the theorem is true for $k = R(C)$, it will also work for $k > R(C)$. It is straightforward that a network code can successfully complete the point-to-point connection of which the rate is lower than the code's maximum capacity. So we only need to prove the case when $k = R(C)$.

From the statements in preliminary section, we have $k \leq n$ for a (n, k) error control code. For a message sequences $(x_1, x_2, \cdots, x_k)$, we have encode it as follows:

$$(z_1, z_2, \cdots, z_n) = (x_1, x_2, \cdots, x_k)G. \tag{3.8}$$

Because $k \leq n$, we can choose k independent columns $(l_1, l_2, \cdots, l_k)$ from G to form a new matrix G', which has the relationship

$$(z_{l_1}, z_{l_2}, \cdots, z_{l_k}) = (x_1, x_2, \cdots, x_k)G'. \tag{3.9}$$

In this case, G' is a $k \times k$ full rank matrix with nonzero determinant. If we view $(x_1, x_2, \cdots, x_k)$ as symbols at source node u with the rate $R(C) = k$ and $(z_{l_1}, z_{l_2}, \cdots, z_{l_k})$ as symbols received at sink node v, then G' is the transfer matrix of the network code. According to Theorem 3.2, this point-to-point connection C with rate $R(C) = k$ is possible. So in this case, the (n, k) error control code can be seen as a feasible network code. □

In the case that the size of transfer matrix is larger than $R(C) \times R(C)$, the represented error control code will have redundancies which can be used to control errors. However, based on our analysis, we can add redundancies appropriately so that the network code is capable of detecting and correcting errors. This can be done in two steps:

1. According to the communication channel and the design requirements (number of errors to detect or correct, bit error rate, etc.), determine an appropriate code rate k/n and the type of the error control code (Hamming code, Cyclic code, etc).

2. According to the source rate $R(C)$, choose proper k such that $k \geq R(C)$ and n, and derive the corresponding generator matrix G. Then apply the generator matrix G as the system transfer matrix to the network coding.

3.1.3.3 Application in Combating pollution attacks

For example, in a linear network shown in Figure 3.7, the source node is going to send 4 symbols $x_1, x_2, \cdots, x_4$ to sink node. According to Theorem 3.3, we can apply the $(7, 4)$ Hamming code with generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{3.10}$$

The corresponding network code is also shown in Figure 3.7. Because the minimum distance of the code is 3, this network code can correct 1 bit error. Suppose the source node sends 4 symbols $(1, 0, 1, 0)$, the expected received symbols will be $(1, 0, 1, 0, 0, 0, 0)$. However, because the malicious node M changes the symbol, the received symbols will be $s = (1, 0, 1, 0, 1, 0, 0)$. Sink node can decode the received symbol using the syndrome-decoding method. The parity-check matrix in sink node is

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

With the syndrome of the received symbols calculated as $s \cdot H^T = (0, 0, 1)$, the sink node can find the error pattern $(0, 0, 0, 0, 1, 0, 0)$ and correct the erroneous symbol. From the location

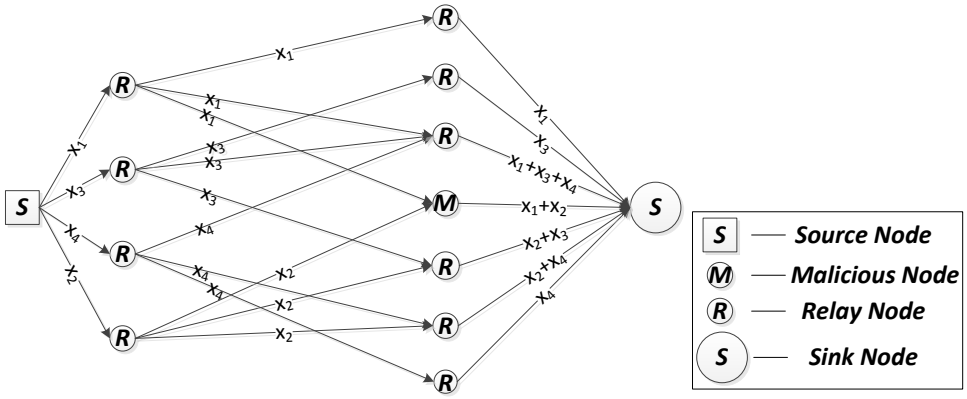

Figure 3.7 Implement the $(7, 4)$ Hamming code in network coding

of the erroneous symbol, the sink node can also find the malicious node from which $x_1 + x_2$ is transmitted.

From this example, we can see that with proper design of error control code, we can make the corresponding network code capable of detecting and correcting errors then finding the malicious nodes.

3.1.4 Multicast Case

In previous section, we study the relationship between network coding and error control coding in point-to-point communication case. We can derive similar results for the multicast case, where the network consists of one source node u and several sink nodes $v_1, v_2, \cdots, v_N$. The network code for multicasting is feasible if and only if all the sink nodes can receive all symbols $\mathcal{X}(u)$ sent from source node u.

3.1.4.1 The Sufficiency

Theorem 3.1 still holds in multicast case because we do not specify whether the communication is unicast or multicast in the proof of the theorem, which indicates the proof of the theorem is independent of the type of the communications.

3.1.4.2 The Necessity

The multicast problem can be divided into N unicast problems:

$$\mathcal{C} = (C_1, C_2, \cdots, C_N) = ((u, v_1, \mathcal{X}(u)), (u, v_2, \mathcal{X}(u)), \cdots, (u, v_N, \mathcal{X}(u))). \tag{3.12}$$

If we write all the received symbols together:

$$\underline{z} = (z_1, z_2, \cdots, z_N) = (Z(v_1, 1), \cdots, Z(v_1, \lambda(v_1)), \cdots, Z(v_N, \lambda(v_N))), \tag{3.13}$$

we can obtain the system transfer equation for the whole network: $\underline{z} = \underline{x}M$, in which M is a matrix defined as

$$M = |\mathcal{X}(u)| \times \sum_{i=1}^{i=N} \lambda(v_i). \tag{3.14}$$

It is obvious that

$$M = \Big[M_1 \quad | \quad M_2 \quad | \quad \cdots \quad | \quad M_N \Big], \tag{3.15}$$

in which $M_1, M_2, \cdots, M_N$ are the system transfer matrixes for each unicast $C_1, C_2, \cdots, C_N$.

Theorem 3.4. *For a linear network with source node u, sink node $v_1, v_2, \cdots, v_N$ and desired connections $C_i = (u, v_i, \mathcal{X}(u))(1 \leq i \leq N)$, A concatenation of N error control codes with the $k \times n_i$ generator matrix G_i can be seen as a feasible network code in the multicast problem $\mathcal{C} = (C_1, C_2, \cdots, C_N)$ if we have the relationship: $k \geq R(C_i)$.*

Proof. This theorem is a natural extension of Theorem 3.3. If $k \geq R(C_i)$, all the generator matrix G_i can be seen as feasible network codes in unicast problem C_i. So the concatenation of G_i can be seen as a feasible network code in multicast problem $\mathcal{C}$. □

3.2 A Cascaded Error Control Coding Approach

3.2.1 Models and Assumptions

If a relay node r is compromised, the symbols transmitted on each edge e such that $head(e) = r$ will be modified. The nodes after node r will be polluted because of the network encoding.

Eventually the sink node will receive more erroneous symbols than those originally brought by the malicious node. In this section, we try to explore the inner structure of the network code to correct the errors and locate the malicious node. We will partition the network into several cascaded levels and explore the inner structure of the network code, thus we must be able to correctly access the outputs of all the relay nodes. To realize this, we add a special monitor node in the network. This node can collect the output encoded messages from all the relay nodes and can never be compromised.

3.2.2 An Illustrative Example

Let us examine the classic example [1] again shown in Figure 3.1. By encoding at node 4, both nodes 6 and 7 can retrieve the two symbols $x_{1,1}, x_{1,2}$ successfully. In Section 3.1, we merged the intermediate nodes and paths and transferred the the network code into a bipartite graph. While in this section, we try to explore the network code to exhibit the inner structure of the network code. To explain our main idea, we will only focus on the communication between node 1 and node 6 (the shaded area in Figure 3.1). The analysis is similar to the communication between node 1 and node 7. In this communication, symbol $x_{1,1}$ is passed to node 2, node 4, node 5 and node 6 through one hop, two hops, three hops and two hops respectively, and symbol $x_{1,2}$ is passed to node 3, node 4, node 5 and node 6 through one hop, two hops, three hops and four hops respectively. As shown in Figure 3.8, if we add two virtual nodes v_1 and v_2 on edge e_5, we can make $x_{1,1}$ passed to node 6 through four hops, thus turn all of the intermediate nodes into 3 cascaded levels. Each of the level can be seen as a single network code, so we can represent each level using the bipartite graph shown in Figure 3.9 according to [92]. In this way, we explore the inner structure of the original network code, which is determined by the network topology. The original network code can be viewed as 3 cascaded error control codes with the generator matrices

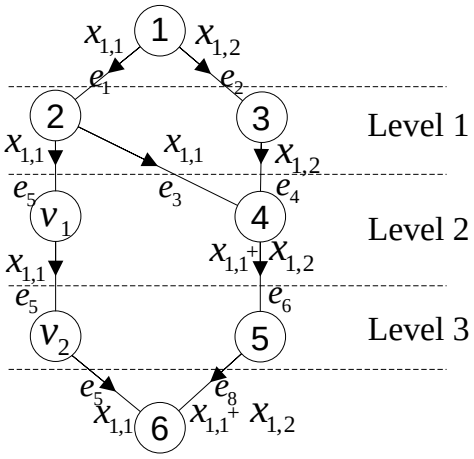

Figure 3.8 Transfer the network coding scheme in Figure 3.1 into a 3-level cascaded coding by adding 2 virtual nodes.

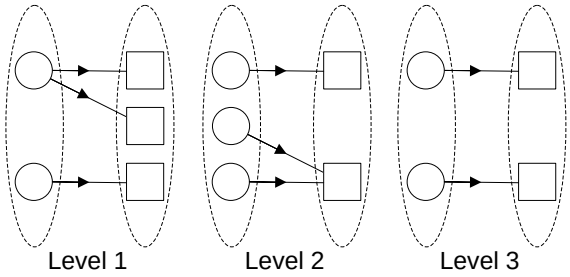

Figure 3.9 The corresponding bipartite graphs of 3 cascaded levels in Figure 3.8

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Although in this example, there is no redundancy in the three error control codes, the corresponding network code cannot detect or correct errors, it is sufficient to show that network code can be expanded to cascaded error control codes.

In the rest of Section 3.2, we will show that network codes can be transferred into cascaded error control codes. In this way, we can characterize and design network codes based on the underlying cascaded error control codes for error detection/correction and malicious nodes locating.

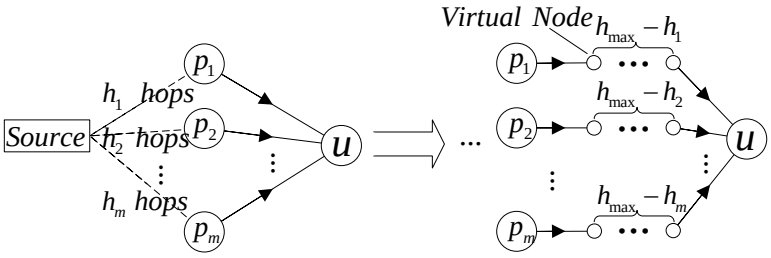

Figure 3.10 Transfer incoming edges of nodes having multiple incoming edges by adding virtual nodes

3.2.3 Characterization of Network Coding using Cascaded Error Control Coding in Point-to-Point Communication

Here we will formally state the relationship between network coding and cascaded error control coding in the point-to-point communication. The sufficiency is studied first then the necessity.

3.2.3.1 The Sufficiency

Theorem 3.5. *Every network code scheme can be expanded to a series of cascaded error control codes.*

Proof. To prove this, we will first show that the network code can be partitioned into several cascaded levels of one hop network codes. For each of the nodes that have multiple incoming edges in the network, we add some virtual nodes on these edges as shown in Figure 3.10. For each of the incoming edges, there may be several paths through which messages are passed from the source node to node u including the edge. Among all the paths, we find the longest one and calculate its number of hops. After calculating the hop values $h_1, \ldots, h_m$ for all the incoming edges, we choose the maximum value h_{max}. For each of the incoming edge i, we add $h_{max} - h_i$ virtual nodes on it, making all the paths from source to node u have the same count of hops. The virtual nodes simply forward the messages passed on the corresponding edges.

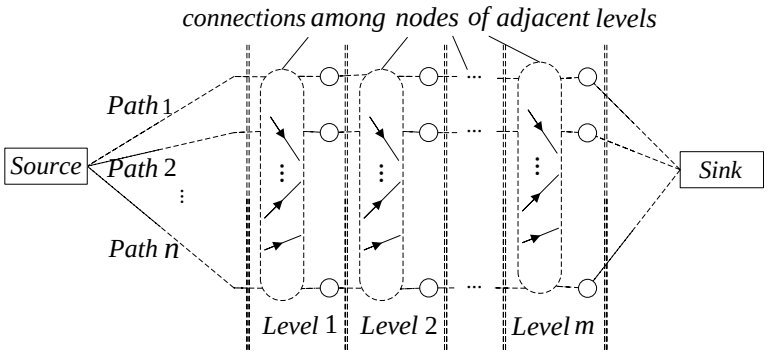

Figure 3.11 Partition a network code into several levels

After the operation in Figure 3.10 is performed in all the nodes having multiple incoming edges, since all the paths from source node to the same the relay node have the same hop counts and the sink node itself must have multiple incoming edges, every path from the source node to the sink node has the same number of hops, thus the same number of intermediate nodes, including the relay nodes and the virtual nodes. We can put the nodes having the same hop counts together as a level as shown in Figure 3.11. Every single level can be viewed as one hop network code determined by the connections from nodes of the previous level. So every network code can be partitioned into several cascaded levels of one hop network code.

According to Theorem 3.1, these one hop network codes can be represented by error control codes. So the cascaded network codes can be represented by concatenating the corresponding error control codes together. We can expand any network code to a series of cascaded error control codes. □

Taking the network code in Figure 3.5 as an example. The source node 1 transmits three symbols $x_{1,1}, x_{1,2}, x_{1,3}$ to sink node 4 in this network code. And sink node 4 can receive 6 encoded symbols, which indicates that there are redundancies in this network coding. In Section 3.1, we analyze the same code and transfer it into a $(6, 3)$ error control code which can correct 1 error. Here we will show this code can be transferred into a series of cascaded error control codes. Following the operations mentioned in the proof of Theorem 3.5, we can get the corresponding cascaded network codes and cascaded error control codes shown

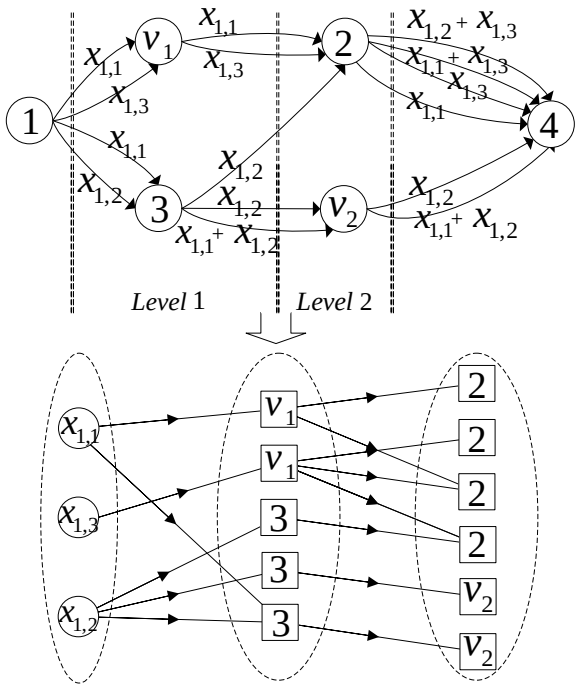

Figure 3.12 The corresponding cascaded bipartite graph of Figure 3.5

in Figure 3.12. Nodes v_1 and v_2 are added as virtual nodes to partition the original network code. The first level error control code is a $(5, 3)$ code and the second level code is a $(6, 5)$ code.

If an error occurs on edge e_1, node 2 will receive wrong $x_{1,1}$. The error will propagate to the succeeding nodes, thus there will be two erroneous $x_{1,1}$ and $x_{1,1} + x_{1,3}$ in the sink node, which is beyond the error correction capability of the $(6, 3)$ error control code. The errors cannot be dealt using the transforming methods in Section 3.1. However, if the monitor node can collect the output symbols of the first level $(5, 3)$ code, it can correct the erroneous symbol $x_{1,1}$ in node 2. So the error propagation is eliminated from the beginning. By exploring the inner structure of the network code, we can make better use of the redundancy in the network.

If node 3 is an malicious node and send out corrupted messages, there will be 3 errors in the output of both the first level error control code and the second level. The error is beyond the capability of the cascaded error control codes, so we cannot correct errors or

locate the malicious node. We will show that we can design network codes corresponding to proper cascaded error control codes to correct errors and locate malicious nodes in the sections below.

3.2.3.2 The Necessity

We have proved that any network code can be viewed as a series of cascaded error control codes, now we will consider the reverse problem. For a point-to-point communication, a network code is feasible only if it can successfully deliver all the desired symbols from the source node to the sink node.

Theorem 3.6. *For a linear network and a desired connection $C = (u, v, \mathcal{X}(u))$, A series of cascaded error control codes with parameters $(n_1, n_0), (n_2, n_1), \ldots, (n_m, n_{m-1})$, can be seen as a feasible network code in the connection C if we have the relationship: $n_0 \geq R(C)$.*

Proof. Suppose the original message is $\mathbf{x} = (x_1, \ldots, x_k)$, the output encoded message for each level of the cascaded error control codes is $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n_i})(1 \leq i \leq m)$ and the generator matrix for each of the cascaded error control codes is $G_i(1 \leq i \leq m)$ of the size $n_{i-1} \times n_i$. $\mathbf{y}_i$ for each level can be written as:

$$\mathbf{y}_1 = \mathbf{x} \cdot G_1, \ \mathbf{y}_2 = \mathbf{y}_1 \cdot G_2, \ \ldots, \ \mathbf{y}_m = \mathbf{y}_{m-1} \cdot G_m. \tag{3.16}$$

So the entire encoding equation for the cascaded error control codes can be written as

$$\mathbf{y}_m = \mathbf{x} \cdot G_1 \cdot G_2 \cdot \cdots \cdot G_m = \mathbf{x} \cdot G. \tag{3.17}$$

If we view the cascaded error control codes as an error control code with the generator matrix G of the size $n_0 \times n_m$, the parameter for the code is (n_m, n_0). According to Theorem 3.3, if $n_0 \geq R(C)$, the network code is feasible. □

Based on the analysis, by implementing the error control code for each level of the cascaded error control codes, we can add appropriate redundancies into the network code to control errors and locate malicious nodes. This can be done in two steps:

1. According to the network topology, determine the number of levels of the cascaded codes. According to the design requirements (number of errors to detect or correct, number of malicious nodes to locate), determine an appropriate code rate and the type of the error control code for each level.

2. According to the source rate $R(C)$, choose a proper n_0 such that $n_0 \geq R(C)$, and derive the rest of the $n_i (1 \leq i \leq m)$ based on the code rate for each level of the error control codes. Generate the generator matrices $G_1, \ldots, G_m$ according to the code types and apply them as the system transfer matrices to each level of the network codes.

3.2.3.3 Application in Combating pollution attacks

Theorem 3.7. *Suppose $d_i, d_{i+1} > 2$ are the minimum distances of 2 adjacent levels (L_i, L_{i+1}) of the cascaded network code. If $2d_i + 1 > d_{i+1}$, then errors in L_{i+1} spread by a single error in L_i is uncorrectable by the L_{i+1}'s error control code. However, they can be corrected by the L_i's error control code.*

Proof. According to [89], one symbol in the source message is related to at least $d_{\min}$ symbols in the encoded codeword. So one error in L_i can become at least d_i errors in L_{i+1}. If $2d_i + 1 > d_{i+1}$, these errors are beyond the capability of the error control code. However, the single error can be corrected at L_i because $d_i > 2$. Then the errors in L_{i+1} can be corrected accordingly. □

Let us analyze the linear network shown in Figure 3.13, the source node 1 is going to send 3 symbols x_1, x_2, x_3 to sink node 12. This network can be partitioned into 2 levels. Nodes $2, 3, 4, 5$ form the first level and nodes $6, 7, 8, 9, 10, 11$ form the second level. In order to get the best error control capability, we implement two systematic RS codes in the two levels. They are $(7, 3, 5)$ code for level 1 and $(11, 7, 5)$ code for level 2. The minimum distances of the two codes are both 5, thus both of them can correct 2 errors. Because the errors occurring next to the source node are more sensitive. They may propagate to the

subsequent nodes causing much more errors. We put the lower rate code that has stronger error control capability at the first level.

When there is no error in the network, we have $(y_{i,1}, y_{i,2}, y_{i,3}) = (x_1, x_2, x_3), i = 1, 2$. It is easy for the sink node to decode the messages. If node 6 is a malicious node and it sends out erroneous $y_{2,1}, y_{2,2}$, the monitor node can correct these 2 errors using the second level RS code and find out this malicious node according to the network topology. If node 2 is a malicious node and it sends out erroneous $y_{1,1}, y_{1,2}$, the errors will propagate to $y_{2,1}, y_{2,2}, y_{2,8}, y_{2,9}, y_{2,10}, y_{2,11}$, which prevents the second level code from correcting the errors. In the corresponding cascaded bipartite graph Figure 3.14, the errors are marked with grey color. It is clear that 2 errors from level 1 spread to 6 errors in level 2. Even if we transfer the network code into one $(11, 3, 9)$ RS code which is capable of correcting 4 errors according to Section 3.1, the errors are still too many to correct. However, based on the fact that the errors are burst and correlated, after the monitor node collects the outputs of the first level, it can correct the 2 errors occurring in node 2 using the first level RS code, find out the malicious node based on the network topology and correct the 6 errors in the second level. Our cascaded RS code can correct at most 6 errors by exploring the inner structure of the code and is more powerful than regular RS codes.

With proper design of each level of the cascaded error control codes, we can make the corresponding network code capable of detecting and correcting errors then locating the malicious nodes.

3.2.4 Multicast Case

Because in point-to-point communication case, our proofs for the relationship (written as $\mathcal{R}_{nc,cec}$) between network code and cascaded error control codes are solely depended on the proofs for the relationship (written as $\mathcal{R}_{nc,ec}$) between network code and error control code in Section 3.1 (Theorem 3.1 and Theorem 3.3) and this kind of dependence has no relationship with the specific communication case, we can prove that $\mathcal{R}_{nc,cec}$ in the multicast

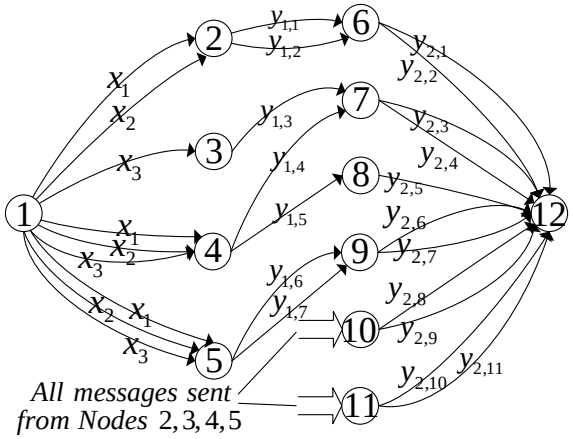

Figure 3.13 Implement a 2 level cascaded error control code in network coding

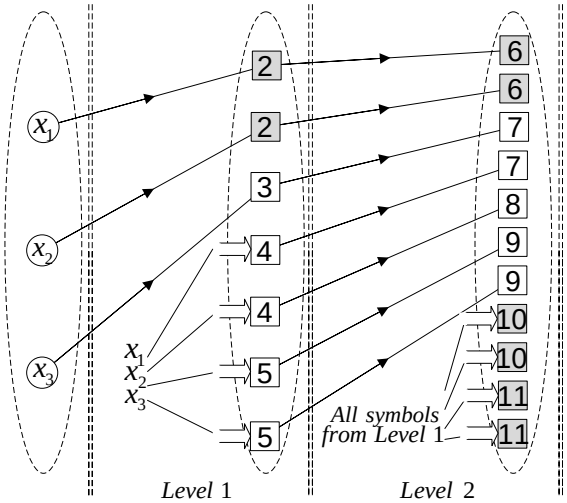

Figure 3.14 The corresponding cascaded bipartite graph of Figure 3.13

case is similar to that in the point-to-point communication case, based on the fact that in Section 3.1 $\mathcal{R}_{nc,ec}$ stays the same in both point-to-point and multicast cases.

CHAPTER 4

COMBATING POLLUTION ATTACKS FOR RANDOM NETWORKS

In this chapter, we will propose a new error-detection and error-correction (EDEC) scheme to detect and remove the malicious attacks for random network coding. The proposed EDEC scheme is similar in structure to the existing error control based schemes. However, it can maintain throughput unchanged when moderate network pollution exists with only a slight increase in computational overhead. Then we propose an improved LEDEC scheme by integrating the low-density parity check (LDPC) decoding. Our theoretical analysis demonstrates that the LEDEC scheme can guarantee a high throughput even for heavily polluted network environment. We also provide extensive performance evaluation and simulation results to validate our theoretical results using ns-2 network simulator.

4.1 System/Adversarial Models and Assumptions

In this chapter, we will study combating pollution attacks for the encoding for random networks, where it is difficult to design efficient error correction network codes without the knowledge of the network topology. In this case, all the encoding coefficients $\alpha_{l,e}, \beta_{e',e}$ and $\gamma_{e',j}$ will be chosen randomly.

In this chapter, we will use some simplified notations of Section 2.1. For a source node u, there is a set of symbols $\mathcal{X}(u) = (x_1, \ldots, x_l)$ to be sent. For a link e between relay nodes r_1 and r_2, written as $e = (r_1, r_2)$, the symbol y_e transmitted on it is the function of all the $y_{e'}$ such that $head(e') = r_1$. And y_e can be written as:

$$y_e = \sum_{e':head(e')=r_1} \beta_{e',e} \cdot y_{e'} = \sum_{i=1}^{l} \beta_{e,i} x_i = \boldsymbol{\beta}_e \mathbf{x}, \tag{4.1}$$

where $\beta_{e',e}$ is the local network encoding coefficient, $\beta_{e,i}$ is the global network encoding

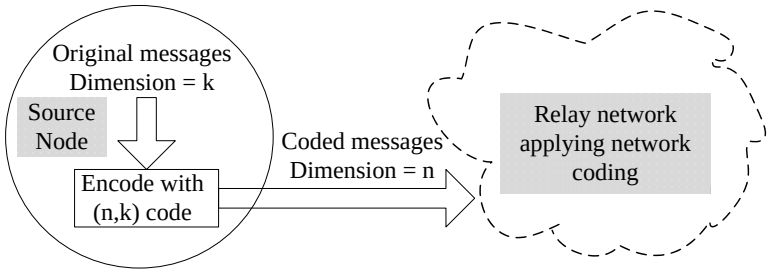

Figure 4.1 Apply error control codes in linear network coding

coefficient for symbol y_e and $\boldsymbol{\beta}_e = \left[\beta_{e,1}, \beta_{e,2}, \ldots, \beta_{e,l}\right]$ is the network encoding vector. For a sink node v, there is a set of incoming symbols $y_{e'}$ $(e' : tail(e') = v)$ to be decoded.

As we mentioned above, if adversaries can modify the contents of the packets and send them to the succeeding relay nodes, the communication will fail and the capacity will be reduced. In addition, for a large scale network, a small error occurring at an intermediate relay node may diffuse to many packets at the sink node. This can cause a significant waste of network resources and sometimes can even ruin the whole network communication.

In this dissertation, the malicious node can add random errors to the symbols in the received packets then send the corrupted packets out to pollute the network. We adopt this simple adversarial model because we mainly focus on the throughput impact brought by different strategies (discard vs. keep) towards corrupted packets in this research.

4.2 Proposed EDEC Scheme

The basic idea of the proposed EDEC scheme in Figure 4.1 is that the source nodes encode the original messages using an error control code before sending them out. The properties of the error control code keep unchanged during the linear network coding.

As we mentioned in Section 1.1.3, the error-detection based schemes mainly focus on detecting the corrupt packets. When a corrupt packet is identified through syndromes, it will be discarded. So if an adversary continues to corrupt certain packets, these packets will be continuously dropped and the communication may never succeed. Therefore, we need to

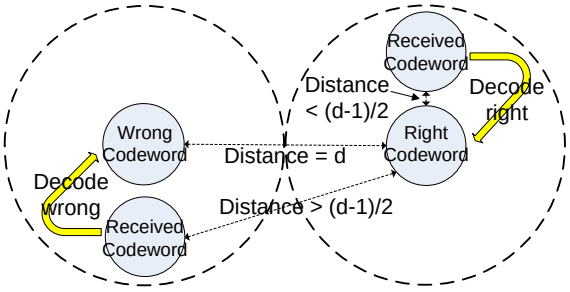

Figure 4.2 Limitations of error control codes

develop techniques that can improve the throughput for these situations.

4.2.1 EDEC Scheme

Similar to the Null Key scheme, our approach also utilizes the error control code, but we use both the error detection and error correction properties. When a corrupted packet is detected, we do not drop it. Instead we collect the corrupted packet to the sink node to correct the errors. However, the corrupted packet will not participate in network coding in the subsequent relay nodes once it is identified to be corrupted.

4.2.1.1 Limitations of Error Control Code

A linear error-correcting code encodes the original k bits message symbol $\mathbf{m}$ to an n bits codeword $\mathbf{c}$ using a generating matrix $\mathbf{G}_{k\times n}$. So the code rate is $r = k/n$. Suppose the minimum distance is d, according to the results in the Preliminary, the maximum number of errors we can correct is $\left\lfloor \frac{d-1}{2} \right\rfloor$. If the number of errors is more than this amount, we may correct the corrupted codeword into a false one, as illustrated in Figure 4.2.

4.2.1.2 Modified Error Control Code

The conventional error control code may have undetected decoding errors. This is an inherent nature. No matter how low we set the code rate, these undetected errors may exist. The

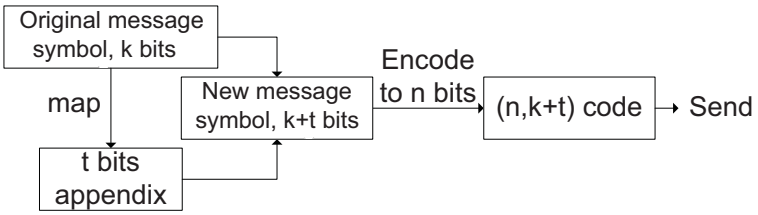

Figure 4.3 The encoding process of modified error control code in EDEC scheme

decoding errors can only be detected using mechanisms other than a stand-alone error-correcting code.

Therefore, we propose to apply modified error-control code to both message symbols and network coding coefficients in equation (4.1). In this section, we will use the message symbol as an example. The original message symbol $\mathbf{m}$ is first mapped to a t bit value $\mathbf{h}$ using a homomorphic MAC algorithm like [33]. The t bits will be appended to $\mathbf{m}$ to form a new $k+t$ bits message symbol and to get the final codeword by encoding this new message symbol. So the code becomes an $(n, k+t)$ code. By adding the extra bits, we can mitigate limitations of the conventional error-control code. Figure. 4.3 illustrates the modified encoding scheme.

Upon a successful decoding, the decoded message symbol is first split into two parts $\mathbf{m}'$ and $\mathbf{h}'$. Then we calculate the mapping of $\mathbf{m}'$: $\mathbf{h}''$. If $\mathbf{h}''$ does not equal to $\mathbf{h}'$, we can detect a decoding error. Our modification is equivalent to choose 2^k message symbols from 2^{k+t} symbols. Other message symbols in the 2^{k+t} symbol space are considered to be illegal. However, the decoding algorithm only guarantees that the decoded codeword is in the $k+t$ dimensional subspace. So if the corrected codeword belongs to the $2^{k+t}-2^k$ illegal symbol space, we know the decoding contains error. Figure 4.4 illustrates the corresponding modified decoding scheme.

Theorem 4.1. *Suppose a decoding error occurs, the wrong codeword will be any codeword in the 2^{k+t} symbol space. So the probability of detecting an erroneous decoding is:*

$$p = \frac{2^{k+t} - 2^k}{2^{k+t}} = \frac{2^t - 1}{2^t} = 1 - \frac{1}{2^t}. \tag{4.2}$$

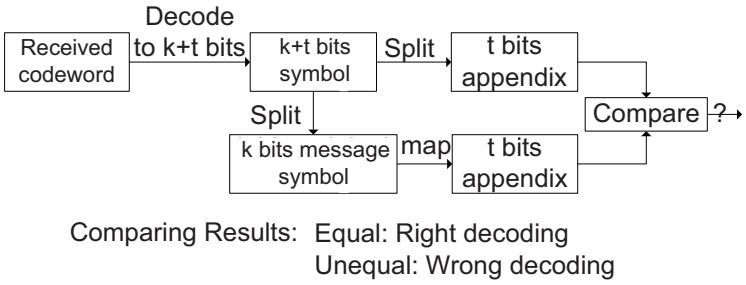

Figure 4.4 The decoding process of modified error control code in EDEC scheme

Table 4.1 Four cases of decoded codewords in modified error control code

Case	k bits original symbol	t bits mapping value	Results
1	Decoded right	Obey mapping rule	Successfully
2	Decoded right	Violate mapping rule	False alarm
3	Decoded wrong	Obey mapping rule	Miss detect
4	Decoded wrong	Violate mapping rule	Successfully

As an example, when $t = 4,\ p = \frac{15}{16}$, the probability for 3 consecutive erroneous decodings to be detected is $1 - \left(\frac{1}{16}\right)^3 \approx 0.9998$. Therefore, we only need to add a very small overhead to detect erroneous decodings.

4.2.1.3 Performance of Modified Error Control Code

In this section, we will select a cyclic code with $n = 15$, $t = 4$ to demonstrate our proposed scheme. We first add some errors to the encoded symbols, then decode these symbols as described in Figure 4.4. We evaluate the performance by checking the numbers of decoded codewords in four different cases. The results are summarized in Table 4.1.

Code with $k = 6$ In this simulation, we use a $(15, 10)$ code for the evaluation. From the results (see Figure 4.5) we can see: (i) This code (with minimum hamming distance 4) can detect and correct all the 1 bit error and part of the two bits errors. (ii) We can successfully detect most of the decoding errors when the number of errors is more than 2. In fact,

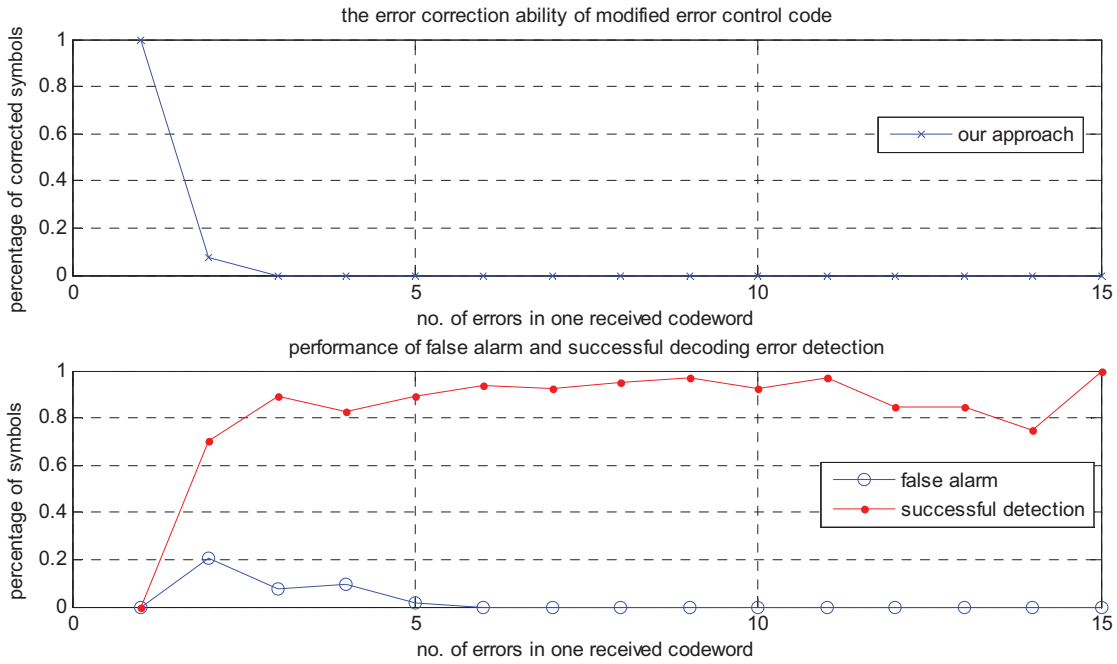

Figure 4.5 Performance of modified error control code in EDEC scheme when $k = 6$

the detection probability is larger than 0.8 most of the time except for 14 errors, in which the probability is about 0.6. (iii) False alarm cannot be distinguished from the successful detection. In fact, the false alarm is also caused by the errors beyond the correcting ability. The only difference is that the t bits appendix part of the symbol is decoded wrong. However, the false alarm is neglectable according to the results.

Code with $k = 4$ In this simulation (see Figure 4.6), we use a $(15, 8)$ code to do the evaluation. The only difference is that this code is able to correct more errors because the code rate is relatively lower. From the results of the two different code rates, we can see that adding 4 extra check bits is enough to detect erroneous correction.

4.2.1.4 Algorithms for EDEC Scheme

The proposed EDEC scheme is divided into two phases: initialization phase and transmission phase. The initialization phase is for null key and security parameter distribution while data symbols are transmitted through network coding in the transmission phase.

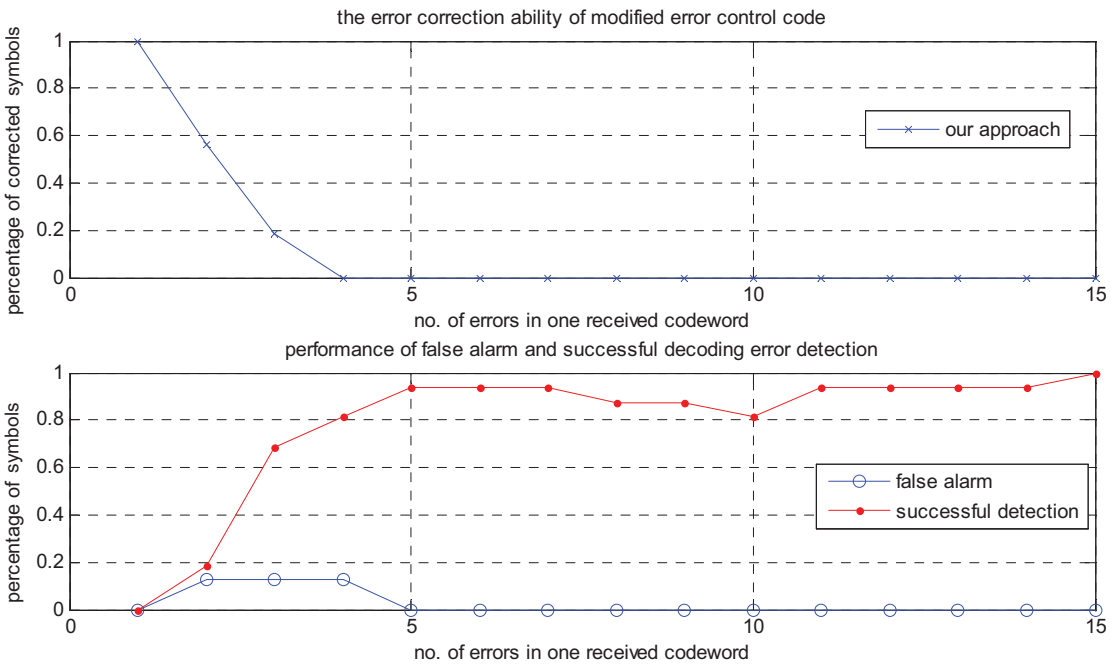

Figure 4.6 Performance of modified error control code in EDEC scheme when $k = 4$

Initialization Phase In initialization phase, the source node will first distribute the row vectors of the parity check matrix corresponding to $\mathbf{G}$ in Algorithm 4.1 (null keys) to all the relay nodes similar to [44] using homomorphic hashes. Unlike normal linear network coding in which the network encoding vectors will be attached to the start or the end of the packets, we propose to insert the encoding vectors to a predetermined secret location in the packets. The source node will send the location information to all the sink nodes during initialization phase through a secure transmission protocol such as TLS [93]. This will prevent the malicious nodes from corrupting the encoding vector, which is essential for the data decoding. Moreover, the source node will also send the encoding matrix $\mathbf{G}_c$ for network encoding vectors and $\mathbf{G}$ for data symbols to all the sink nodes. Once the initialization phase is done, the source nodes can multicast any number of packets to sink nodes. The overhead of the initialization phase is negligible.

Transmission Phase In the transmission phase, the source nodes, relay nodes and sink nodes will perform the proposed EDEC scheme according to Algorithm 4.1, 4.2 and 4.3.

Algorithm 4.1 EDEC Algorithm for Source Nodes

for packet i **do**
//Encode network encoding vector $\boldsymbol{\beta}_i$ *in equation (4.1) using the modified error-control code (Figure. 4.3)*
$\mathbf{h}_c \leftarrow \text{map}(\boldsymbol{\beta}_i)$
$\mathbf{u}_c \leftarrow (\boldsymbol{\beta}_i|\mathbf{h}_c)$
Encoded network encoding vector $\leftarrow \mathbf{u}_c \cdot \mathbf{G}_c$
for every symbol **m** of the packet **do**
//Encode **m** *using the modified error-control code (Figure. 4.3)*
$\mathbf{h} \leftarrow \text{map}(\mathbf{m})$
$\mathbf{u} \leftarrow (\mathbf{m}|\mathbf{h})$
Encoded symbol $\leftarrow \mathbf{u} \cdot \mathbf{G}$
end for
Send out the encoded encoding vector and symbols
end for

In algorithm 4.1, the source node will encode the network encoding vector $\boldsymbol{\beta}_i$ using the modified error-control code with a much longer appendix and lower code rate, compared to the encoding of data symbols. This can improve the error resistance and detection probability for erroneous decodings to guarantee the correctness of encoding vectors used for data decoding. Since there is only one encoding vector in each packet, the overhead brought by this higher security level is negligible.

Algorithm 4.2 presents the EDEC algorithm for relay nodes. Since the null keys (row vectors of the parity check matrix corresponding to $\mathbf{G}$ in algorithm 4.1) are already distributed in initialization phase, the relay nodes can check whether a packet is intact.

Algorithm 4.3 presents the EDEC algorithm for sink nodes. Since the sink node has already received the encoding matrix $\mathbf{G}_c$ and $\mathbf{G}$ in Algorithm 4.1 in initialization phase, it can perform the error-control code decoding and detection for erroneous decoding. Then it can derive the original data symbols through decoding of network coding.

Algorithm 4.2 EDEC Algorithm for Relay Nodes

```
if every symbol in the received packet is intact then
  if the packet is independent then
    Save the packet
    if x (a predetermined number) packets are collected then
      repeat
        Generate x randomly, linearly combined packets using the saved packets (network
        coding)
      until the x new packets are independent
      Send out the x packets
    end if
  end if
else
  if the packet is independent from all the previous packets then
    Mark the packet as corrupted and send it out
  end if
end if
```

Algorithm 4.3 EDEC Algorithm for Sink Nodes

```
A packet is received
Decode the network encoding vector and every symbol in the packet using the decoding
algorithm for the modified error-control code (Figure. 4.4)
if the network encoding vector and all symbols are decoded correctly then
  if the packet is independent then
    Save the packet
    if l (in equation (1.1)) independent packets are saved then
      Solve the network coding equations
    end if
  end if
end if
```

4.2.2 Simulation in ns-2

In this section, the simulation platform for EDEC scheme in ns-2 [94] is first presented. Then we will compare the EDEC scheme and the error-detection based schemes. In the simulation, we implement the Null Key scheme to represent the error-detection based schemes.

4.2.2.1 Simulation Platform

ns-2 is a discrete event simulator that provides comprehensive support for simulation of network protocols. It is ideal for the simulation of EDEC scheme. The scenario is set as a grid network with one source node, a number of relay nodes and sink nodes. All the nodes are set as wireless nodes using wireless physical layer, 802.11 MAC protocol and AODV routing protocol. The wireless channel is set to TwoRayGround. The nodes transmit packets using broadcasting. Once a node receives a packet, it will start the corresponding operations depending on its type (source, relay, malicious, sink) and the packet content.

Figure 4.7 shows the topology of the simulated network. The source node is located at the lower left corner and 19 sink nodes lie at the upper right. The rest nodes are all intermediate nodes that can relay packets. In the simulation, we randomly pick a number of intermediate nodes as malicious nodes to perform pollution attacks. These nodes can add certain errors to received packets before sending the packets out to pollute the network. We can change the number of malicious nodes to evaluate performance of the algorithms under different network conditions. As an example, in Figure 4.7, we randomly pick 50 nodes out of 209 intermediate nodes to be malicious nodes. The malicious ratio is about $50/209 = 24\%$. The rest of the intermediate nodes act as relay nodes. After receiving a packet, they will first conduct the pollution detection. In the error-detection based schemes, if the packet is corrupted, it will be dropped. While in the EDEC scheme, we will forward these packets. However, these packets will not participate in network coding. The nodes behaviors will be detailed in the next section.

Because the packets are transmitted through broadcasting, although the MAC protocol is IEEE 802.11, we will still have packets collisions that will eventually influent the simulation results. In this dissertation, we only focus on network layer protocols. Thus after considering the transmission range of the single node, adjacent nodes are assigned different time slots (see Figure 4.8) to avoid packets collisions. There are 9 time slots in total and the duration

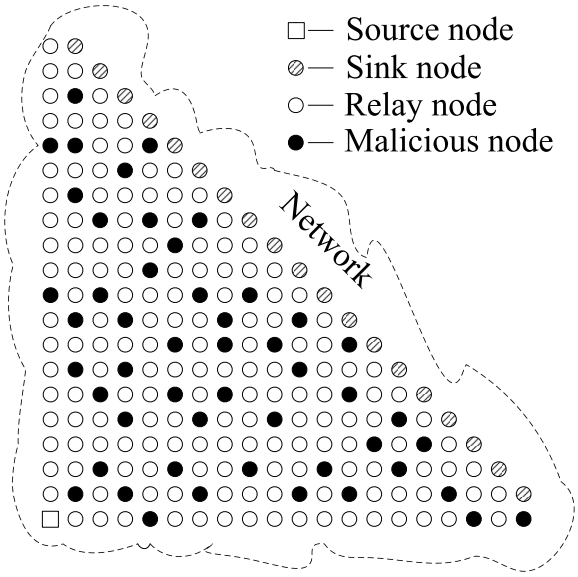

Figure 4.7 Simulation scenario

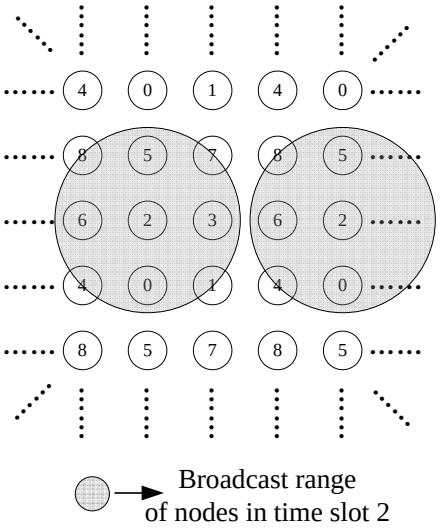

Figure 4.8 9 time slots to avoid packets collisions

of each time slot is 100ms. The nodes are allowed to send packets only if they are in their own slots. If not, they will have to wait until their next slots. In Figure 4.8, we give an example for nodes that belong to time slot 2 to simultaneously transmit without packets collisions.

4.2.2.2 Nodes Design

After setting up the simulation platform, we can fairly evaluate the algorithms without considering other facts. Four types of nodes are designed according to the algorithms described above.

Source Node In the simulation, the source node will multicast a 352-symbol message, which is fragmented into 32 packets of 11 symbols. Each symbol has the size $k = 512$ bits. In the whole network there will be 32 linearly independent packets. That is $l = 32$ in equation (1.1). After initializing the network, the source node will encode each data symbol using the modified error-control code presented in Figure. 4.3 with $t = 16$. The encoding vector $\boldsymbol{\beta}_i$ will also be encoded using the modified error-control code with $t = 32$. According to Theorem 4.1, the probability of detecting an erroneous decoding for the encoding vector is about $1 - 2^{-10}$, which means once the decoded encoding vector pass the verification in Figure. 4.4, we can view the encoding vector as intact. Then the source node will insert encoded network encoding vector into the predetermined location in each packet and send out the packet.

Relay Nodes Relay nodes will perform EDEC scheme according to algorithm 4.2. Because the network is collision free and all the transmitted packets can be received, each packet only needs to be transmitted once. So if a newly received valid packet is linearly dependent of previous transmitted packets, it will be discarded. Since there are 32 linearly independent packets in total, if a relay node does not transmit packets until all the 32 packets have been received, the time delay will be huge. Moreover, a relay node may never be able to collect all the 32 valid packets due to malicious attacks. To use network coding efficiently while minimizing the time delay, relay nodes will perform network encoding once they collect 4 independent valid packets.

Malicious Nodes Similar to the relay nodes, malicious nodes only send out independent packets. However, we assume that the malicious nodes will not perform network encoding in this case. They only pollute packets and send out corrupted packets.

Sink Nodes Sink nodes will decoding both the network coding and the underling modified error-control code according to algorithm 4.3. After the original symbols are successfully retrieved, all the packets received afterwards will be ignored.

4.2.2.3 Simulation Results

We conducted simulations under different percentages of the malicious relay nodes. To make the results more clear, we first fix the number of bits that the malicious nodes can corrupt for each symbol. Then we make this number random according to our adversary model.

Small Number of Errors When the number of bits that malicious nodes can corrupt for each symbol is within the capability of error control codes, the throughput comparison between the EDEC scheme and the error-detection based schemes is shown in Figure 4.9. In the figure we can see that: (i) When the percentage of malicious nodes is less than 10%, the performance of the two schemes are almost the same. (ii) With the increasing of the malicious nodes, the performance of error-detection based schemes degrade significantly. While the throughout of the EDEC scheme remains unchanged. (iii) When the percentage of the malicious nodes is larger than 65%, the error-detection based schemes do not work at all because too many corrupted packets have been dumped. However, the throughput for the EDEC scheme still remains unchanged because the EDEC scheme can successfully recover all of the message symbols from the corrupted packets. This scenario will remain true as long as the corrupted packet symbols are within the capability of the error control codes. In this case, the EDEC scheme surpasses the error-detection based schemes in throughput.

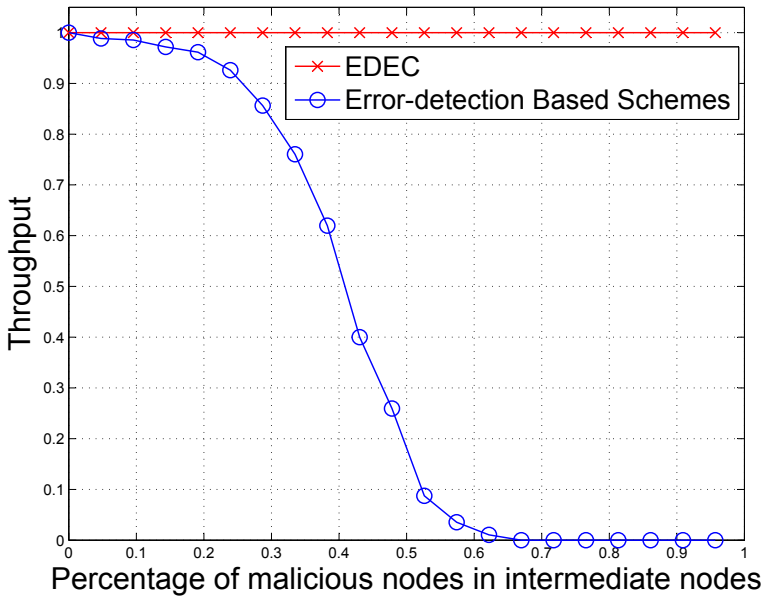

Figure 4.9 Throughput comparison between EDEC scheme and the error-detection schemes based on the number of bit corrupted in each symbol — for small number of errors

Large Number of Errors When the number of bits corrupted in each symbol of the received packets is beyond the capability of the error control codes, the throughput comparison between the EDEC scheme and the error-detection based schemes is shown in Figure 4.10. From the results we can see that the performance of the two schemes are almost the same. This is because the corrupted packets that cannot be recovered by the EDEC scheme have already been dumped by the error-detection based schemes.

Random Number of Errors When the malicious nodes adds random errors to the symbols in the received packets, the performance of the EDEC scheme is comparable with the error-detection based schemes. This is because that while some of the symbols in the corrupted packets can be corrected, but some are beyond the decoding capability of the error control code, which makes the packet unusable with result similar to the packet being dumped in the error-detection based schemes.

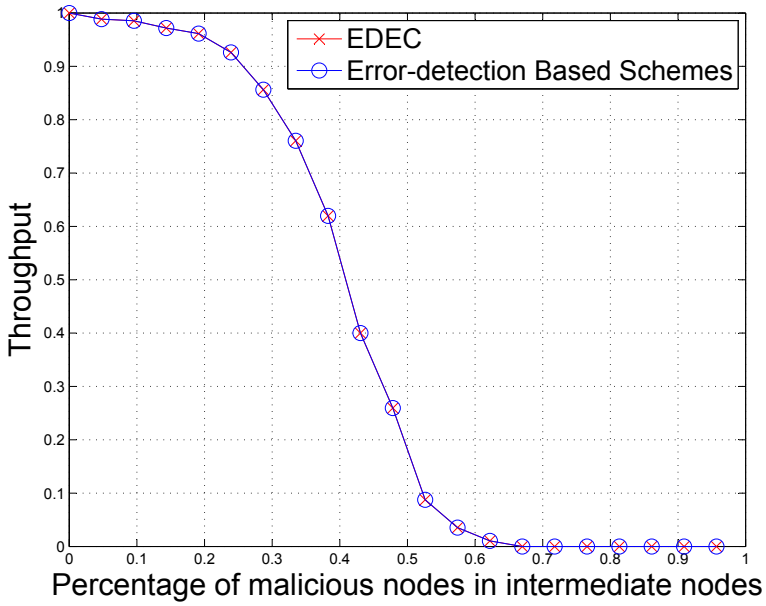

Figure 4.10 Throughput comparison between EDEC scheme and the error-detection schemes based on the number of bit corrupted in each symbol — for large number of errors

4.3 LDPC Decoding and LEDEC Scheme

In the EDEC scheme, only linearly independent packets participate in the network decoding at the sink nodes. Corrupted or linearly dependent packets will not be used. In this section, we will explore utilizing these packets to recover more message symbols using LDPC decoding.

4.3.1 LDPC Code

Low density parity check (LDPC) linear block code was first introduced by Gallager in 1962 [95]. One of the important characteristic of LDPC code is its sparse parity check matrix. By using iterative decoding, LDPC code can achieve error-correction performance close to Shannon bounds [96]. The advantages of LDPC code were discussed in [97, 98]. Some new classes of asymptotically good LDPC codes were studied in [99–101]. And some decoding algorithms of LDPC codes were presented in [102–105].

LDPC codes can be categorized as the regular LDPC code, of which the parity check

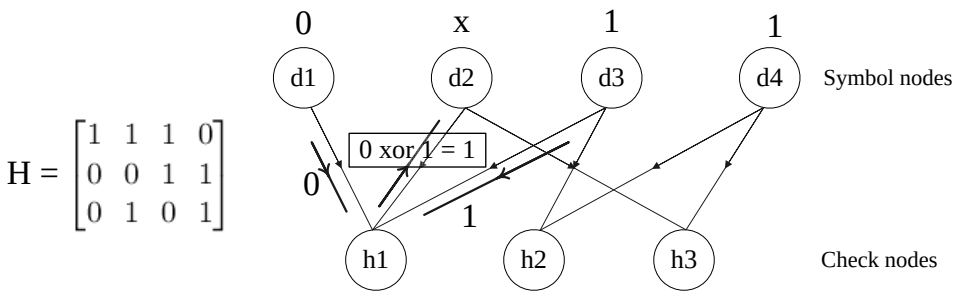

Figure 4.11 An illustrative example of parity check matrix and Tanner graph

matrix **H** has fixed number of 1's per column and per row, and the irregular LDPC code [106], of which the parity check matrix may have different number of 1's in each column and each row. In this dissertation, we will formulate the network coding to the irregular LDPC code.

4.3.2 Decoding of LDPC Code

The iterative decoding algorithm, known as belief propagation algorithm (BPA), is generally used to decode the LDPC code. The BPA is a soft-decision algorithm studied in [107–109]. For a binary erasure channel (BEC), the bits in the codewords are received as 0's, 1's or x's (erasures). The BPA can be described over the Tanner graph [110], which is a bipartite graph. In a Tanner graph, there are two types of nodes: the symbol nodes (corresponding to the received bits), and the check nodes (corresponding to the rows of the parity check matrix).An illustrative example of the parity check matrix and its Tanner graph is shown in Figure 4.11. In the parity check matrix, every row represents a parity check equation. The symbol nodes, which correspond to the bits equal to 1's in a row of the parity check matrix, are connected to the check node which corresponds to the same row. These nodes and edges in the Tanner graph express the parity check equation of that row.In Figure 4.11, node $h1$ represents the first row of the parity check matrix. And the first, the second and third elements of the first row in parity check matrix are 1's, so symbol nodes $d1$, $d2$ and $d3$ are connected to $h1$ in the Tanner graph.

The decoding algorithm can be described through the following algorithm:

Algorithm 4.4 BPA Decoding Algorithm for BEC

while There are check nodes connected to only one unknown symbol node **do**
  for Each of these check nodes **do**
    The unknown symbol node ← xor (All of the other symbol nodes connected to the check node)
  end for
end while
if All the unknown symbol nodes are recovered **then**
  Decode successfully
end if

4.3.3 Relationship Between Linear Network Code and LDPC Code

In linear network coding, packets are linearly combined at the intermediate nodes. The packets that are received at the sink nodes satisfy equation (1.1). In the network code decoding part of the EDEC algorithm, only independent valid packets are used. However, there is also helpful information in the linearly dependent packets or corrupted packets. If we can exploit and use these packets, we can improve the system performance. Denote the received encoding vector as $\mathbf{a}_i = (a_{1,i}, \cdots, a_{l,i})^T$, where $1 \leq i \leq m$ and m is the number of received packets. Then the generation matrix of the block code can be defined as:

$$\mathbf{G} = [\mathbf{a}_1 \cdots \mathbf{a}_l, \mathbf{a}_{l+1} \cdots \mathbf{a}_m] = [P_1, P_2], \tag{4.3}$$

where the matrix P_1 can be made as a $l \times l$ full rank matrix through column exchange after l independent packets are received, and P_2 is a $l \times (m - l)$ matrix.

As an example, suppose there is only one bit x_i in every original packet in the source node ($1 \leq i \leq l$). Define $\mathbf{x} = (x_1, \ldots, x_l)$. In this case, there is also only one bit y_j in every received packet in the sink nodes ($1 \leq j \leq m$). Denote all the m received packets as a vector $\mathbf{y}$. We have the following encoding equation:

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{G}. \tag{4.4}$$

The corresponding parity check matrix $\mathbf{H}$ can be derived as follows.

Define the generating matrix as

$$\mathbf{G} = P_1^{-1} \cdot [I_l, P_1^{-1} P_2], \tag{4.5}$$

and the parity-check matrix as

$$\mathbf{H} = [(P_1^{-1} P_2)^T, I_{m-l}]. \tag{4.6}$$

We can verify the correctness of $\mathbf{H}$ by verifying the follow equation:

$$\mathbf{G} \cdot \mathbf{H}^T = P_1^{-1} \cdot [I_l, P_1^{-1} P_2] \cdot [(P_1^{-1} P_2)^T, I_{m-l}]^T = \mathbf{0}. \tag{4.7}$$

After deriving the corresponding parity check matrix $\mathbf{H}$, we can decode the linear network code using the BPA algorithm. The linear network code can be viewed as a rateless LDPC code, and has the property of error control codes.

Although linear network codes can be seen as rateless LDPC codes, the BPA algorithm cannot be used to decode a network code if the network code is derived after a normal error control encode, because we cannot find the incorrect decodings which can be viewed as erasures. However, for the modified error control codes in the EDEC scheme, we can determine the erroneous decodings and mark the corresponding bits as erasures. Therefore, we can decode the linear network code using the BPA.

4.3.4 LEDEC Scheme Using BPA

In the LEDEC scheme, we use the linearly dependent packets and the corrupted packets and decode the linear network code using BPA algorithm. Figure 4.12 illustrates this main idea of the decoding algorithm.

4.3.5 Theoretical Analysis

When the number of errors is partially beyond the decoding capability of the error control code, the LEDEC scheme can get additional benefits from decoding of the LDPC codes.

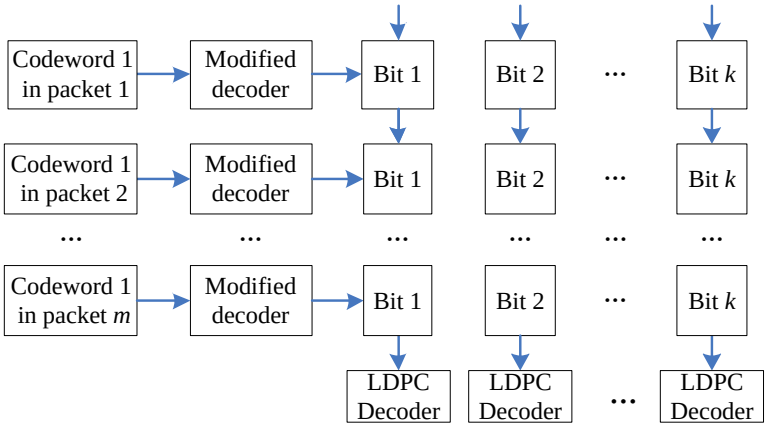

Figure 4.12 Main idea of the LEDEC scheme

Here we use a $(31, 15)$ cyclic code with generation polynomial $x^{16} + x^{14} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$ as an illustrative example. It is easy to calculate that there are only 17515 entries for the 4-bit errors in the syndrome table, while the number for all the 4-bit errors is $\binom{31}{4} = 31465$. It means only 17515 distinct 4-bit errors can be successfully corrected. In this situation, 4-bit errors are considered to be partially beyond the decoding capability of the $(31, 15)$ code. The successful error correction probability is about $17515/31465 = 0.5567$. Because we use the modified error-control code, which can detect the erroneous decodings, the failed error corrections can be seen as erasures with erasure probability $P_e = 1 - 0.5567 = 0.4433$.

Consider the worst case in which almost all the packets are corrupted by the malicious nodes. The error-detection based schemes do not work at all because all of the packets are dumped. The EDEC scheme does not work either because with erasure probability $P_e = 0.4433$ there will not be enough correctable packets to solve the network coding equation (1.1).

For the LEDEC scheme, let λ_d denote the probability that an edge from a check node is connected to a symbol node of degree d, and ρ_d denote the probability that an edge from a symbol node is connected to a check node of degree d in the Tanner graph of the corresponding LDPC code. The generating functions for an LDPC code is defined as: $\lambda(x) =$

$\sum_d \lambda_d x^{d-1}$, $\rho(x) = \sum_d \rho_d x^{d-1}$. According to [111], the maximal fraction of erasures that a random LDPC code with given generating functions can correct is bounded by $P_{\max} = \min\{\frac{x}{\lambda(1-\rho(1-x))}\}$ $(0 < x < 1)$ with probability at least $1 - \mathcal{O}(l^{-3/4})$, where l is the length of the code. For the throughput of the LEDEC scheme, we have the following theorem:

Theorem 4.2. *The throughput of the LEDEC scheme is*

$$F = \sum_{i=0}^{\lfloor N \cdot P_{\max} \rfloor} \binom{N}{i} P_e^i (1 - P_e)^{N-i},$$

where $P_{\max} = \min\{\frac{x}{\lambda(1-\rho(1-x))}\}$ $(0 < x < 1)$*,* P_e *is the erasure probability,* N *is the number of packets a sink node received and* $\lfloor \cdot \rfloor$ *is the floor function.*

Proof. Suppose a sink node receives N packets and the erasures in the packets are independent, the distribution of the number of erasures i in every N received packet symbols is a binomial distribution with $\Pr(i) = \binom{N}{i} P_e^i (1 - P_e)^{N-i}$, $0 \leq i \leq N$ as the probability mass function (PMF).

The proposed scheme can combat all erasures up to $N \cdot P_{\max}$ with probability at least $1 - \mathcal{O}(N^{-3/4})$, which is close to 1. Thus the throughput can be written as $F = \sum_{i=0}^{\lfloor N \cdot P_{\max} \rfloor} \Pr(i)$.

□

4.3.6 Performance Analysis and Simulation

In this section, we provide simulation results of the LEDEC scheme on the simulation platform presented in Section 4.2.2. All the settings and parameter are the same as Section 4.2.2.

4.3.6.1 Nodes Design

For the LEDEC scheme, the source node, relay nodes and malicious nodes are the same as those in Section 4.2.2. However, the decoding process in the sink nodes is different. All packets received will be used, but the BPA decoding will not start until the sink nodes collect all

the $l = 32$ independent packets. After receiving $l = 32$ independent packets, theoretically, we can use the BPA algorithm to decode whenever a new packet arrives. However, there is a trade-off in determining when to start the BPA algorithm. When it is used too frequently, it may result in a high computational overhead. On the other side, if we do not start the BPA decoding until we have collected a large number of packets, the communication delay may be too high. To balance these two conflicting issues, the sink nodes will trigger the BPA decoding upon receiving of every 10 new packets. This process will continue until all the message symbols have been successfully decoded. While the BPA decoding is more powerful than the linear-equation-solving hard-decision decoding method described in Section 4.2.2, the computational overhead of the BPA scheme is relatively higher. To optimize the advantages of the two algorithms, in our scheme, when a sink node receives 32 independent and intact packets, we will directly solve the equations and decode the packets using the scheme described in Section 4.2.2. The flowchart of the LEDEC algorithm that is implemented in the sink nodes is shown in Figure 4.13.

4.3.6.2 Simulation Results

Same as in Section 4.2.2.3, the simulations in this section are carried out under different percentage of malicious relay nodes. And the number of bits that the malicious nodes can corrupt in each symbol is fixed first then set to be random. One example of the parity check matrix generated in the linear network coding is shown in Figure 4.14. In this example, the sink node receives 90 packets and decodes the linear network code using the BPA algorithm. The size of the matrix is 58×90. In the figure, white squares represent 0 and black squares represent 1. We can see that this matrix is a sparse matrix.

1. Small Number and Large Number of Errors

   Remark 1. *When the number of bits that the malicious nodes can corrupt in each symbol is either within or entirely beyond the capability of the error control code and*

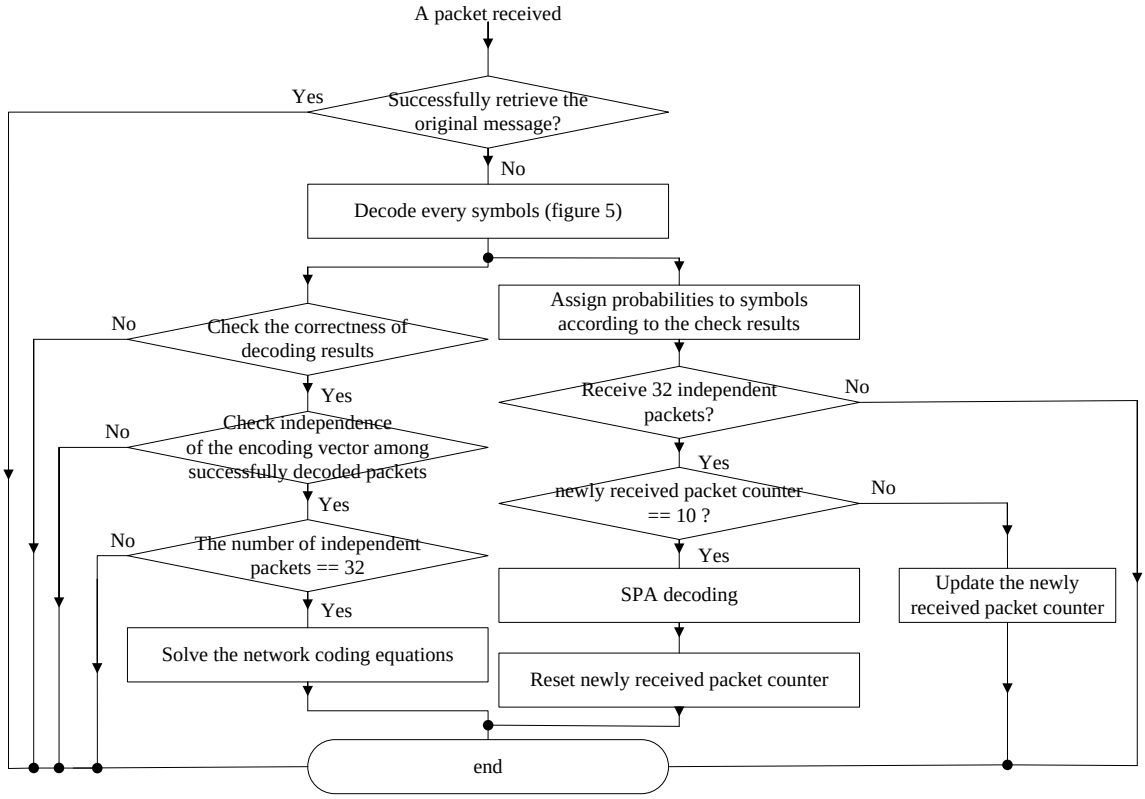

Figure 4.13 Flowchart of the LEDEC algorithm implemented in the sink nodes

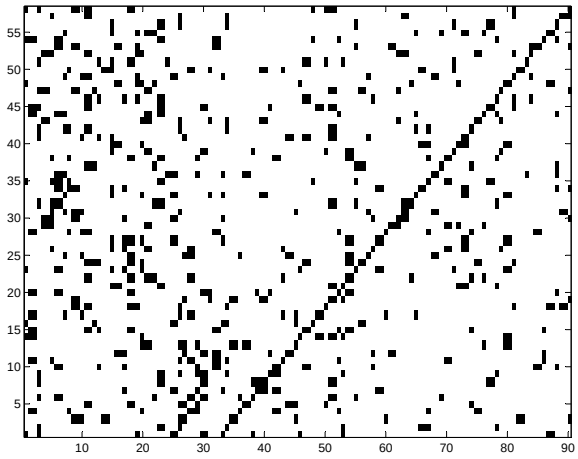

Figure 4.14 An example of the parity check matrix in network coding

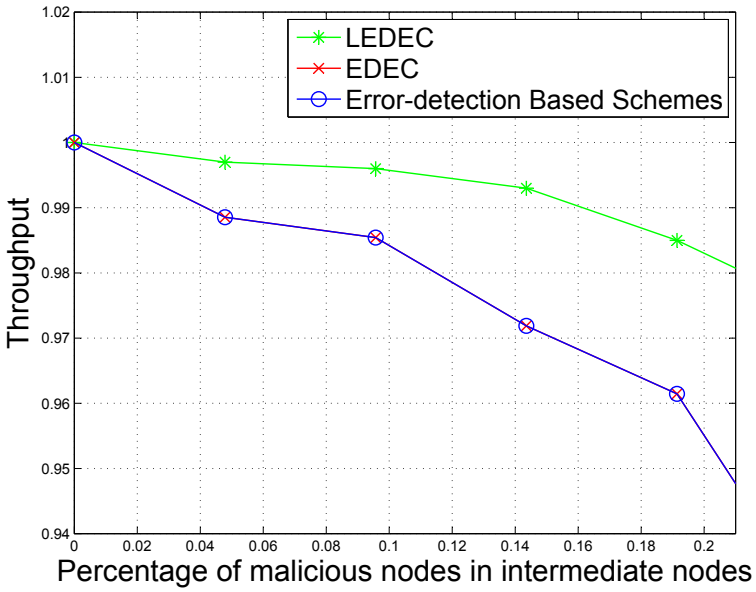

Figure 4.15 Performance comparison for small number of malicious nodes

the network code, the performance of the LEDEC scheme is the same as that of the EDEC scheme shown in Figure 4.9 and Figure 4.10.

2. Medium Number of Errors

When the number of errors is partially beyond the capability of error control code and network code, the performance of the LEDEC scheme is shown in Figure 4.15, Figure 4.16 and Figure 4.17.

Remark 2. *When the percentage of the malicious nodes is less than 20%, the performance of the LEDEC scheme is slightly better than the EDEC and the error-detection based schemes. This is because the sink node can successfully decode the corrupted packets using only the intact packets.*

Remark 3. *When the percentage of malicious nodes is between 20% and 60%, the performance of the LEDEC scheme is about 15% better than the EDEC and the error-detection based schemes. This is because the sink nodes can recover extra information from the corrupted packets.*

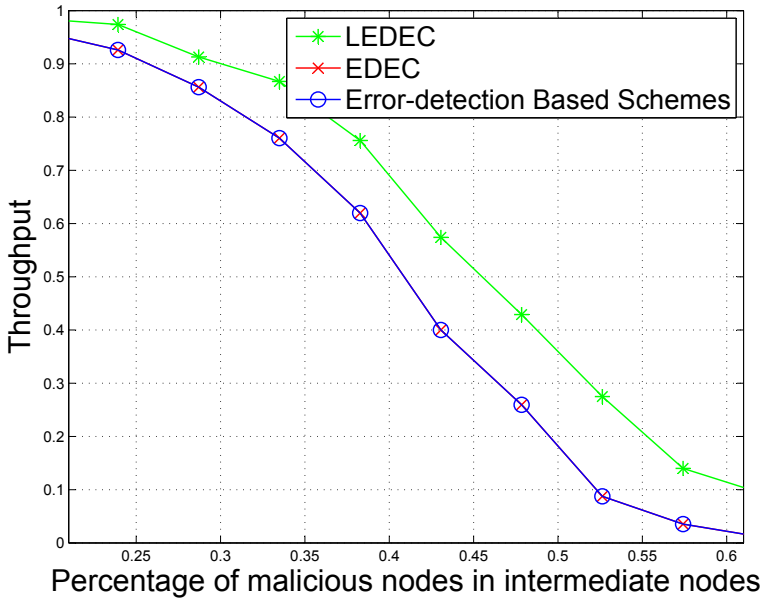

Figure 4.16 Performance comparison for medium number of malicious nodes

When the percentage of malicious nodes becomes more than 65%, the average generating functions for the random parity-check matrix are: $\lambda(x) = 0.001x^{18} + 0.0013x^{17} + 0.0003x^{16} + 0.0008x^{15} + 0.0048x^{14} + 0.0098x^{13} + 0.0116x^{12} + 0.028x^{11} + 0.0404x^{10} + 0.0369x^9 + 0.0644x^8 + 0.129x^7 + 0.0902x^6 + 0.076x^5 + 0.1318x^4 + 0.2119x^3 + 0.1364x^2 + 0.0255x$, $\rho(x) = 0.8397x^9 + 0.1575x^7 + 0.0027x^5$, $P_{\max} = \min\{\frac{x}{\lambda(1-\rho(1-x))}\} \approx 0.3812$. In this worst case, every sink node will receive about $N = 90$ packets according to the topology, and at most about $N \cdot P_{\max} = 0.3812 \times 90 \approx 34$ erasures in every 90 packet symbols can be recovered. The parameters for the binomial distribution is $N = 90, P_e = 0.44$. So the throughput can be calculated as $F(34) = \sum_{K=0}^{K=34} Pr(K) \approx 0.1$. This result is very close to our simulation, which is summarized in the Remark 4.

Remark 4. *When percentage of the malicious nodes becomes more than 65%, the error-detection based schemes and the EDEC scheme do not work because the number of the corrupted packets has exceeded the decoding capacity of the network codes. However, the LEDEC scheme can still maintain a throughput around 8% due to the partial information available from the corrupted packets.*

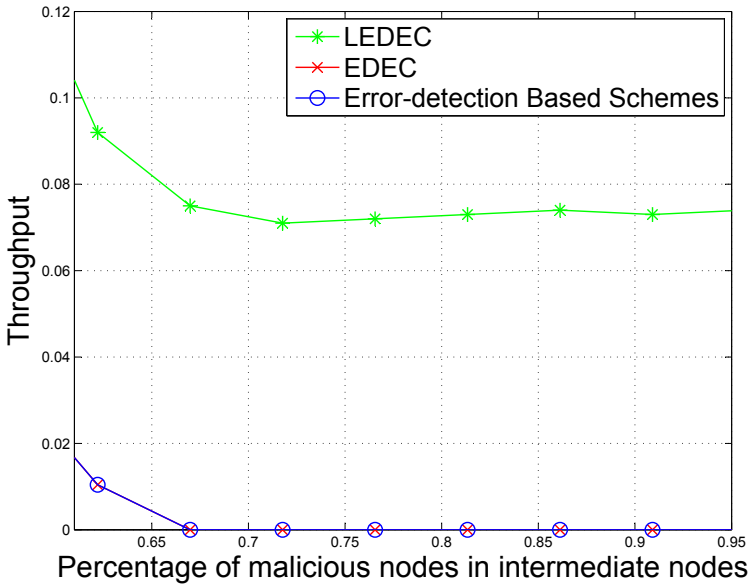

Figure 4.17 Performance comparison for large number of malicious nodes

3. Random Number of Errors Here we study the case when the malicious nodes adds random number of errors to the symbols in the received packets. The number of errors may vary from small number of errors to large number of errors.

Remark 5. *For random errors, although some symbols in the corrupted message cannot be corrected, the LEDEC scheme can still recover symbols using the LDPC decoding from the correctable symbols in corrupted packets. From Figure. 4.18 we can see that while the percentage of malicious nodes is between 30% and 60%, the performance of the LEDEC scheme is about 4% better than the EDEC and the error-detection based schemes on average.*

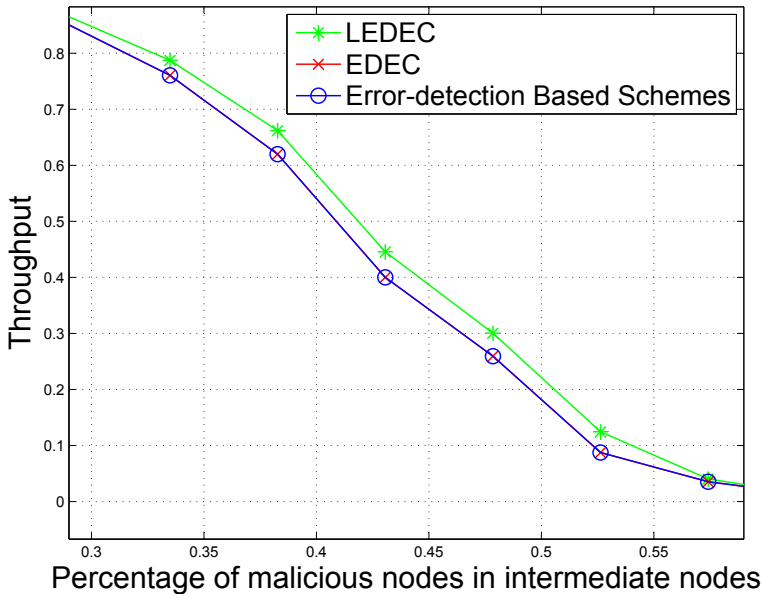

Figure 4.18 Performance comparison based on medium number of malicious nodes (random number of errors)

CHAPTER 5

DISTRIBUTED STORAGE IN HOSTILE NETWORKS — HERMITIAN CODE BASED REGENERATING CODES APPROACH

In this chapter, we will propose Hermitian code based regenerating codes: H-MSR code and H-MBR code. Theoretical evaluation shows that our proposed schemes can detect the erroneous decodings and correct more errors in the hostile network than the RS-MSR code and the RS-MBR code with the same code rate respectively. We will construct the H-MSR code by combining the Hermitian code and regenerating code at the MSR point, then we will construct the H-MBR code by combining the Hermitian code and regenerating code at the MBR point. We will prove that these codes can achieve the theoretical MSR bound and MBR bound respectively. We will also propose data regeneration and reconstruction algorithms for the H-MSR code and the H-MBR code in both error-free networks and hostile networks. Then we will compare their performance with RS code based regenerating codes.

5.1 System/Adversarial Models and Assumptions

In this chapter, we assume there is a secure server that is responsible for encoding and distributing the data to storage nodes. Replacement nodes will also be initialized by the secure server. DC and the secure server can be implemented in the same computer and can never be compromised. We also assume that DC keeps the encoding matrix as a secret and each storage node only knows its own encoding vector.

We assume some storage nodes can be corrupted due to hardware failure or communication errors, and/or be compromised by malicious users. As a result, upon request, these nodes may send out incorrect response to disrupt the data regeneration and reconstruction. For malicious users, they can take full control of τ ($\tau \leq n$) storage nodes and collude to perform attacks.

We will refer these symbols as *bogus* symbols without making distinction between the corrupted symbols and compromised symbols. We will also use corrupted nodes, malicious nodes and compromised nodes interchangeably without making any distinction.

5.2 An Illustrative Example

In this section, we will show an example in distributed storage using pure RS code and Hermitian code to show the starting point of this research: the Hermitian code can correct more errors than the RS code under the same code rate.

5.2.1 RS Code in Distributed Storage

Suppose we have data

$$\begin{aligned}
\boldsymbol{m} \;=\; (\;& 1,1,1,0,0,1,\ 1,1,1,1,0,1,\ 1,0,0,0,1,1,\ 0,0,1,0,0,0,\\
& 0,0,1,0,0,1,\ 0,1,0,0,0,0,\ 1,1,0,1,0,1,\ 0,1,0,0,0,0,\\
& 1,1,0,1,0,0,\ 0,0,1,1,1,1,\ 1,1,1,0,1,1,\ 0,1,0,1,1,0,\\
& 0,0,1,1,0,0,\ 0,1,0,0,0,0,\ 1,0,0,1,1,1,\ 0,1,1,1,1,0,\\
& 0,1,0,1,1,0,\ 1,1,0,1,0,1,\ 1,0,0,1,0,1,\ 1,0,0,0,1,1,\\
& 1,1,1,0,1,0,\ 0,1,0,0,1,0,\ 1,1,0,0,0,0,\ 1,1,0,0,0,0,\\
& 0,1,1,0,0,0,\ 1,0,0,1,0,0,\ 0,0,0,1,0,0,\ 0,0,0,0,1,1,\\
& 1,0,0,0,0,1,\ 1,1,0,0,0,1,\ 1,1,1,0,1,1,\ 0,0,1,0,0,0\ ),
\end{aligned} \tag{5.1}$$

to be distributively stored in the distribute storage network. If we view $\boldsymbol{m}$ as composed of elements from F_{2^6}, then $\boldsymbol{m}$ can be represented as 32 symbols, each symbol can be represented using 6 bits: $(1,1,1,0,0,1)$, $(1,1,1,1,0,1)$, $\ldots$, $(0,0,1,0,0,0)$. Let F_{2^6} be generated through

field expansion with the primitive polynomial $f(x) = x^6 + x + 1$ over F_2 by adding the root α of $f(x)$ to F_{2^6}. In this way, we have $GF(2^6) = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6 = \alpha + 1, \alpha^7 = \alpha^2 + \alpha, \alpha^8 = \alpha^3 + \alpha^2, \alpha^9 = \alpha^4 + \alpha^3, \alpha^{10} = \alpha^5 + \alpha^4, \alpha^{11} = \alpha^5 + \alpha + 1, \alpha^{12} = \alpha^2 + 1, \alpha^{13} = \alpha^3 + \alpha, \alpha^{14} = \alpha^4 + \alpha^2, \alpha^{15} = \alpha^5 + \alpha^3, \alpha^{16} = \alpha^4 + \alpha + 1, \alpha^{17} = \alpha^5 + \alpha^2 + \alpha, \alpha^{18} = \alpha^3+\alpha^2+\alpha^1+1, \alpha^{19} = \alpha^4+\alpha^3+\alpha^2+\alpha, \alpha^{20} = \alpha^5+\alpha^4+\alpha^3+\alpha^2, \alpha^{21} = \alpha^5+\alpha^4+\alpha^3+\alpha+1, \alpha^{22} = \alpha^5 + \alpha^4 + \alpha^2 + 1, \alpha^{23} = \alpha^5 + \alpha^3 + 1, \alpha^{24} = \alpha^4 + 1, \alpha^{25} = \alpha^5 + \alpha, \alpha^{26} = \alpha^2 + \alpha + 1, \alpha^{27} = \alpha^3+\alpha^2+\alpha, \alpha^{28} = \alpha^4+\alpha^3+\alpha^2, \alpha^{29} = \alpha^5+\alpha^4+\alpha^3, \alpha^{30} = \alpha^5+\alpha^4+\alpha+1, \alpha^{31} = \alpha^5+\alpha^2+1, \alpha^{32} = \alpha^3 + 1, \alpha^{33} = \alpha^4 + \alpha, \alpha^{34} = \alpha^5 + \alpha^2, \alpha^{35} = \alpha^3 + \alpha + 1, \alpha^{36} = \alpha^4 + \alpha^2 + \alpha, \alpha^{37} = \alpha^5 + \alpha^3 + \alpha^2, \alpha^{38} = \alpha^4 + \alpha^3 + \alpha + 1, \alpha^{39} = \alpha^5 + \alpha^4 + \alpha^2 + \alpha, \alpha^{40} = \alpha^5 + \alpha^3 + \alpha^2 + \alpha + 1, \alpha^{41} = \alpha^4 + \alpha^3 + \alpha^2 + 1, \alpha^{42} = \alpha^5 + \alpha^4 + \alpha^3 + \alpha, \alpha^{43} = \alpha^5 + \alpha^4 + \alpha^2 + \alpha + 1, \alpha^{44} = \alpha^5 + \alpha^3 + \alpha^2 + 1, \alpha^{45} = \alpha^4 + \alpha^3 + 1, \alpha^{46} = \alpha^5 + \alpha^4 + \alpha, \alpha^{47} = \alpha^5 + \alpha^2 + \alpha + 1, \alpha^{48} = \alpha^3 + \alpha^2 + 1, \alpha^{49} = \alpha^4 + \alpha^3 + \alpha, \alpha^{50} = \alpha^5 + \alpha^4 + \alpha^2, \alpha^{51} = \alpha^5 + \alpha^3 + \alpha + 1, \alpha^{52} = \alpha^4 + \alpha^2 + 1, \alpha^{53} = \alpha^5 + \alpha^3 + \alpha, \alpha^{54} = \alpha^4 + \alpha^2 + \alpha + 1, \alpha^{55} = \alpha^5 + \alpha^3 + \alpha^2 + 1, \alpha^{56} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1, \alpha^{57} = \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha, \alpha^{58} = \alpha^5 + \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1, \alpha^{59} = \alpha^5+\alpha^4+\alpha^3+\alpha^2+1, \alpha^{60} = \alpha^5+\alpha^4+\alpha^3+1, \alpha^{61} = \alpha^5+\alpha^4+1, \alpha^{62} = \alpha^5+1\}$, and $\boldsymbol{m}$ can be represented as $\boldsymbol{m} = (\alpha^5+\alpha^4+\alpha^3+1, \alpha^5+\alpha^4+\alpha^3+\alpha^2+1, \alpha^5+\alpha+1, \alpha^3, \alpha^3+1, \alpha^4, \alpha^5+\alpha^4+\alpha^2+1, \alpha^4, \alpha^5+\alpha^4+\alpha^2, \alpha^3+\alpha^2+\alpha+1, \alpha^5+\alpha^4+\alpha^3+\alpha+1, \alpha^4+\alpha^2+\alpha, \alpha^3+\alpha^2, \alpha^4, \alpha^5+\alpha^2+\alpha+1, \alpha^4+\alpha^3+\alpha^2+\alpha, \alpha^4+\alpha^2+\alpha, \alpha^5+\alpha^4+\alpha^2+1, \alpha^5+\alpha^2+1, \alpha^5+\alpha+1, \alpha^5+\alpha^4+\alpha^3+\alpha, \alpha^4+\alpha, \alpha^5+\alpha^4, \alpha^5+\alpha^4, \alpha^4+\alpha^3, \alpha^5+\alpha^2, \alpha^2, \alpha+1, \alpha^5+1, \alpha^5+\alpha^4+1, \alpha^5+\alpha^4+\alpha^3+\alpha+1, \alpha^3)$. Define $g(x) = \alpha^5+\alpha^4+\alpha^3+1+(\alpha^5+\alpha^4+\alpha^3+\alpha^2+1)x+(\alpha^5+\alpha+1)x^2+\alpha^3x^3+(\alpha^3+1)x^4+\alpha^4x^5+(\alpha^5+\alpha^4+\alpha^2+1)x^6+\alpha^4x^7+(\alpha^5+\alpha^4+\alpha^2)x^8+(\alpha^3+\alpha^2+\alpha+1)x^9+(\alpha^5+\alpha^4+\alpha^3+\alpha+1)x^{10}+(\alpha^4+\alpha^2+\alpha)x^{11}+(\alpha^3+\alpha^2)x^{12}+\alpha^4x^{13}+(\alpha^5+\alpha^2+\alpha+1)x^{14}+(\alpha^4+\alpha^3+\alpha^2+\alpha)x^{15}+(\alpha^4+\alpha^2+\alpha)x^{16}+(\alpha^5+\alpha^4+\alpha^2+1)x^{17}+(\alpha^5+\alpha^2+1)x^{18}+(\alpha^5+\alpha+1)x^{19}+(\alpha^5+\alpha^4+\alpha^3+\alpha)x^{20}+(\alpha^4+\alpha)x^{21}+(\alpha^5+\alpha^4)x^{22}+(\alpha^5+\alpha^4)x^{23}+(\alpha^4+\alpha^3)x^{24}+(\alpha^5+\alpha^2)x^{25}+\alpha^2x^{26}+(\alpha+1)x^{27}+(\alpha^5+1)x^{28}+(\alpha^5+\alpha^4+1)x^{29}+(\alpha^5+\alpha^4+\alpha^3+\alpha+1)x^{30}+\alpha^3x^{31}$.

Then using Reed-Solomon code, we can encode $\boldsymbol{m}$ to $\boldsymbol{c}$ as follows:

$$
\begin{aligned}
\mathbf{c} &= (g(0), g(1), g(\alpha), \ldots, g(\alpha^{62})) \\
&= (\alpha^5+\alpha^4+\alpha^3+1, \alpha^5+\alpha^4+\alpha^2+1, \alpha^3+\alpha^2+1, \alpha^2+\alpha, \alpha^5+\alpha^4+\alpha^2+\alpha+1, \\
&\quad \alpha^5+\alpha^2+\alpha, \alpha^2, \alpha^3+\alpha+1, \alpha^5+\alpha^2, \alpha^5+\alpha^3+\alpha^2+1, \alpha^3+\alpha^2+\alpha, \\
&\quad \alpha^5+\alpha^4+\alpha^2+1, \alpha^4+\alpha^2+\alpha+1, \alpha^4+\alpha^2+\alpha+1, \alpha^4+\alpha^3+\alpha+1, 0, \alpha^2+\alpha, \\
&\quad \alpha^4+\alpha^2+\alpha, \alpha^3+1, \alpha^5+1, \alpha^5+\alpha+1, \alpha^4+\alpha^3+\alpha^2+\alpha+1, \alpha^3, \alpha^2+\alpha+1, \\
&\quad \alpha^5+\alpha^4+\alpha^3, \alpha^4+\alpha+1, \alpha^5+\alpha^3+\alpha^2, \alpha^4+\alpha^3+\alpha^2+\alpha+1, \alpha^3+\alpha^2+1, \\
&\quad \alpha^2+\alpha+1, \alpha^5+\alpha^2+\alpha, \alpha^4+\alpha^3+1, \alpha^4+\alpha^3+\alpha^2+\alpha+1, \alpha^3+\alpha^2+\alpha, \\
&\quad \alpha^5+\alpha^3+\alpha+1, 1, \alpha^4+\alpha+1, \alpha^5+\alpha^3+\alpha+1, \alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha+1, \\
&\quad \alpha^3+\alpha^2+\alpha, \alpha^5+1, \alpha^2+\alpha, \alpha^4+\alpha, \alpha^5+\alpha^4+\alpha+1, \alpha^4+\alpha^3+\alpha^2, \alpha^5+\alpha+1, \\
&\quad \alpha^5+\alpha^4+\alpha^2+1, \alpha^5+\alpha^4+\alpha+1, \alpha^5+\alpha^4+\alpha^3+\alpha^2+\alpha+1, \alpha^4+\alpha^3+\alpha^2+\alpha, \\
&\quad \alpha^5+\alpha^3+\alpha^2+\alpha+1, \alpha^5+\alpha^4+\alpha+1, \alpha^5+\alpha^3+\alpha^2, \alpha^5+\alpha^4+\alpha^3+\alpha^2, \alpha^2, \\
&\quad \alpha^5+\alpha^4+\alpha^2, \alpha^5+\alpha, 1, \alpha^5+\alpha^4+\alpha^3+\alpha^2, \alpha^3+1, \alpha^2, \alpha^3+\alpha^2+\alpha, \alpha^5+\alpha^2+\alpha, \\
&\quad \alpha^5+\alpha^4+\alpha^3+\alpha^2).
\end{aligned}
\tag{5.2}
$$

This code has parameter $(64, 32, 33)$, which means the code can correct 32 erasures, or 16 errors. The code ratio is $32/64 = 1/2$. If we split the code $\boldsymbol{c}$ into 8 groups so that each group contains 8 symbols, say $\{g(0), g(1), g(\alpha), g(\alpha^2), g(\alpha^3), g(\alpha^4), g(\alpha^5), g(\alpha^6)\}$, $\{g(\alpha^7), g(\alpha^8), g(\alpha^9), g(\alpha^{10}), g(\alpha^{11}), g(\alpha^{12}), g(\alpha^{13}), g(\alpha^{14})\}$, $\{g(\alpha^{15}), g(\alpha^{16}), g(\alpha^{17}), g(\alpha^{18}), g(\alpha^{19}), g(\alpha^{20}), g(\alpha^{21}), g(\alpha^{22})\}$, $\{g(\alpha^{23}), g(\alpha^{24}), g(\alpha^{25}), g(\alpha^{26}), g(\alpha^{27}), g(\alpha^{28}), g(\alpha^{29}), g(\alpha^{30})\}$, $\{g(\alpha^{31}), g(\alpha^{32}), g(\alpha^{33}),$

$g(\alpha^{34}), g(\alpha^{35}), g(\alpha^{36}), g(\alpha^{37}), g(\alpha^{38})\}, \{g(\alpha^{39}), g(\alpha^{40}), g(\alpha^{41}), g(\alpha^{42}), g(\alpha^{43}), g(\alpha^{44}),$ $g(\alpha^{45}), g(\alpha^{46})\}, \{g(\alpha^{47}), g(\alpha^{48}), g(\alpha^{49}), g(\alpha^{50}), g(\alpha^{51}), g(\alpha^{52}), g(\alpha^{53}), g(\alpha^{54})\}, \{g(\alpha^{55}),$ $g(\alpha^{56}), g(\alpha^{57}), g(\alpha^{58}), g(\alpha^{59}), g(\alpha^{60}), g(\alpha^{61}), g(\alpha^{62})\}$, then using Lagrange interpolation, we can recover the entire data from any 4 groups if all the individual piece are available without corrupted. However, when more than 2 groups are corrupted, the message is no longer recoverable even if Reed-Solomon error-decoding algorithm is used. In other words, the corruption level cannot be higher than $2/8 = 1/4$.

5.2.2 Hermitian Code in Distributed Storage

In our preliminary research, we have developed a decoding algorithm for Hermitian code, which is designed on the curve $y^4 + y = x^5$ over the finite field $GF(2^4)$. Our decoding algorithm can correct erasures as well as errors, however, it can correct more errors than the Reed-Solomon code in the aforementioned scenario, while maintaining the existing code ratio. We will first explain Hermitian code using the notation introduced in the previous example.

Let

$$\mathcal{G}_j = \big\{\big((y^j f_j)(R_0), (y^j f_j)(R_1), \cdots, (y^j f_j)(R_{q^3-1})\big)\big\}, \tag{5.3}$$

where $k(j) = \max\{t \mid 4t + 5j \leq 32\} + 1$. R_i runs through $(\eta, \eta^5 y_0 + \beta_j)$, for $\eta \in GF(q^2)$ and $\beta_j = 0, \cdots, q-1$ are the solutions to the equation $x^q + x = 0$.

Define

$$\mathcal{H}_m = \mathcal{G}_0 \oplus \mathcal{G}_1 \oplus \mathcal{G}_2 \oplus \cdots \oplus \mathcal{G}_{q-1}, \tag{5.4}$$

then we can prove that the parameter of the above defined code is $(64, 32)$. However, we can correct more than $(n-k)/2 = (64-32)/2 = 16$ errors due to the special structure of the code.

In the following, we will present a scheme that can correct 24 errors. Since the solutions to $y^4 + y = 0$ are $\beta_0 = 0, \beta_1 = 1, \beta_2 = \alpha^5, \beta_3 = \alpha^{10}$, and $(1, \alpha)$ is in the curve, the R_i's can

be represented through the following $P_{i,j}$'s:

$$P_{0,j} = (0, \beta_j),\ P_{i,j} = (\alpha^{i-1}, \alpha^{(i-1)(q+1)+1} + \beta_j), i = 1, 2, \cdots, 15, j = 0, 1, 2, 3. \tag{5.5}$$

Consider a received vector $\boldsymbol{u} = (0, 0, 0, 0,\ \alpha, \alpha^2, \alpha^4, \alpha^5,\ \alpha^7, \alpha^9, \alpha^8, \alpha^6,\ 0, \cdots, 0,\ \alpha^5, \alpha^{10}, \alpha^4, \alpha,$ $\alpha^{11}, \alpha^{13}, \alpha, \alpha^8, \alpha^6, \alpha^5, \alpha^{10}, \alpha^7, \alpha^{14}, \alpha^2, \alpha^3, 1, 0, 0, 0, 0)$, where $\alpha^4 = \alpha + 1$. The decoding will first break $\boldsymbol{u}$ into four Reed-Solomon code:

$$\begin{aligned}
\mathbf{r}_3 &= (0, \alpha^4, \alpha^3, 0, \cdots, 0, 0, \alpha^2, \alpha^5, \alpha^2, 0), \\
\mathbf{r}_2 &= (0, \alpha^{14}, \alpha^{12}, 0, \cdots, 0, \alpha^4, \alpha^{12}, \alpha^{12}, \alpha^4, 0), \\
\mathbf{r}_1 &= (0, \alpha^5, \alpha^{13}, 0, \cdots, 0, \alpha, \alpha^7, \alpha^2, \alpha^5, 0), \\
\mathbf{r}_0 &= (0, \alpha^{10}, \alpha^{11}, 0, \cdots, 0, \alpha^{12}, \alpha^{12}, \alpha^3, \alpha^4, 0)
\end{aligned} \quad . \tag{5.6}$$

Decode $\mathbf{r}_3$ in using the decoding algorithm of Reed-Solomon codes [89], we find the error vector

$$\mathbf{e}_3 = (0, \alpha^4, \alpha^3, 0, \cdots, 0, 0, \alpha^2, \alpha^5, \alpha^2, 0) \tag{5.7}$$

for $\mathbf{r}_3$, Therefore, the error locations are $E_2, E_3, E_{13}, E_{14}, E_{15}$.

Replace the locations $E_2, E_3, E_{13}, E_{14}, E_{15}$ in $\mathbf{r}_2$ with erasures, marked as "$\otimes$", we get the vector: $(0, \otimes, \otimes, 0, \cdots, 0, \alpha, \otimes, \otimes, \otimes, 0)$.

Decode this vector and we find the error vector

$$\mathbf{e}_2 = (0, \alpha^{14}, \alpha^{12}, 0, \cdots, 0, \alpha^4, \alpha^{12}, \alpha^{12}, \alpha^4, 0) \tag{5.8}$$

with a new error location E_{12}.

Decode $\mathbf{r}_1$ and $\mathbf{r}_0$, We can find the error vectors

$$\mathbf{e}_1 = (0, \alpha^5, \alpha^{13}, 0, \cdots, 0, \alpha, \alpha^7, \alpha^2, \alpha^5, 0), \tag{5.9}$$

$$\mathbf{e}_0 = (0, \alpha^{10}, \alpha^{11}, 0, \cdots, 0, \alpha^{12}, \alpha^{12}, \alpha^3, \alpha^4, 0). \tag{5.10}$$

Now we can reconstruct the entire error vector as:

$$\begin{aligned}\boldsymbol{e} &= (0,0,0,0,\alpha,\alpha^2,\alpha^4,\alpha^5,\ \alpha^7,\alpha^9,\alpha^8,\alpha^6,\ 0,\cdots,0,\ \alpha^5,\alpha^{10},\alpha^4,\alpha,\\ &\quad \alpha^{11},\alpha^{13},\alpha,\alpha^8,\ \alpha^6,\alpha^5,\alpha^{10},\alpha^7,\ \alpha^{14},\alpha^2,\alpha^3,1,\ 0,0,0,0).\end{aligned} \tag{5.11}$$

Therefore the transmitted codeword is

$$\boldsymbol{u} = (0,0,0,\cdots,0,0). \tag{5.12}$$

For this code scheme, if we represent the bits using symbols over $GF(2^4)$, the entire message can be represented using 64 symbols. If we split the 64 symbols into groups so that each group contains 8 symbols, then when no more than 3 groups are corrupted, we can fix the corrupted groups while fixing the entire message. Therefore, we have the following claim.

Claim 1. *The error correction ratio for Hermitian code is* $24/64 = 3/8$*, which is higher than the Reed-Solomon code error correction ratio* $1/4$ *for the coding ratio* $1/2$*.*

5.2.3 Inspiration from this example

From this example, we find that the Hermitian code can correct more errors than the RS code under the same code rate. However, directly applying Hermitian code into distributed storage is a naive approach like directly applying the RS code. Thus we propose to combine the advantages of the Hermitian code and the regenerating code for distritbuted storage in the following sections.

5.3 Hermitian Code Based MSR Regenerating Code (H-MSR Code)

5.3.1 Encoding H-MSR Code

In this section, we will analyze the H-MSR code based on the MSR point with $d = 2k-2 = 2\alpha$. The code based on the MSR point with $d > 2k-2$ can be derived the same way through

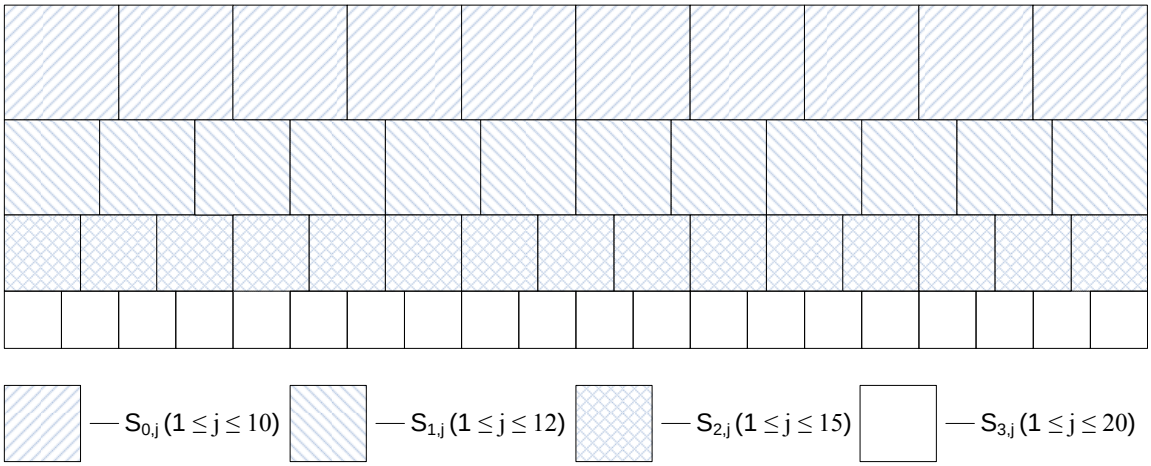

Figure 5.1 An example illustration of matrix S

truncating operations.

Let $\alpha_0, \cdots, \alpha_{q-1}$ be a strictly decreasing integer sequence satisfying $0 < \alpha_i \leq \kappa(i)$, $0 \leq i \leq q-1$, where α_i is the parameter α for the underlying regenerating code. The least common multiple of $\alpha_0, \cdots, \alpha_{q-1}$ is A. Suppose the data contains $B = A\sum_{i=0}^{q-1}(\alpha_i + 1)$ message symbols from the finite field $GF(q^2)$. In practice, if the size of the actual data is larger than B symbols, we can fragment it into blocks of size B and process each block individually.

We arrange the B symbols into two matrices S, T as below:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{q-1} \end{bmatrix}, \quad T = \begin{bmatrix} T_0 \\ T_1 \\ \vdots \\ T_{q-1} \end{bmatrix}, \tag{5.13}$$

where

$$\begin{aligned} S_i &= [S_{i,1}, S_{i,2}, \cdots, S_{i,A/\alpha_i}], \\ T_i &= [T_{i,1}, T_{i,2}, \cdots, T_{i,A/\alpha_i}]. \end{aligned} \tag{5.14}$$

$S_{i,j}$, $0 \le i \le q-1, 1 \le j \le A/\alpha_i$, is a symmetric matrix of size $\alpha_i \times \alpha_i$ with the upper-triangular entries filled by data symbols. Thus $S_{i,j}$ contains $\alpha_i(\alpha_i+1)/2$ symbols. The number of columns of each submatrix S_i, $0 \le i \le q-1$, is the same: $\alpha_i \cdot A/\alpha_i = A$. The size of matrix S is $(\sum_{i=0}^{q-1} \alpha_i) \times A$. So it contains $\sum_{i=0}^{q-1}(\alpha_i(\alpha_i+1)/2)A/\alpha_i = (A\sum_{i=0}^{q-1}(\alpha_i+1))/2$ data symbols. Figure 5.1 shows an example of matrix S for $q = 4, \alpha_0 = 6, \alpha_1 = 5, \alpha_2 = 4, \alpha_3 = 3$. In figure 5.1, the submatrix $S_{i,j}$ is represented by the square in the corresponding position with the size representing the size of the submatrix.

$T_{i,j}$ $(0 \le i \le q-1, 1 \le j \le A/\alpha_i)$ is constructed the same as $S_{i,j}$. So T has the same structure as S and contains the other $(A \cdot \sum_{i=0}^{q-1}(\alpha_i+1))/2$ data symbols.

Definition 1. *For a Hermitian code $\mathcal{H}_m$ over $GF(q^2)$, we encode matrix $M_{\dim(\mathcal{H}_m)\times A} = [M_1, M_2 \cdots, M_A]$ by encoding each column M_i, $i = 1, 2, \cdots, A$, individually using $\mathcal{H}_m$. The codeword matrix is defined as*

$$\mathcal{H}_m(M) = [\mathcal{H}_m(M_1), \mathcal{H}_m(M_2), \cdots, \mathcal{H}_m(M_A)], \tag{5.15}$$

where $\mathcal{H}_m(M_i)$ has the following form ($\varrho \in L(mQ)$):

$$[\varrho(P_{0,0}), \cdots, \varrho(P_{0,q-1}), \cdots, \varrho(P_{q^2-1,0}), \cdots, \varrho(P_{q^2-1,q-1})]^T, \tag{5.16}$$

and the elements of M_i are viewed as the coefficients of the polynomials $f_0(x), \cdots, f_{q-1}(x)$ in ϱ when M_i is encoded.

Let

$$\Phi_i = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \\ 1 & \phi & \phi^2 & \cdots & \phi^{\alpha_i-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \phi^{q^2-2} & (\phi^{q^2-2})^2 & \cdots & (\phi^{q^2-2})^{\alpha_i-1} \end{bmatrix} \tag{5.17}$$

be a Vandermonde matrix, where ϕ is the primitive element in $GF(q^2)$ mentioned in section 2.4 and $0 \leq i \leq q-1$.

Define

$$\Delta = \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{q^2-1} \end{bmatrix} \tag{5.18}$$

to be a diagonal matrix comprised of q^2 elements, where λ_i, $0 \leq i \leq q^2-1$, is chosen using the following two criteria: (i) $\lambda_i \neq \lambda_j$, $\forall i \neq j$, $0 \leq i, j \leq q^2-1$. (ii) Any $d_i = 2\alpha_i$ rows of the matrix $[\Phi_i, \Delta \cdot \Phi_i]$, $0 \leq i \leq q-1$, are linearly independent.

We also define

$$\Lambda_i = \lambda_i I \tag{5.19}$$

to be a $q \times q$ diagonal matrix for $0 \leq i \leq q^2-1$, where I is the $q \times q$ identical matrix. And

$$\Gamma = \begin{bmatrix} \Lambda_0 & 0 & \cdots & 0 \\ 0 & \Lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Lambda_{q^2-1} \end{bmatrix} \tag{5.20}$$

is a $q^3 \times q^3$ diagonal matrix formed by q^2 diagonal submatrices $\Lambda_0, \cdots, \Lambda_{q^2-1}$.

For distributed storage, we encode each pair of matrices (S, T) using Algorithm 5.1. We will name this encoding scheme as *Hermitian-MSR code encoding*, or *H-MSR code encoding*.

For H-MSR coding encoding, we have the following theorem.

Algorithm 5.1 Encoding H-MSR Code

Step 1: Encode the data matrices S, T defined in equation (5.13) using a Hermitian code $\mathcal{H}_m$ over $GF(q^2)$ with parameters $\kappa(j)$ $(0 \leq j \leq q-1)$ and m $(m \geq q^2-1)$. Denote the generated $q^3 \times A$ codeword matrices as $\mathcal{H}_m(S)$ and $\mathcal{H}_m(T)$.

Step 2: Compute the $q^3 \times A$ codeword matrix $Y = \mathcal{H}_m(S) + \Gamma\mathcal{H}_m(T)$.

Step 3: Divide Y into q^2 submatrices $Y_0, \cdots, Y_{q^2-1}$ of size $q \times A$ and store each submatrix in a storage node as shown in Figure. 5.2.

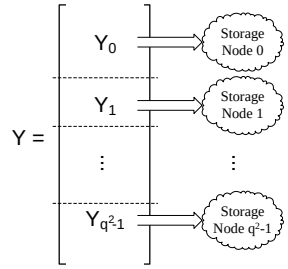

Figure 5.2 Illustration of storing the codeword matrices in distributed storage nodes

Theorem 5.1. *The H-MSR code encoding described in Algorithm 5.1 can achieve the MSR point in distributed storage.*

Proof. We first study the structure of the codeword matrix $\mathcal{H}_m(S)$. Since every column of the matrix is an independent Hermitian codeword, we can decode the first column $\mathbf{h} = [h_{0,0}, \cdots, h_{0,q-1}, \cdots, h_{q^2-1,0}, \cdots, h_{q^2-1,q-1}]^T$ as an example without loss of generality. We arrange the q^3 rational points of the Hermitian curve following the order in Table 2.1. In the table, we can find that for each i, $i = 0, 1, \cdots, q^2-1$, the rational points $P_{i,0}, P_{i,1}, \cdots, P_{i,q-1}$ all have the same first coordinate.

Suppose $\varrho \in L(mQ)$: $\varrho(P_{i,l}) = f_0(P_{i,l}) + y(P_{i,l})f_1(P_{i,l}) + \cdots + (y(P_{i,l}))^{q-1}f_{q-1}(P_{i,l})$, $0 \leq i \leq q^2-1$, $0 \leq l \leq q-1, \deg f_j(x) = \alpha_j - 1$ for $0 \leq j \leq q-1$. Since $P_{i,0}, P_{i,1}, \cdots, P_{i,q-1}$ all have the same first coordinate and $f_j(P_{i,l})$ is only applied to the first coordinate of $P_{i,l}$, we have $f_j(P_{i,l}) = f_j(\phi^{s_i})$, $s_0 = -\infty, s_i = i-1, \text{for } i \geq 1, \phi^{-\infty} = 0$, which does not depend

on l. Therefore, we can derive q^2 sets of equations for $0 \leq i \leq q^2 - 1$:

$$\begin{cases} f_0(\phi^{s_i}) + y(P_{i,0}) f_1(\phi^{s_i}) + \cdots + (y(P_{i,0}))^{q-1} f_{q-1}(\phi^{s_i}) = h_{i,0} \\ f_0(\phi^{s_i}) + y(P_{i,1}) f_1(\phi^{s_i}) + \cdots + (y(P_{i,1}))^{q-1} f_{q-1}(\phi^{s_i}) = h_{i,1} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ f_0(\phi^{s_i}) + y(P_{i,q-1}) f_1(\phi^{s_i}) + \cdots + (y(P_{i,q-1}))^{q-1} f_{q-1}(\phi^{s_i}) = h_{i,q-1} \end{cases} . \tag{5.21}$$

If we store the codeword matrix in storage nodes according to Figure. 5.2, each set of the equations corresponds to a storage node. As we mentioned above, the set of equations in equation (5.21) can be derived in storage node i.

Since the coefficient matrix B_i is a Vandermonde matrix:

$$B_i = \begin{bmatrix} 1 & y(P_{i,0}) & \cdots & y(P_{i,0})^{q-1} \\ 1 & y(P_{i,1}) & \cdots & y(P_{i,1})^{q-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & y(P_{i,q-1}) & \cdots & y(P_{i,q-1})^{q-1} \end{bmatrix} . \tag{5.22}$$

we can solve $\mathbf{u}_i = [f_0(\phi^{s_i}), f_1(\phi^{s_i}), \cdots, f_{q-1}(\phi^{s_i})]^T$ from

$$\mathbf{u}_i = B_i^{-1} \mathbf{h}_i, \tag{5.23}$$

where $\mathbf{h}_i = [h_{i,0}, h_{i,1}, \cdots, h_{i,q-1}]^T$.

From all the q^2 storage nodes, we can get vectors $\mathcal{F}_i = [f_i(0), f_i(1), \cdots, f_i(\phi^{q^2-2})]^T$, $i = 0, \cdots, q-1$, which can be viewed as extended Reed-Solomon codes.

Now consider all the columns of $\mathcal{H}_m(S)$, we can get the following equation:

$$\Phi_i S_{i,j} = F_{i,j}, \tag{5.24}$$

where $F_{i,j} = [\mathcal{F}_i^{(1)}, \cdots, \mathcal{F}_i^{(\alpha_i)}]$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, and $\mathcal{F}_i^{(l)}$ corresponds to the l^{th} column of the submatrix $S_{i,j}$.

Next we will consider the structure of the codeword matrix $\mathcal{H}_m(T)$. Because the encoding process for $\mathcal{H}_m(T)$ is the same as that of $\mathcal{H}_m(S)$, for $\Gamma\mathcal{H}_m(T)$, we can derive

$$\Delta\Phi_i T_{i,j} = \Delta E_{i,j}, \tag{5.25}$$

where $\mathcal{E}_i = [e_i(0), e_i(1), \cdots, e_i(\phi^{q^2-2})]^T$, $E_{i,j} = [\mathcal{E}_i^{(1)}, \cdots, \mathcal{E}_i^{(\alpha_i)}]$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, and $\mathcal{E}_i^{(l)}$ corresponds to the l^{th} column of the submatrix $T_{i,j}$.

Thirdly, we will study the optimality of the code in the sense of the MSR point. For $\Phi_i S_{i,j} + \Delta\Phi_i T_{i,j}$, $0 \leq i \leq q-1, 1 \leq j \leq A/\alpha_i$, since $S_{i,j}, T_{i,j}$ are symmetric and satisfy the requirements for MSR point according to [57] with parameters $d = 2\alpha_i, k = \alpha_i + 1, \alpha = \alpha_i, \beta = 1, B = \alpha_i \cdot (\alpha_i + 1)$. By encoding S, T using $\mathcal{H}_m(S) + \Gamma\mathcal{H}_m(T)$ and distributing $Y_0, \cdots, Y_{q^2-1}$ into q^2 storage nodes, each row of the matrix $\Phi_i S_{i,j} + \Delta\Phi_i T_{i,j}$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, can be derived in a corresponding storage node. Because $\Phi_i S_{i,j} + \Delta\Phi_i T_{i,j}$ achieves the MSR point, data related to matrices $S_{i,j}, T_{i,j}$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, can be regenerated at the MSR point. Therefore, Algorithm 5.1 can achieve the MSR point. □

5.3.2 Regeneration of the H-MSR Code in the Error-free Network

In this section, we will discuss the regeneration for the H-MSR code in the error-free network. Let $\mathbf{v}_i = [e_0(\phi(^{s_i})), e_1(\phi(^{s_i})), \cdots, e_{q-1}(\phi(^{s_i}))]^T$, then

$$\mathbf{u}_i + \Lambda_i \mathbf{v}_i = B_i^{-1}\mathbf{y}_i = [f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}), \cdots, f_{q-1}(\phi^{s_i}) + \lambda_i e_{q-1}(\phi^{s_i})]^T, \tag{5.26}$$

for every column $\mathbf{y}_i$ of Y_i.

The main idea of the regeneration algorithms is to regenerate $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$, $0 \leq l \leq q-1$, by downloading help symbols from $d_l = 2\alpha_l$ nodes, where d_l represents the regeneration parameter d for $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$ in the H-MSR code regeneration.

Suppose node z fails, we devise Algorithm 5.2 in the network to regenerate the exact H-MSR code symbols of node z in a replacement node z'. For convenience, we suppose $d_q = 2\alpha_q = 0$ and define

$$\mathbf{V}_{i,j,l} = \begin{bmatrix} \nu_{i,l} \\ \nu_{i+1,l} \\ \vdots \\ \nu_{j,l} \end{bmatrix}, \tag{5.27}$$

where $\nu_{t,l}$, $i \leq t \leq j$, is the t^{th} row of $[\Phi_l, \Delta\Phi_l]$. Each node i, $0 \leq i \leq q^2 - 1$, only stores its own encoding vector $\nu_{i,l}$, $0 \leq l \leq q - 1$.

First, replacement node z' will send requests to helper nodes for regeneration: z' sends the integer j to $d_j - d_{j+1}$ helper nodes, to which z' has not sent requests before, for every j from $q - 1$ to 0 in descending order.

Upon receiving the request integer j, helper node i will calculate and send the help symbols as follows: node i will first calculate $\widetilde{Y}_i = B_i^{-1} Y_i$ to remove the coefficient matrix B_i from the codeword matrix. Since the $(l+1)^{th}$ row of $\widetilde{Y}_i$ corresponds to the symbols related to $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$, for $0 \leq l \leq j$, node i will divide the $(l+1)^{th}$ row of $\widetilde{Y}_i$ into A/α_l row vectors of the size $1 \times \alpha_l$: $[\tilde{\mathbf{y}}_{i,l,1}, \tilde{\mathbf{y}}_{i,l,2}, \cdots, \tilde{\mathbf{y}}_{i,l,A/\alpha_l}]$. Then for every $0 \leq l \leq j$ and $1 \leq t \leq A/\alpha_l$, node i will calculate the help symbol $\tilde{p}_{i,l,t} = \tilde{\mathbf{y}}_{i,l,t} \mu_{z,l}^T$, where $\mu_{z,l}$ is the z^{th} row of the encoding matrix Φ_l defined in equation (6.2). At last, node i will send out all the calculated symbols $\tilde{p}_{i,l,t}$. Here j indicates that z' is requesting symbols $\tilde{p}_{i,l,t}$, $0 \leq l \leq j$ and $1 \leq t \leq A/\alpha_l$, calculated by $[f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}), \cdots, f_j(\phi^{s_i}) + \lambda_i e_j(\phi^{s_i})]^T$

Since $d_{l_1} > d_{l_2}$ for $l_1 < l_2$, for efficiency consideration, only d_{q-1} helper nodes need to send out symbols $\tilde{p}_{i,l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, calculated by $[f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}), f_1(\phi^{s_i}) + \lambda_i e_1(\phi^{s_i}), \cdots, f_{q-1}(\phi^{s_i}) + \lambda_i e_{q-1}(\phi^{s_i})]^T$. Then $d_j - d_{j+1}$ nodes only need to send out symbols $\tilde{p}_{i,l,t}$, $0 \leq l \leq j$ and $1 \leq t \leq A/\alpha_l$, calculated by $[f_0(\phi^{s_i}) +$

$\lambda_i e_0(\phi^{s_i}), f_1(\phi^{s_i}) + \lambda_i e_1(\phi^{s_i}), \cdots, f_j(\phi^{s_i}) + \lambda_i e_j(\phi^{s_i})]^T$ for $0 \leq j \leq q-2$. In this way, the total number of helper nodes that send out symbols $\tilde{p}_{i,l,t}$, $1 \leq t \leq A/\alpha_l$, calculated by $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$ is $d_{q-1} + \sum_{j=l}^{q-2}(d_j - d_{j+1}) = d_l$.

When the replacement node z' receives all the requested symbols, it can regenerate the symbols stored in the failed node z using the following algorithm:

Algorithm 5.2 z' Regenerates Symbols of the Failed Node z

Step 1: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, calculate the regenerated symbols related to the help symbols $\tilde{p}_{i,l,t}$ from d_l helper nodes. Without loss of generality, we assume $0 \leq i \leq d_l - 1$:

Step 1.1: Let $\mathbf{p} = [\tilde{p}_{0,l,t}, \tilde{p}_{1,l,t}, \cdots, \tilde{p}_{d_l-1,l,t}]^T$, solve the equation: $\mathbf{V}_{0,d_l-1,l}\mathbf{x} = \mathbf{p}$.

Step 1.2: Since $\mathbf{x} = \begin{bmatrix} S_{l,t} \\ T_{l,t} \end{bmatrix} \mu_{z,l}^T$ and $S_{l,t}, T_{l,t}$ are symmetric, we can calculate $\mathbf{x}^T = [\mu_{z,l} S_{l,t}, \mu_{z,l} T_{l,t}]$.

Step 1.3: Compute $\tilde{\mathbf{y}}_{z,l,t} = \mu_{z,l} S_{l,t} + \lambda_z \mu_{z,l} T_{l,t} = \nu_{z,l} \begin{bmatrix} S_{l,t} \\ T_{l,t} \end{bmatrix}$.

Step 2: Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z \widetilde{Y}_z$.

From Algorithm 5.2, we can derive the equivalent storage parameters for each symbol block of size $B_j = A(\alpha_j + 1)$: $d = 2\alpha_j$, $k = \alpha_j + 1$, $\alpha = A$, $\beta = A/\alpha_j$, $0 \leq j \leq q-1$ and equation (2.7) of the MSR point holds for these parameters. Theorem 5.1 guarantees that Algorithm 5.2 can achieve the MSR point for data regeneration of the H-MSR code.

5.3.3 Regeneration of the H-MSR Code in the Hostile Network

In hostile network, Algorithm 5.2 may not be able to regenerate the failed node due to possible bogus symbols received from the responses. In fact, even if the replacement node

z' can derive the symbol matrix $Y_{z'}$ using Algorithm 5.2, it cannot verify the correctness of the result.

There are two modes for the helper nodes to regenerate the contents of a failed storage node in hostile network. One mode is the detection mode, in which no error has been found in the symbols received from the helper nodes. Once errors are detected, the recovery mode will be used to correct the errors and locate the malicious nodes.

5.3.3.1 Detection Mode

In the detection mode, the replacement node z' will send requests in the way similar to that of the error-free network in Section 5.3.2. The only difference is that when $j = q - 1$, z' sends requests to $d_{q-1} - d_q + 1$ nodes instead of $d_{q-1} - d_q$ nodes. Helper nodes will still use the way similar to that of the error-free network in Section 5.3.2 to send the help symbols. The regeneration algorithm is described in Algorithm 5.3 with the detection probability characterized in Theorem 5.2.

Lemma 1. *Suppose* $e_0, \cdots, e_{d_l}$ *are the* $d_l + 1$ *errors* $e_{0,l,t}, \cdots, e_{d_l,l,t}$ *in Algorithm 5.3,* $\hat{\mathbf{x}}_1 = \mathbf{V}_{0,d_l-1,l}^{-1} \cdot [e_0, \cdots, e_{d_l-1}]^T$ *and* $\hat{\mathbf{x}}_2 = \mathbf{V}_{1,d_l,l}^{-1} \cdot [e_1, \cdots, e_{d_l}]^T$. *When the number of malicious nodes in the* $d_l + 1$ *helper nodes of Algorithm 5.3 is less than* $d_l + 1$, *the probability that* $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$ *is at most* $1/q^2$.

Proof. Since $\mathbf{V}_{0,d_l-1,l}$ and $\mathbf{V}_{1,d_l,l}$ are full rank matrices, we can get their corresponding inverse matrices. $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$ is equivalent to $\mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_1 = \mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_2$.

First, we have

$$\mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_1 = [e_0, e_1, \cdots, e_{d_l-1}]^T. \tag{5.28}$$

Algorithm 5.3 [Detection Mode] z' Regenerates Symbols of the Failed Node z in Hostile Network

Step 1: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, we can calculate the regenerated symbols which are related to the help symbols $\tilde{p}'_{i,l,t}$ from d_l helper nodes. $\tilde{p}'_{i,l,t} = \tilde{p}_{i,l,t} + e_{i,l,t}$ is the response from the i^{th} helper node. If $\tilde{p}_{i,l,t}$ has been modified by the malicious node i, we have $e_{i,l,t} \in GF(q^2)\backslash\{0\}$. Otherwise we have $e_{i,l,t} = 0$. To detect whether there are errors, we will calculate symbols from two sets of helper nodes then compare the results. (Without loss of generality, we assume $0 \leq i \leq d_l$.)

Step 1.1: Let $\mathbf{p}_1{}' = [\tilde{p}'_{0,l,t}, \tilde{p}'_{1,l,t}, \cdots, \tilde{p}'_{d_l-1,l,t}]^T$, where the symbols are collected from node 0 to node $d_l - 1$, solve the equation $\mathbf{V}_{0,d_l-1,l}\mathbf{x}_1 = \mathbf{p}_1{}'$.

Step 1.2: Let $\mathbf{p}_2{}' = [\tilde{p}'_{1,l,t}, \tilde{p}'_{2,l,t}, \cdots, \tilde{p}'_{d_l,l,t}]^T$, where the symbols are collected from node 1 to node d_l, solve the equation $\mathbf{V}_{1,d_l,l}\mathbf{x}_2 = \mathbf{p}_2{}'$.

Step 1.3: Compare $\mathbf{x}_1$ with $\mathbf{x}_2$. If they are the same, compute $\tilde{\mathbf{y}}_{z,l,t} = \mu_{z,l}S_{l,t} + \lambda_z\mu_{z,l}T_{l,t}$ as described in Algorithm 5.2. Otherwise, errors are detected in the help symbols. Exit the algorithm and switch to recovery regeneration mode.

Step 2: No error has been detected for the calculating of the regeneration so far. Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z\widetilde{Y}_z$.

Suppose $\mathbf{V}^{-1}_{1,d_l,l} = [\eta_0, \eta_1, \cdots, \eta_{d_l-1}]$, then we have:

$$\nu_{r,l} \cdot \eta_s = \begin{cases} 1, & r = s+1 \\ 0, & r \neq s+1 \end{cases}, \ 1 \leq r \leq d_l, 0 \leq s \leq d_l - 1. \tag{5.29}$$

$$\begin{aligned} \mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_2 &= \mathbf{V}_{0,d_l-1,l} \cdot \mathbf{V}^{-1}_{1,d_l,l} \cdot [e_1, e_2, \cdots, e_{d_l}]^T \\ &= \mathbf{V}_{0,d_l-1,l} \cdot [\eta_0, \eta_1, \cdots, \eta_{d_l-1}] \cdot [e_1, e_2, \cdots, e_{d_l}]^T \\ &= [x_{2,0}, e_1, \cdots, e_{d_l-1}]^T. \end{aligned} \tag{5.30}$$

To calculate $x_{2,0}$, we first derive the expression of $\nu_{0,l}$. Because $\nu_{1,l}, \nu_{2,l}, \cdots, \nu_{d_l,l}$ are linearly independent, they can be viewed as a set of bases of the d_l dimensional linear space.

So we have

$$\nu_{0,l} = \sum_{r=1}^{r=d_l} \zeta_r \cdot \nu_{r,l}, \tag{5.31}$$

where $\zeta_r \neq 0$, $r = 1, \cdots, d_l$, because any d_l vectors out of $\nu_{0,l}, \nu_{1,l}, \cdots, \nu_{d_l,l}$ are linearly independent. Then

$$\begin{aligned} x_{2,0} &= \left(\sum_{r=1}^{r=d_l} \zeta_r \cdot \nu_{r,l} \right) [\eta_0, \eta_1, \cdots, \eta_{d_l-1}][e_1, e_2, \cdots, e_{d_l}]^T \\ &= \sum_{r=1}^{r=d_l} \zeta_r \cdot e_r. \end{aligned} \tag{5.32}$$

If

$$e_0 = \sum_{r=1}^{r=d_l} \zeta_r \cdot e_r, \tag{5.33}$$

then $\mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_1 = \mathbf{V}_{0,d_l-1,l} \cdot \hat{\mathbf{x}}_2$ and $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$.

When only one element of $e_0, e_1, \cdots, e_{d_l}$ is nonzero, since $\zeta_1, \cdots, \zeta_{d_l}$ are all nonzero, equation (5.33) will never hold. In this case, the probability is 0. When there are more than one nonzero elements, it means there are more than one malicious nodes. If the number of malicious nodes is less than $d_l + 1$, they will not be able to collude to solve the coefficients ζ_r in (5.31). The probability that equation (5.33) holds will be $1/q^2$. □

Theorem 5.2. *When the number of malicious nodes in the $d_l + 1$ helper nodes of Algorithm 5.3 is less than $d_l + 1$, the probability for the bogus symbols sent from the malicious nodes to be detected is at least $1 - 1/q^2$.*

Proof. Since $\mathbf{V}_{0,d_l-1,l}$ and $\mathbf{V}_{1,d_l,l}$ are full rank matrices, $\mathbf{x}_1$ can be calculated by (For convenience, use e_i to represent $e_{i,l,t}$):

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{V}_{0,d_l-1,l}^{-1} \cdot \left[\tilde{p}_{0,l,t} + e_0, \cdots, \tilde{p}_{d_l-1,l,t} + e_{d_l-1} \right]^T \\ &= \mathbf{x} + \mathbf{V}_{0,d_l-1,l}^{-1} \cdot [e_0, e_1, \cdots, e_{d_l-1}]^T \\ &= \mathbf{x} + \hat{\mathbf{x}}_1. \end{aligned} \tag{5.34}$$

$\mathbf{x}_2$ can be calculated the same way:

$$\mathbf{x}_2 \;=\; \mathbf{x} + \mathbf{V}_{1,d_l,l}^{-1} \cdot [e_1, e_2, \cdots, e_{d_l}]^T = \mathbf{x} + \hat{\mathbf{x}}_2. \tag{5.35}$$

If $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$, Algorithm 5.3 will fail to detect the errors. So we will focus on the relationship between $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$. According to Lemma 1, when the number of malicious nodes in the $d_l + 1$ helper nodes is less than $d_l + 1$, the probability that $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$ is at most $1/q^2$. So the probability that $\mathbf{x}_1 \neq \mathbf{x}_2$, equivalently the detection probability, is at least $1 - 1/q^2$. □

5.3.3.2 Recovery Mode

Once the replacement node z' detects errors using Algorithm 5.3, it will send integer $j = q-1$ to all the other $q^2 - 1$ nodes in the network requesting help symbols. Helper node i will send help symbols similar to Section 5.3.2. z' can regenerate symbols using Algorithm 5.4.

5.3.4 Reconstruction of the H-MSR Code in the Error-free Network

Here we will discuss the reconstruction of the H-MSR code in the error-free network. The main idea of the reconstruction algorithms is to reconstruct $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$, $0 \leq l \leq q-1$, by downloading help symbols from $k_l = \alpha_l + 1$ nodes, where k_l is used to represent the reconstruction parameter k for $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$ in the H-MSR code reconstruction. We devise Algorithm 5.5 in the network for the data collector DC to reconstruct the original file. For convenience, we suppose $\alpha_q = 0$.

First, DC will send requests to the storage nodes for reconstruction: DC sends integer j to $k_j - k_{j+1}$ helper nodes, to which DC has not sent requests before, for every j from $q-1$ to 0 in descending order.

Upon receiving the request integer j, node i will calculate and send symbols as follows: first node i will calculate $\tilde{Y_i} = B_i^{-1} Y_i$ to remove the coefficient matrix B_i from the codeword matrix. Since the $(l+1)^{th}$ row of $\tilde{Y_i}$ corresponds to the symbols related to $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$, for $0 \leq l \leq j$, node i will send out the $(l+1)^{th}$ row of $\tilde{Y_i}$: $\tilde{\mathbf{y}}_{i,l}$. Here j indicates that DC

Algorithm 5.4 [Recovery Mode] z' Regenerates Symbols of the Failed Node z in Hostile Network

Step 1: For every $q-1 \geq l \geq 0$ in descending order and $1 \leq t \leq A/\alpha_l$ in ascending order, we can regenerate the symbols when the errors in the received help symbols $\tilde{p}'_{i,l,t}$ from q^2-1 helper nodes can be corrected. Without loss of generality, we assume $0 \leq i \leq q^2-2$.

Step 1.1: Let $\mathbf{p'} = [\tilde{p}'_{0,l,t}, \tilde{p}'_{1,l,t}, \cdots, \tilde{p}'_{q^2-2,l,t}]^T$. Since $\mathbf{V}_{0,q^2-2,l} \cdot \mathbf{x} = \mathbf{p'}$, $\mathbf{p'}$ can be viewed as an MDS code with parameters (q^2-1, d_l, q^2-d_l).

Step 1.2: Substitute $\tilde{p}'_{i,l,t}$ in $\mathbf{p'}$ with the symbol $\otimes$ representing an erasure if node i has been detected to be corrupted in the previous loops (previous values of l, t).

Step 1.3: If the number of erasures in $\mathbf{p'}$ is larger than $\min\{q^2-d_l-1, \lfloor (q^2-d_{q-1}-1)/2 \rfloor\}$, then the number of errors have exceeded the error correction capability. So here we will flag the decoding failure and exit the algorithm.

Step 1.4: Since the number of errors is within the error correction capability of the MDS code, decode $\mathbf{p'}$ to $\mathbf{p}'_{cw}$ and solve $\mathbf{x}$.

Step 1.5: If the i^{th} position symbols of $\mathbf{p}'_{cw}$ and $\mathbf{p'}$ are different, mark node i as corrupted.

Step 1.6: Compute $\tilde{\mathbf{y}}_{z,l,t} = \mu_{z,l} \cdot S_{l,t} + \lambda_z \cdot \mu_{z,l} \cdot T_{l,t}$ as described in Algorithm 5.2.

Step 2: Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z \widetilde{Y}_z$.

---

is requesting symbols of $\tilde{\mathbf{y}}_{i,l}$, $0 \leq l \leq j$, calculated by $[f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}), \cdots, f_j(\phi^{s_i}) + \lambda_i e_j(\phi^{s_i})]^T$.

Since $k_{l_1} > k_{l_2}$ for $l_1 < l_2$, for efficiency consideration, only k_{q-1} helper nodes need to send out symbols of $\tilde{\mathbf{y}}_{i,l}$, $0 \leq l \leq q-1$, calculated by $[f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}),\ f_1(\phi^{s_i}) + \lambda_i e_1(\phi^{s_i}), \cdots,\ f_{q-1}(\phi^{s_i}) + \lambda_i e_{q-1}(\phi^{s_i})]^T$. Then $k_j - k_{j+1}$ nodes only need to send out symbols of $\tilde{\mathbf{y}}_{i,l}$, $0 \leq l \leq j$, calculated by $[f_0(\phi^{s_i}) + \lambda_i e_0(\phi^{s_i}), f_1(\phi^{s_i}) + \lambda_i e_1(\phi^{s_i}), \cdots, f_j(\phi^{s_i}) + \lambda_i e_j(\phi^{s_i})]^T$ for $0 \leq j \leq q-2$. In this way, the total number of helper nodes that send out symbols of $\tilde{\mathbf{y}}_{i,l}$ calculated by $f_l(\phi^{s_i}) + \lambda_i e_l(\phi^{s_i})$ is $k_{q-1} + \sum_{j=l}^{q-2}(k_j - k_{j+1}) = k_l$.

When DC receives all the requested symbols, it can reconstruct the original file using the

following algorithm:

Algorithm 5.5 DC Reconstructs the Original File

Step 1: For every $0 \leq l \leq q-1$, divide the response symbol vector $\tilde{\mathbf{y}}_{i,l}$ from the i^{th} node into A/α_l equal row vectors: $[\tilde{\mathbf{y}}_{i,l,1}, \tilde{\mathbf{y}}_{i,l,2}, \cdots, \tilde{\mathbf{y}}_{i,l,A/\alpha_l}]$, $0 \leq i \leq k_l - 1$.

Step 2: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, DC reconstructs the matrices related to the original file:

Step 2.1: Let $R = [\tilde{\mathbf{y}}_{0,l,t}^T, \tilde{\mathbf{y}}_{1,l,t}^T, \cdots, \tilde{\mathbf{y}}_{k_l-1,l,t}^T]^T$, we have the equation: $\mathbf{V}_{0,k_l-1,l} \cdot \begin{bmatrix} S_{l,t} \\ T_{l,t} \end{bmatrix} = R$ according to the encoding algorithm.

Step 2.2: DC reconstructs $S_{l,t}, T_{l,t}$ using techniques similar to [57].

Step 3: DC reconstructs the original file from all the matrices $S_{l,t}, T_{l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$.

5.3.5 Reconstruction of the H-MSR Code in the Hostile Network

Similar to the regeneration algorithms, the reconstruction algorithms in error-free network do not work in hostile network. Even if the data collecter can calculate the symbol matrices S, T using Algorithm 5.5, it cannot verify whether the result is correct or not. There are two modes for the original file to be reconstructed in hostile network. One mode is the detection mode, in which no error has been found in the symbols received from the storage nodes. Once errors are detected in the detection mode, the recovery mode will be used to correct the errors and locate the malicious nodes.

5.3.5.1 Detection Mode

In the detection mode, DC will send requests in the way similar to that for the error-free network in Section 5.3.4. The only difference is that when $j = q-1$, DC will send requests to $k_{q-1} - k_q + 1$ nodes instead of $k_{q-1} - k_q$ nodes. Storage nodes will still use the way similar to

that for the error-free network in Section 5.3.4 to send symbols. The reconstruction algorithm is described in Algorithm 5.6 with the detection probability described in Theorem 5.3.

Algorithm 5.6 [Detection mode] DC Reconstructs the Original File in Hostile Network

Step 1: For every $0 \leq l \leq q-1$, we can divide the symbol vector $\tilde{\mathbf{y}}'_{i,l}$ into A/α_l equal row vectors: $[\tilde{\mathbf{y}}'_{i,l,1}, \tilde{\mathbf{y}}'_{i,l,2}, \cdots, \tilde{\mathbf{y}}'_{i,l,A/\alpha_l}]$. $\tilde{\mathbf{y}}'_{i,l} = \tilde{\mathbf{y}}_{i,l} + \mathbf{e}_{i,l}$ is the response from the i^{th} storage node. If $\tilde{\mathbf{y}}_{i,l}$ has been modified by the malicious node i, we have $\mathbf{e}_{i,l} \in (GF(q^2))^A \backslash \{\mathbf{0}\}$. To detect whether there are errors, we will reconstruct the original file from two sets of storage nodes then compare the results. (Without loss of generality, we assume $0 \leq i \leq k_l$.)

Step 2: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, DC can reconstruct the matrices related to the original file:

Step 2.1: Let $R' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{k_l,l,t}]^T$.

Step 2.2: Let $R_1{}' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{\alpha_l,l,t}]^T$, which are the symbols collected from node 0 to node $k_l - 1 = \alpha_l$, then we have $\mathbf{V}_{0,\alpha_l,l} \cdot \begin{bmatrix} S_1 \\ T_1 \end{bmatrix} = R_1{}'$. Solve S_1, T_1 using the method same to algorithm 5.5.

Step 2.3: Let $R_2{}' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{\alpha_l-1,l,t}, \tilde{\mathbf{y}}'^T_{\alpha_l+1,l,t}]^T$, which are the symbols collected from node 0 to node $k_l = \alpha_l + 1$ except node α_l, and $\Psi_{DC2} = \begin{bmatrix} \nu_{0,l} \\ \vdots \\ \nu_{\alpha_l-1,l} \\ \nu_{\alpha_l+1,l} \end{bmatrix}$, then we have $\Psi_{DC2} \cdot \begin{bmatrix} S_2 \\ T_2 \end{bmatrix} = R_2{}'$. Solve S_2, T_2 using the method same to algorithm 5.5.

Step 2.4: Compare $[S_1, T_1]$ with $[S_2, T_2]$. If they are the same, let $[S_{l,t}, T_{l,t}] = [S_1, T_1]$. Otherwise, errors are detected in the received symbols. Exit the algorithm and switch to recovery reconstruction mode.

Step 3: No error has been detected for the calculating of the reconstruction so far. So DC can reconstruct the original file from all the matrices $S_{l,t}, T_{l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$.

Theorem 5.3. *When the number of malicious nodes in the $k_l + 1$ nodes of Algorithm 5.6 is less than $k_l + 1$, the probability for the bogus symbols sent from the malicious nodes to be detected is at least $1 - (1/q^2)^{2(\alpha_l - 2)}$.*

Proof. We arrange this proof as follows. We will first study the requirements for $S_1 = S_2, T_1 = T_2$ in Algorithm 5.6 which will lead to the failure of the Algorithm when there are bogus symbols. Then we will study the corresponding failure probabilities depending on different values of λ_i of the matrix Δ defined in section 5.3.1.

For convenience we write $\mathbf{e}_{i,l,t}$ as $\mathbf{e}_i$ in the proof. $\mathbf{e}_i \in [GF(q^2)]^{\alpha_l}$ for $0 \leq i \leq \alpha_l + 1$. We also write $\Psi_{DC} = [\Phi_{DC}, \Delta_{DC} \cdot \Phi_{DC}]$, where $\Phi_{DC} = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_{k_l - 1} \end{bmatrix}$ and μ_i represents $\mu_{i,l}$ which is the i^{th} row of the encoding matrix Φ_l defined in section 5.3.1.

Step 1. Derive the requirements For ${R_1}' = R_1 + W_1$ in Algorithm 5.6, we have:

$$\Phi_{DC1} S_1 \Phi_{DC1}^T + \Delta_{DC1} \Phi_{DC1} T_1 \Phi_{DC1}^T = R_1 \Phi_{DC1}^T + W_1 \Phi_{DC1}^T, \tag{5.36}$$

where $\Phi_{DC1} = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_{\alpha_l} \end{bmatrix}$, $W_1 = \begin{bmatrix} \mathbf{e}_0 \\ \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_{\alpha_l} \end{bmatrix}$. Suppose $C_1 = \Phi_{DC1} S_1 \Phi_{DC1}^T, D_1 = \Phi_{DC1} T_1 \Phi_{DC1}^T$, we can write equation (5.36) as:

$$C_1 + \Delta_{DC1} D_1 = R_1 \Phi_{DC1}^T + W_1 \Phi_{DC1}^T = \hat{R}_1 + \hat{W}_1. \tag{5.37}$$

It is easy to see that C_1 and D_1 are symmetric, so we have

$$\begin{cases} C_{1,i,j} + \lambda_i \cdot D_{1,i,j} = \hat{R}_{1,i,j} + \hat{W}_{1,i,j} \\ C_{1,i,j} + \lambda_j \cdot D_{1,i,j} = \hat{R}_{1,j,i} + \hat{W}_{1,j,i} \end{cases}, \tag{5.38}$$

where $C_{1,i,j}, D_{1,i,j}, \hat{R}_{1,i,j}, \hat{W}_{1,i,j}$ are the elements in the i^{th} row, j^{th} column of $C_1, D_1, \hat{R}_1, \hat{W}_1$ respectively. Solve equation (5.38) for all the i, j ($i \neq j, 0 \leq i \leq \alpha_l, 0 \leq j \leq \alpha_l - 1$), we can get the corresponding $C_{1,i,j}, D_{1,i,j}$. Because the structure of C_1 and D_1 are the same, we will only focus on C_1 (corresponding to S_1) in the proof. The calculation for D_1 (corresponding to T_1) is the same.

$$\Phi_{DC1} S_1 \Phi_{DC1}^T = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_{\alpha_l} \end{bmatrix} \cdot S_1 \cdot [\mu_0^T, \mu_1^T, \cdots, \mu_{\alpha_l}^T] = C_1. \tag{5.39}$$

So the elements of the i^{th} row of C_1 (except the element in the diagonal position) can be written as:

$$\mu_i \cdot S_1 \cdot [\mu_0^T, \cdots, \mu_{i-1}^T, \mu_{i+1}^T \cdots, \mu_{\alpha_l}^T] = [C_{1,i,0}, \cdots, C_{1,i,i-1}, C_{1,i,i+1}, \cdots, C_{1,i,\alpha_l}]. \tag{5.40}$$

Let $\Omega = \begin{bmatrix} \mu_0 \\ \mu_1 \\ \vdots \\ \mu_{\alpha_l - 1} \end{bmatrix}$, then Ω is an $\alpha_l \times \alpha_l$ full rank matrix, and we can derive S_1 from

$$\Omega \cdot S_1 = \begin{bmatrix} [C_{1,0,1}, C_{1,0,2}, \cdots, C_{1,0,\alpha_l}][\mu_1^T, \mu_2^T, \cdots, \mu_{\alpha_l}^T]^{-1} \\ [C_{1,1,0}, C_{1,1,2}, \cdots, C_{1,1,\alpha_l}][\mu_0^T, \mu_2^T, \cdots, \mu_{\alpha_l}^T]^{-1} \\ \cdots \\ [C_{1,\alpha_l-1,0}, C_{1,\alpha_l-1,1}, \cdots, C_{1,\alpha_l-1,\alpha_l}][\mu_0^T, \mu_1^T, \cdots, \mu_{\alpha_l}^T]^{-1} \end{bmatrix}. \quad (5.41)$$

For $R_2{}' = R_2 + W_2$ in Algorithm 5.6, we can get $\Omega \cdot S_2$ the same way. If $\Omega \cdot S_1 = \Omega \cdot S_2$, Algorithm 5.6 will fail to detect the errors. This will happen if all the rows of $\Omega \cdot S_1$ and $\Omega \cdot S_2$ are the same. So we will focus on the i^{th} row of $\Omega \cdot S_1$ and $\Omega \cdot S_2$.

Step 2. Calculate the failure probabilities Depending on the values of λ_i, we discuss two cases:

(a) If none of the λ_i $(0 \leq i \leq \alpha_l)$ equals to 0, we can solve $C_{1,i,j}$ in equation (5.38):

$$\begin{aligned} C_{1,i,j} &= \frac{\lambda_j \cdot \hat{R}_{1,i,j} - \lambda_i \cdot \hat{R}_{1,j,i}}{\lambda_i \cdot \lambda_j} + \frac{\mathbf{e}_i \cdot \mu_j^T}{\lambda_i} - \frac{\mathbf{e}_j \cdot \mu_i^T}{\lambda_j} \\ &= N_{1,i,j} + Q_{1,i,j}. \end{aligned} \quad (5.42)$$

In equation (5.42), $N_{1,i,j}$ represents the original solution without errors, while $Q_{1,i,j}$ repre-

sents the impact of the errors. So the i^{th} row of $\Omega \cdot S_1$ can be written as:

$$\begin{aligned} & [C_{1,i,0}, \cdots, C_{1,i,i-1}, C_{1,i,i+1}, \cdots, C_{1,i,\alpha_l}] \cdot \Pi_{1,i}^{-1} \\ = \; & [N_{1,i,0}, \cdots, N_{1,i,i-1}, N_{1,i,i+1}, \cdots, N_{1,i,\alpha_l}] \cdot \Pi_{1,i}^{-1} \\ & + [Q_{1,i,0}, \cdots, Q_{1,i,i-1}, Q_{1,i,i+1}, \cdots, Q_{1,i,\alpha_l}] \cdot \Pi_{1,i}^{-1} \\ = \; & \xi_i + \delta_{1,i}, \end{aligned} \tag{5.43}$$

where $\Pi_{1,i} = [\mu_0^T, \cdots, \mu_{i-1}^T, \mu_{i+1}^T, \cdots, \mu_{\alpha_l}^T]$. ξ_i corresponds to the part independent of the errors. $\delta_{1,i}$ is the error part and can be further expanded as:

$$\begin{aligned} \delta_{1,i} \; = \; & \left[\frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{i-1}^T}{\lambda_i}, \frac{\mathbf{e}_i \cdot \mu_{i+1}^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{\alpha_l}^T}{\lambda_i} \right] \cdot \Pi_{1,i}^{-1} \\ & - \left[\frac{\mathbf{e}_0 \cdot \mu_i^T}{\lambda_0}, \cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}} \right] \cdot \Pi_{1,i}^{-1}. \end{aligned} \tag{5.44}$$

The first part of equation (5.44) can be reduced as follows:

$$\begin{aligned} & \left[\frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{i-1}^T}{\lambda_i}, \frac{\mathbf{e}_i \cdot \mu_{i+1}^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{\alpha_l}^T}{\lambda_i} \right] \cdot \Pi_{1,i}^{-1} \\ = \; & \frac{\mathbf{e}_i}{\lambda_i} \cdot \left[\mu_0^T, \cdots, \mu_{i-1}^T, \mu_{i+1}^T, \cdots, \mu_{\alpha_l}^T \right] \cdot \Pi_{1,i}^{-1} \\ = \; & \frac{\mathbf{e}_i}{\lambda_i}. \end{aligned} \tag{5.45}$$

So we have:

$$\begin{aligned} \delta_{1,i} \; = \; & \frac{\mathbf{e}_i}{\lambda_i} - \left[\frac{\mathbf{e}_0 \cdot \mu_i^T}{\lambda_0}, \cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}} \right] \cdot \Pi_{1,i}^{-1} \\ = \; & \frac{\mathbf{e}_i}{\lambda_i} - \rho_{1,i}. \end{aligned} \tag{5.46}$$

For $R_2{}' = R_2 + W_2$ in Algorithm 5.6 where $W_2 = \begin{bmatrix} \mathbf{e}_0 \\ \vdots \\ \mathbf{e}_{\alpha_l - 1} \\ \mathbf{e}_{\alpha_l + 1} \end{bmatrix}$, we can derive $C_{2,i,j}$, then

$\Omega \cdot S_2$ the same way. The i^{th} row of $\Omega \cdot S_2$ can be written as:

$$\xi_i + \delta_{2,i} = \xi_i + \frac{\mathbf{e}_i}{\lambda_i} - \rho_{2,i}, \tag{5.47}$$

where $\rho_{2,i} = \left[\frac{\mathbf{e}_0 \cdot \mu_i^T}{\lambda_0}, \cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l-1} \cdot \mu_i^T}{\lambda_{\alpha_l-1}}, \frac{\mathbf{e}_{\alpha_l+1} \cdot \mu_i^T}{\lambda_{\alpha_l+1}}\right] \cdot \Pi_{2,i}^{-1}$, $\Pi_{2,i} = [\mu_0^T, \cdots, \mu_{i-1}^T, \mu_{i+1}^T, \cdots, \mu_{\alpha_l-1}^T, \mu_{\alpha_l+1}^T]$.

Because $\Pi_{1,i}$ is a full rank matrix, $\rho_{1,i} = \rho_{2,i}$ is equivalent to $\rho_{1,i} \cdot \Pi_{1,i} = \rho_{2,i} \cdot \Pi_{1,i}$.

Similar to the proof of Lemma 1, suppose $\Pi_{2,i}^{-1} = \begin{bmatrix} \eta_0 \\ \vdots \\ \eta_{\alpha_l-1} \\ \eta_{\alpha_l+1} \end{bmatrix}$, we have $\eta_s \cdot \mu_r^T = \begin{cases} 1 & r = s \\ 0 & r \neq s \end{cases}$. So

$$\rho_{1,i} \cdot \Pi_{1,i} = \left[\cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l-1} \cdot \mu_i^T}{\lambda_{\alpha_l-1}}, \frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}}\right], \tag{5.48}$$

$$\rho_{2,i} \cdot \Pi_{1,i} = \left[\cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l-1} \cdot \mu_i^T}{\lambda_{\alpha_l-1}}, x_{2,\alpha_l}\right]. \tag{5.49}$$

Because $\mu_0^T, \cdots, \mu_{i-1}^T, \mu_{i+1}^T, \cdots, \mu_{\alpha_l-1}^T, \mu_{\alpha_l+1}^T$ are linearly independent, they can be viewed as a set of bases of the α_l dimensional linear space. So we have

$$\mu_{\alpha_l}^T = \sum_{r=0, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \mu_r^T. \tag{5.50}$$

Thus

$$\begin{aligned} x_{2,\alpha_l} &= \left[\cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l-1} \cdot \mu_i^T}{\lambda_{\alpha_l-1}}, \frac{\mathbf{e}_{\alpha_l+1} \cdot \mu_i^T}{\lambda_{\alpha_l+1}}\right] \cdot \Pi_{2,i}^{-1} \cdot \left(\sum_{r=0, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \mu_r^T\right) \\ &= \left(\sum_{r=0, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \frac{\mathbf{e}_r \cdot \mu_i^T}{\lambda_r}\right). \end{aligned} \tag{5.51}$$

If

$$\frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}} = \sum_{r=0, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \frac{\mathbf{e}_r \cdot \mu_i^T}{\lambda_r} \ (0 \leq i \leq \alpha_l - 1), \tag{5.52}$$

$\rho_{1,i}$ and $\rho_{2,i}$ will be equal, so are $\Omega \cdot S_1$ and $\Omega \cdot S_2$. Therefore, Algorithm 5.6 will fail.

For the error $\mathbf{e}_i$ $(0 \leq i \leq \alpha_l + 1)$, the following equation holds:

$$\mathbf{e}_i \cdot [\mu_0^T, \mu_1^T, \cdots, \mu_{\alpha_l - 1}^T] = [\hat{e}_{i,0}, \hat{e}_{i,1}, \cdots, \hat{e}_{i,\alpha_l - 1}] = \hat{\mathbf{e}}_i. \tag{5.53}$$

Because $[\mu_0^T, \mu_1^T, \cdots, \mu_{\alpha_l - 1}^T]$ is a full rank matrix, there is a one-to-one mapping between $\mathbf{e}_i$ and $\hat{\mathbf{e}}_i$. Equation (5.52) can be written as:

$$\frac{\hat{e}_{\alpha_l,i}}{\lambda_{\alpha_l}} = \sum_{r=0, r \neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \frac{\hat{e}_{r,i}}{\lambda_r} \ (0 \leq i \leq \alpha_l - 1). \tag{5.54}$$

When the number of malicious nodes in the $k_l + 1$ nodes is less than $k_l + 1$, the malicious nodes can collude to satisfy equation (5.54) for at most one particular i. So the probability that equation (5.54) holds is $1/q^2$ for at least $\alpha_l - 1$ out of α_l $i's$ between 0 and $\alpha_l - 1$. If we consider equation (5.54) for all the $i's$ simultaneously, the probability will be at most $(1/q^2)^{\alpha_l - 1}$. As we have mentioned above, the probability for $T_1 = T_2$ will be at most $(1/q^2)^{\alpha_l - 1}$. In this case, the detection probability is at least $1 - (1/q^2)^{2(\alpha_l - 1)}$.

(b) If one of the λ_i $(0 \leq i \leq \alpha_l)$ equals to 0, we can assume $\lambda_0 = 0$ without loss of generality. When $i = 0$, the solution for equation (5.38) is:

$$C_{1,0,j} = \hat{R}_{1,0,j} + \mathbf{e}_0 \cdot \mu_j^T = N_{1,0,j} + Q_{1,0,j}. \tag{5.55}$$

Similar to equations (5.43), (5.44) and (5.45), we have $\delta_{1,0} = \mathbf{e}_0$. For $R_2{}' = R_2 + W_2$, it is easy to see that $\delta_{2,0} = \mathbf{e}_0$. So the first rows of $\Omega \cdot S_1$ and $\Omega \cdot S_2$ are the same no matter what the error vector $\mathbf{e}_0$ is.

When $i > 0, j = 0$, the solution for equation (5.38) is:

$$C_{1,i,0} = \hat{R}_{1,i,0} + \mathbf{0} \cdot \mu_0^T + \mathbf{e}_0 \cdot \mu_i^T = N_{1,i,0} + Q_{1,i,0}, \tag{5.56}$$

where $\mathbf{0}$ is a zero row vector. When $i > 0, j > 0$, the solution has the same expression as equation (5.42). In this case, for the i^{th} $(i > 0)$ row of $\Omega \cdot S_1$, equation (5.44) can be written

as:

$$\begin{aligned}\delta_{1,i} &= \left[\mathbf{0}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{i-1}^T}{\lambda_i}, \frac{\mathbf{e}_i \cdot \mu_{i+1}^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{\alpha_l}^T}{\lambda_i}\right] \cdot \Pi_{1,i}^{-1} \\ &\quad - \left[-\mathbf{e}_0 \cdot \mu_i^T, \cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}}\right] \cdot \Pi_{1,i}^{-1}. \qquad (5.57)\end{aligned}$$

The first part of equation (5.57) can be divided into two parts:

$$\begin{aligned}&\left[\frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{i-1}^T}{\lambda_i}, \frac{\mathbf{e}_i \cdot \mu_{i+1}^T}{\lambda_i}, \cdots, \frac{\mathbf{e}_i \cdot \mu_{\alpha_l}^T}{\lambda_i}\right] \cdot \Pi_{1,i}^{-1} - \left[\frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i}, \mathbf{0}, \cdots, \mathbf{0}\right] \cdot \Pi_{1,i}^{-1} \\ = \; &\frac{\mathbf{e}_i}{\lambda_i} - \frac{\mathbf{e}_i}{\lambda_i} \cdot [\mu_0^T, \mathbf{0}, \cdots, \mathbf{0}] \cdot \Pi_{1,i}^{-1}. \qquad (5.58)\end{aligned}$$

So equation (5.57) can be further written as:

$$\begin{aligned}\delta_{1,i} &= \frac{\mathbf{e}_i}{\lambda_i} - \left[\frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i} - \mathbf{e}_0 \cdot \mu_i^T, \cdots, \frac{\mathbf{e}_{i-1} \cdot \mu_i^T}{\lambda_{i-1}}, \frac{\mathbf{e}_{i+1} \cdot \mu_i^T}{\lambda_{i+1}}, \cdots, \frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}}\right] \cdot \Pi_{1,i}^{-1} \\ &= \frac{\mathbf{e}_i}{\lambda_i} - \rho_{1,i}. \qquad (5.59)\end{aligned}$$

By employing the same derivation in case (a), for $1 \leq i \leq \alpha_l - 1$, $\rho_{1,i}$ and $\rho_{2,i}$ will be equal if

$$\frac{\mathbf{e}_{\alpha_l} \cdot \mu_i^T}{\lambda_{\alpha_l}} = \sum_{r=1, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \frac{\mathbf{e}_r \cdot \mu_i^T}{\lambda_r} - \zeta_0 \cdot \mathbf{e}_0 \cdot \mu_i^T + \zeta_0 \cdot \frac{\mathbf{e}_i \cdot \mu_0^T}{\lambda_i}, \qquad (5.60)$$

$$\frac{\hat{e}_{\alpha_l,i}}{\lambda_{\alpha_l}} = \sum_{r=1, r\neq i, \alpha_l}^{r=\alpha_l+1} \zeta_r \cdot \frac{\hat{e}_{r,i}}{\lambda_r} - \zeta_0 \cdot \hat{e}_{0,i} + \zeta_0 \cdot \frac{\hat{e}_{i,0}}{\lambda_i}. \qquad (5.61)$$

When the number of malicious nodes in the k_l+1 nodes is less than k_l+1, for the same reason as in case (a), the probability that equation (5.61) holds is $1/q^2$ for at least $\alpha_l - 2$ out of $\alpha_l - 1$ $i's$ between 1 and $\alpha_l - 1$. If we consider equation (5.61) for all the $i's$ simultaneously, the probability will be at most $(1/q^2)^{\alpha_l - 2}$. Here the probability for $T_1 = T_2$ will be $(1/q^2)^{\alpha_l - 2}$. In this case, the detection probability is $1 - (1/q^2)^{2(\alpha_l - 2)}$.

Combining both cases, the detection probability is at least $1 - (1/q^2)^{2(\alpha_l - 2)}$. □

5.3.5.2 Recovery Mode

Once DC detects errors using Algorithm 5.6, it will send integer $j = q - 1$ to all the q^2 nodes in the network requesting symbols. Storage nodes will still use the way similar to that of the error-free network in Section 5.3.4 to send symbols. The reconstruct procedures are described in Algorithm 5.7.

Algorithm 5.7 [Recovery Mode] DC Reconstructs the Original File in Hostile Network

---

Step 1: For every $0 \leq l \leq q-1$, we divide the symbol vector $\tilde{\mathbf{y}}'_{i,l}$ into A/α_l equal row vectors: $[\tilde{\mathbf{y}}'_{i,l,1}, \tilde{\mathbf{y}}'_{i,l,2}, \cdots, \tilde{\mathbf{y}}'_{i,l,A/\alpha_l}]$. (Without loss of generality, we assume $0 \leq i \leq q^2 - 1$.)

Step 2: For every $q - 1 \geq l \geq 0$ in descending order and $1 \leq t \leq A/\alpha_l$ in ascending order, DC can reconstruct the matrices related to the original file when the errors in the received symbol vectors $\tilde{\mathbf{y}}'_{i,l,t}$ from q^2 storage nodes can be corrected:

Step 2.1: Let $R' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{q^2-1,l,t}]^T$.

Step 2.2: If the number of corrupted nodes detected is larger than $\min\{q^2 - k_l, \lfloor (q^2 - k_{q-1})/2 \rfloor\}$, then the number of errors have exceeded the error correction capability. We will flag the decoding failure and exit the algorithm.

Step 2.3: Since the number of errors is within the error correction capability of the H-MSR code, substitute $\tilde{\mathbf{y}}'_{i,l,t}$ in R' with the symbol $\otimes$ representing an erasure vector if node i has been detected to be corrupted in the previous loops (previous values of l, t).

Step 2.4: Solve $S_{l,t}, T_{l,t}$ using the method described in section 5.3.6. If symbols from node i are detected to be erroneous during the calculation, mark node i as corrupted.

Step 3: DC reconstructs the original file from all the matrices $S_{l,t}, T_{l,t}$, $0 \leq l \leq q - 1$ and $1 \leq t \leq A/\alpha_l$.

5.3.6 Recover Matrices $S_{l,t}, T_{l,t}$ from $q\hat{}2$ Storage Nodes

When there are bogus symbols $\tilde{p}'_{i,l,t}$ sent by the corrupted nodes for certain l, t, we can recover the matrices $S_{l,t}, T_{l,t}$ as follows:

For R' in Algorithm 5.7, we have $\Psi_{DC} \cdot \begin{bmatrix} S' \\ \\ T' \end{bmatrix} = R'$, and

$$\Phi_{DC} S' \Phi_{DC}^T + \Delta_{DC} \Phi_{DC} T' \Phi_{DC}^T = R' \Phi_{DC}^T, \tag{5.62}$$

where $\Psi_{DC} = [\Phi_{DC}, \Delta_{DC} \cdot \Phi_{DC}]$, $\Phi_{DC} = \begin{bmatrix} \mu_0 \\ \\ \mu_1 \\ \\ \vdots \\ \\ \mu_{q^2-1} \end{bmatrix}$ and μ_i represents $\mu_{i,l}$ which is the i^{th} row of the encoding matrix Φ_l in the proof of Theorem 5.1.

Let $C = \Phi_{DC} S' \Phi_{DC}^T$, $D = \Phi_{DC} T' \Phi_{DC}^T$, and $\hat{R}' = R' \Phi_{DC}^T$, then

$$C + \Delta_{DC} D = \hat{R}'. \tag{5.63}$$

Since C, D are both symmetric, we can solve the non-diagonal elements of them as follows:

$$\begin{cases} C_{i,j} + \lambda_i \cdot D_{i,j} = \hat{R}'_{i,j} \\ \\ C_{i,j} + \lambda_j \cdot D_{i,j} = \hat{R}'_{j,i} \end{cases}. \tag{5.64}$$

Because matrices C and D have the same structure, here we only focus on C (corresponding to S'). It is straightforward to see that if node i is malicious and there are errors in the i^{th} row of R', there will be errors in the i^{th} row of $\hat{R}'$. Furthermore, there will be errors in the i^{th} row and i^{th} column of C. Define $S' \Phi_{DC}^T = \hat{S}'$, we have

$$\Phi_{DC} \hat{S}' = C. \tag{5.65}$$

Here we can view each column of C as a $(q^2 - 1, \alpha_l, q^2 - \alpha_l)$ MDS code because Φ_{DC} is a Vandermonde matrix. The length of the code is $q^2 - 1$ since the diagonal elements of C is

unknown. Suppose node j is uncorrupted. If the number of erasures σ (corresponding to the previously detected corrupted nodes) and the number of the corrupted nodes τ that have not been detected satisfy:

$$\sigma + 2\tau + 1 \leq q^2 - \alpha_l, \tag{5.66}$$

then the j^{th} column of C can be recovered and the error locations (corresponding to the corrupted nodes) can be pinpointed. The non-diagonal elements of C can be recovered. So DC can reconstruct $S_{l,t}$ using the method similar to [57]. For $T_{l,t}$, the recovering process is similar.

5.4 Hermitian Code Based MBR Regenerating Code (H-MBR Code)

5.4.1 Encoding H-MBR Code

In this section, we will analyze the H-MBR code based on the MBR point with $\beta = 1$. According to equation (2.8), we have $d = \alpha$.

Let $\alpha_0, \cdots, \alpha_{q-1}$ be a strictly decreasing integer sequence satisfying $0 < \alpha_i \leq \kappa(i), 0 \leq i \leq q-1$. The least common multiple of $\alpha_0, \cdots, \alpha_{q-1}$ is A. Let $k_0, \cdots, k_{q-1}$ be a integer sequence satisfying $0 < k_i \leq \alpha_i, 0 \leq i \leq q-1$. Suppose the data contains $B = A \cdot \sum_{i=0}^{q-1} (k_i(2\alpha_i - k_i + 1)/(2\alpha_i))$ message symbols from the finite field $GF(q^2)$. In practice, if the size of the actual data is larger than B symbols, we can fragment it into blocks of size B and process each block individually.

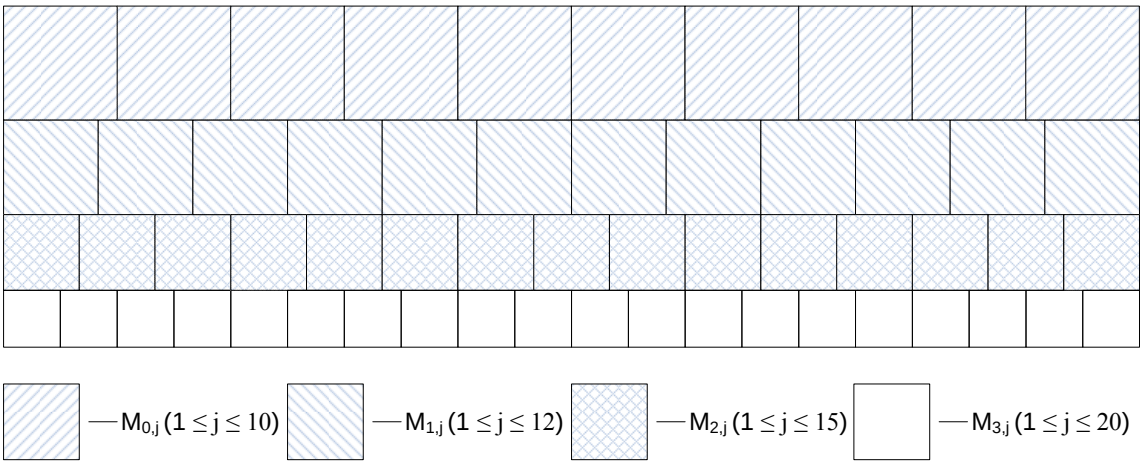

Figure 5.3 An example illustration of matrix M

We arrange the B symbols into matrix M as below:

$$M = \begin{bmatrix} M_0 \\ M_1 \\ \vdots \\ M_{q-1} \end{bmatrix}, \tag{5.67}$$

where

$$M_i = [M_{i,1}, M_{i,2}, \cdots, M_{i,A/\alpha_i}] \tag{5.68}$$

and

$$M_{i,j} = \begin{bmatrix} S_{i,j} & T_{i,j} \\ T_{i,j}^T & \mathbf{0}. \end{bmatrix} \tag{5.69}$$

$S_{i,j}, 0 \leq i \leq q-1, 1 \leq j \leq A/\alpha_i$ is a symmetric matrix of size $k_i \times k_i$ with the upper-triangular entries filled by data symbols. $T_{i,j}$ is a $k_i \times (\alpha_i - k_i)$ matrix. Thus $M_{i,j}$ contains $k_i(2\alpha_i - k_i + 1)/2$ symbols, M_i contains $A \cdot k_i(2\alpha_i - k_i + 1)/(2\alpha_i)$ symbols and M contains B symbols. Figure 5.3 shows an example of matrix M for $q = 4, \alpha_0 = 6, \alpha_1 = 5, \alpha_2 = 4, \alpha_3 = 3$.

In figure 5.3, the submatrix $M_{i,j}$ is represented by the square in the corresponding position with the size representing the size of the submatrix.

For distributed storage, we encode M using Algorithm 5.8:

Algorithm 5.8 Encoding H-MBR Code

Step 1: First we encode the data matrices M defined above using a Hermitian code $\mathcal{H}_m$ over $GF(q^2)$ with parameters $\kappa(j)$ $(0 \leq j \leq q-1)$ and m $(m \geq q^2-1)$. The $q^3 \times A$ codeword matrix can be written as $Y = \mathcal{H}_m(M)$.

Step 2: Then we divide the codeword matrix Y into q^2 submatrices $Y_0, \cdots, Y_{q^2-1}$ of the size $q \times A$ and store one submatrix in each of the q^2 storage nodes as shown in Figure. 5.2.

Then we have the following theorem:

Theorem 5.4. *By processing the data symbols using Algorithm 5.8, we can achieve the MBR point in distributed storage.*

Proof. Similar to the proof of Theorem 5.1, we can get the following equation considering all the columns of $\mathcal{H}_m(M)$:

$$\Phi_i \cdot M_{i,j} = G_{i,j}, \tag{5.70}$$

where $G_{i,j} = [\mathcal{G}_i^{(1)}, \cdots, \mathcal{G}_i^{(\alpha_i)}]$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$. $\mathcal{G}_i^{(l)}$ corresponds to the l^{th} column of the submatrix $M_{i,j}$ and each element of $\mathcal{G}_i = [g_i(0), g_i(1), \cdots, g_i(\phi^{q^2-2})]^T$ can be derived from a distinct storage node. Φ_i is defined in equation (6.2).

Next we will study the optimality of the code in the sense of the MBR point. For $\Phi_i \cdot M_{i,j}$, $0 \leq i \leq q-1, 1 \leq j \leq A/\alpha_i$, $M_{i,j}$ is symmetric and satisfies the requirements for MBR point according to [57] with parameters $d = \alpha_i, k = k_i, \alpha = \alpha_i, \beta = 1, B = k_i(2\alpha_i - k_i + 1)/2$. By encoding M using $\mathcal{H}_m(M)$ and distributing $Y_0, \cdots, Y_{q^2-1}$ into q^2 storage nodes, each row of the matrix $\Phi_i \cdot M_{i,j}$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, can be derived in a corresponding storage node. Because $\Phi_i \cdot M_{i,j}$ achieves the MBR point, data related

to matrices $M_{i,j}$, $0 \leq i \leq q-1$, $1 \leq j \leq A/\alpha_i$, can be regenerated at the MBR point. Therefore, Algorithm 5.8 can achieve the MBR point. □

5.4.2 Regeneration of the H-MBR Code in the Error-free Network

In this section, we will discuss the regeneration for the H-MBR code in the error-free network. Let $\mathbf{w}_i = [g_0(\phi(^{s_i})), g_1(\phi(^{s_i})), \cdots, g_{q-1}(\phi(^{s_i}))]^T$, then $\mathbf{w}_i = B_i^{-1} \cdot \mathbf{y}_i = [g_0(\phi^{s_i}), \cdots, g_{q-1}(\phi^{s_i})]^T$, for every column $\mathbf{y}_i$ of Y_i.

The main idea of the regeneration algorithms is similar to that of the H-MSR code: regenerate $g_l(\phi(^{s_i})), 0 \leq l \leq q-1$, by downloading help symbols from $d_l = \alpha_l$ nodes, where d_l is the regeneration parameter d for $g_l(\phi(^{s_i}))$ in the H-MBR code regeneration.

Suppose node z fails, we use Algorithm 5.9 to regenerate the exact H-MBR code symbols of node z. For convenience, we suppose $d_q = \alpha_q = 0$ and define

$$\mathbf{W}_{i,j,l} = \begin{bmatrix} \mu_{i,l} \\ \mu_{i+1,l} \\ \vdots \\ \mu_{j,l} \end{bmatrix}, \tag{5.71}$$

where $\mu_{t,l}$, $i \leq t \leq j$, is the t^{th} row of Φ_l.

Similar to the H-MSR code, replacement node z' will send requests to helper nodes in the way same to that in Section 5.3.2. Upon receiving the request integer j, helper node i will calculate and send the help symbols similar to that of Section 5.3.2.

When the replacement node z' receives all the requested symbols, it can regenerate the symbols stored in the failed node z using the following algorithm:

For Algorithm 5.9 we can derive the equivalent storage parameters for each symbol block of size $B_j = Ak_j(2\alpha_j - k_j + 1)/(2\alpha_j) : d = \alpha_j, k = k_j, \alpha = A, \beta = A/\alpha_j$, $0 \leq j \leq q-1$ and

Algorithm 5.9 z' Regenerates Symbols of the Failed Node z

Step 1: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, we can calculate the regenerated symbols which are related to the help symbols $\tilde{p}_{i,l,t}$ from d_l helper nodes: (Without loss of generality, we assume $0 \leq i \leq d_l - 1$.)

Step 1.1: Let $\mathbf{p} = [\tilde{p}_{0,l,t}, \tilde{p}_{1,l,t}, \cdots, \tilde{p}_{d_l-1,l,t}]^T$, solve the equation: $\mathbf{W}_{0,d_l-1,l} \cdot \mathbf{x} = \mathbf{p}$.

Step 1.2: Since $\mathbf{x} = M_{l,t} \cdot \mu_{z,l}^T$ and $M_{l,t}$ is symmetric, we can calculate $\tilde{\mathbf{y}}_{z,l,t} = \mathbf{x}^T = \mu_{z,l} \cdot M_{l,t}$.

Step 2: Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z \cdot \widetilde{Y}_z$.

equation (2.8) of the MBR point holds for these parameters. Theorem 5.4 guarantees that Algorithm 5.9 can achieve the MBR point for data regeneration of the H-MBR code.

5.4.3 Regeneration of the H-MBR Code in the Hostile Network

In hostile network, Algorithm 5.9 may be unable to regenerate the failed node due to the possible bogus symbols received from the responses. In fact, even if the replacement node z' can derive the symbol matrix $Y_{z'}$ using Algorithm 5.9, it cannot verify the correctness of the result.

Similar to the H-MSR code, there are two modes for the helper nodes to regenerate the H-MBR code of a failed storage node in hostile network. One mode is the detection mode, in which no error has been found in the symbols received from the helper nodes. Once errors are detected, the recovery mode will be used to correct the errors and locate the malicious nodes.

5.4.3.1 Detection Mode

In the detection mode, the replacement node z' will send requests in the way similar to that of the error-free network in Section 5.4.2. The only difference is that when $j = q-1$, z' sends requests to $d_{q-1} - d_q + 1$ nodes instead of $d_{q-1} - d_q$ nodes. Helper nodes will still use the way similar to that of the error-free network in Section 5.4.2 to send the help symbols. The regeneration algorithm is described in Algorithm 5.10 with the detection probability characterized in Theorem 5.5.

Algorithm 5.10 [Detection Mode] z' Regenerates Symbols of the Failed Node z in Hostile Network

---

Step 1: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, we can calculate the regenerated symbols which are related to the help symbols $\tilde{p}'_{i,l,t}$ from d_l helper nodes. $\tilde{p}'_{i,l,t} = \tilde{p}_{i,l,t} + e_{i,l,t}$ is the response from the i^{th} helper node. If $\tilde{p}_{i,l,t}$ has been modified by the malicious node i, we have $e_{i,l,t} \in GF(q^2)\backslash\{0\}$. To detect whether there are errors, we will calculate symbols from two sets of helper nodes then compare the results. (Without loss of generality, we assume $0 \leq i \leq d_l$.)

Step 1.1: Let $\mathbf{p}_1{}' = [\tilde{p}'_{0,l,t}, \tilde{p}'_{1,l,t}, \cdots, \tilde{p}'_{d_l-1,l,t}]^T$, where the symbols are collected from node 0 to node $d_l - 1$, solve the equation $\mathbf{W}_{0,d_l-1,l} \cdot \mathbf{x}_1 = \mathbf{p}_1{}'$.

Step 1.2: Let $\mathbf{p}_2{}' = [\tilde{p}'_{1,l,t}, \tilde{p}'_{2,l,t}, \cdots, \tilde{p}'_{d_l,l,t}]^T$, where the symbols are collected from node 1 to node d_l, solve the equation $\mathbf{W}_{1,d_l,l} \cdot \mathbf{x}_2 = \mathbf{p}_2{}'$.

Step 1.3: If $\mathbf{x}_1 = \mathbf{x}_2$, compute $\tilde{\mathbf{y}}_{z,l,t} = \mu_{z,l} \cdot M_{l,t}$ as described in Algorithm 5.9. Otherwise, errors are detected in the help symbols. Exit the algorithm and switch to recovery regeneration mode.

Step 2: No error has been detected for the calculating of the regeneration so far. Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z \cdot \widetilde{Y}_z$.

Theorem 5.5. *When the number of malicious nodes in the $d_l + 1$ helper nodes of Algorithm 5.10 is less than $d_l + 1$, the probability for the bogus symbols sent from the malicious nodes to be detected is at least $1 - 1/q^2$.*

Proof. Similar to the proof of Theorem 5.2, we can write

$$\mathbf{x}_1 = \mathbf{x} + \mathbf{W}_{0,d_l-1,l}^{-1} \cdot [e_0, \cdots, e_{d_l-1}]^T = \mathbf{x} + \hat{\mathbf{x}}_1, \tag{5.72}$$

$$\mathbf{x}_2 = \mathbf{x} + \mathbf{W}_{1,d_l,l}^{-1} \cdot [e_1, \cdots, e_{d_l}]^T = \mathbf{x} + \hat{\mathbf{x}}_2. \tag{5.73}$$

Since $\mathbf{W}_{0,d_l-1,l}, \mathbf{W}_{1,d_l,l}$ are full rank matrices like the matrices $\mathbf{V}_{0,d_l-1,l}, \mathbf{V}_{1,d_l,l}$ in the proof of Lemma 1 and any d_l vectors out of $\mu_{0,l}, \mu_{1,l}, \cdots, \mu_{d_l,l}$ are linearly independent, the rest of this proof is similar to that of Lemma 1. When the number of malicious nodes in the d_l+1 helper nodes is less than d_l+1, the probability for $\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_2$ is at most $1/q^2$. Therefore, the detection probability is at least $1 - 1/q^2$. □

5.4.3.2 Recovery Mode

Once the replacement node z' detects errors using Algorithm 5.10, it will send integer $j = q-1$ to all the other $q^2 - 1$ nodes in the network requesting help symbols. Helper nodes will still use the way similar to that of the error-free network in Section 5.4.2 to send the help symbols. z' can regenerate symbols using Algorithm 5.11.

5.4.4 Reconstruction of the H-MBR code in the Error-free Network

In this section, we will discuss the reconstruction of the H-MBR code in the error-free network. The main idea of the reconstruction algorithms is similar to that of the H-MSR code: reconstruct $g_l(\phi(^{s}i)), 0 \leq l \leq q-1$, by downloading help symbols from k_l nodes, where k_l represents the reconstruction parameter k for $g_l(\phi(^{s}i))$ in the H-MBR code. We use Algorithm 5.12 in the network for the data collector DC to reconstruct the original file. For convenience, we suppose $k_q = 0$.

Similar to the H-MSR code described in Section 5.3.4, DC will send requests to storage nodes. Upon receiving the request integer j, node i will calculate and send symbols. When DC receives all the requested symbols, it can reconstruct the original file using the following algorithm:

Algorithm 5.11 [Recovery Mode] z' Regenerates Symbols of the Failed Node z in Hostile Network

Step 1: For every $q-1 \geq l \geq 0$ in descending order and $1 \leq t \leq A/\alpha_l$ in ascending order, we can regenerate the symbols when the errors in the received help symbols $\tilde{p}'_{i,l,t}$ from q^2-1 helper nodes can be corrected. Without loss of generality, we assume $0 \leq i \leq q^2-2$.

Step 1.1: Let $\mathbf{p'} = [\tilde{p}'_{0,l,t}, \tilde{p}'_{1,l,t}, \cdots, \tilde{p}'_{q^2-2,l,t}]^T$. Since $\mathbf{W}_{0,q^2-2,l} \cdot \mathbf{x} = \mathbf{p'}$, $\mathbf{p'}$ can be viewed as an MDS code with parameters (q^2-1, d_l, q^2-d_l).

Step 1.2: Substitute $\tilde{p}'_{i,l,t}$ in $\mathbf{p'}$ with the symbol $\otimes$ representing an erasure if node i has been detected to be corrupted in the previous loops (previous values of l, t).

Step 1.3: If the number of erasures in $\mathbf{p'}$ is larger than $\min\{q^2-d_l-1, \lfloor (q^2-d_{q-1}-1)/2 \rfloor\}$, then the number of errors have exceeded the error correction capability. We will flag the decoding failure and exit the algorithm.

Step 1.4: Since the number of errors is within the error correction capability of the MDS code, decode $\mathbf{p'}$ to $\mathbf{p}'_{cw}$ and solve $\mathbf{x}$.

Step 1.5: If the i^{th} position symbols of $\mathbf{p}'_{cw}$ and $\mathbf{p'}$ are different, mark node i as corrupted.

Step 1.6: Compute $\tilde{\mathbf{y}}_{z,l,t} = \mu_{z,l} \cdot M_{l,t}$ as described in Algorithm 5.9.

Step 2: Let $\widetilde{Y}_z$ be a $q \times A$ matrix with the l^{th} row defined as $[\tilde{\mathbf{y}}_{z,l,1}, \cdots, \tilde{\mathbf{y}}_{z,l,A/\alpha_l}], 0 \leq l \leq q-1$.

Step 3: Calculate the regenerated symbols of the failed node z: $Y_{z'} = Y_z = B_z \cdot \widetilde{Y}_z$.

Algorithm 5.12 DC Reconstructs the Original File

Step 1: For every $0 \leq l \leq q-1$, divide the symbol vector $\tilde{\mathbf{y}}_{i,l}$ into A/α_l equal row vectors: $[\tilde{\mathbf{y}}_{i,l,1}, \tilde{\mathbf{y}}_{i,l,2}, \cdots, \tilde{\mathbf{y}}_{i,l,A/\alpha_l}]$. ($\tilde{\mathbf{y}}_{i,l}$ is the response from the i^{th} node and we assume $0 \leq i \leq k_l - 1$ without loss of generality.)

Step 2: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, DC reconstructs the matrices related to the original file:

Step 2.1: Let $R = [\tilde{\mathbf{y}}^T_{0,l,t}, \tilde{\mathbf{y}}^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}^T_{k_l-1,l,t}]^T$, we have the equation: $\mathbf{W}_{0,k_l-1,l} \cdot M_{l,t} = R$ according to the encoding algorithm.

Step 2.2: DC reconstructs $M_{l,t}$ using techniques similar to that of [57].

Step 3: DC reconstructs the original file from all the matrices $M_{l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$.

5.4.5 Reconstruction of the H-MBR code in the Hostile Network

Similar to the H-MSR code, the reconstruction algorithms for H-MBR code in error-free network do not work in hostile network. Even if the data collecter can calculate the symbol matrices M using Algorithm 5.12, it cannot verify whether the result is correct or not. There are two modes for the original file to be reconstructed in hostile network. One mode is the detection mode, in which no error has been found in the symbols received from the storage nodes. Once errors are detected in the detection mode, the recovery mode will be used to correct the errors and locate the malicious nodes.

5.4.5.1 Detection Mode

In the detection mode, DC will send requests in the way similar to that of the error-free network in Section 5.4.4. The only difference is that when $j = q - 1$, DC will send requests to $k_{q-1} - k_q + 1$ nodes instead of $k_{q-1} - k_q$ nodes. Storage nodes will send symbols similar to that of the error-free network in Section 5.4.4. The reconstruction algorithm is described in Algorithm 5.13 with the detection probability described in Theorem 5.6.

Theorem 5.6. *When the number of malicious nodes in the $k_l + 1$ nodes of Algorithm 5.13 is less than $k_l + 1$, the probability for the bogus symbols sent from the malicious nodes to be detected is at least $1 - 1/q^{2\alpha_l}$.*

Proof. For convenience, we write $\mathbf{e}_{i,l,t}$ as $\mathbf{e}_i$ in the proof. $\mathbf{e}_i \in [GF(q^2)]^{\alpha_l}$ for $0 \leq i \leq k_l$. In Algorithm 5.13, ${R_1}' = R_1 + Q_1$ where $Q_1 = \begin{bmatrix} \mathbf{e}_0 \\ \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_{k_l-1} \end{bmatrix}$. Let $\mathbf{W}_{0,k_l-1,l} = [\Omega_{DC1}, \Delta_{DC1}]$,

Algorithm 5.13 [Detection Mode] DC Reconstructs the Original File in Hostile Network

Step 1: For every $0 \leq l \leq q-1$, we can divide the symbol vector $\tilde{\mathbf{y}}'_{i,l}$ into A/α_l equal row vectors: $[\tilde{\mathbf{y}}'_{i,l,1}, \tilde{\mathbf{y}}'_{i,l,2}, \cdots, \tilde{\mathbf{y}}'_{i,l,A/\alpha_l}]$. $\tilde{\mathbf{y}}'_{i,l} = \tilde{\mathbf{y}}_{i,l} + \mathbf{e}_{i,l}$ is the response from the i^{th} storage node. If $\tilde{\mathbf{y}}_{i,l}$ has been modified by the malicious node i, we have $\mathbf{e}_{i,l} \in (GF(q^2))^A \backslash \{\mathbf{0}\}$. To detect whether there are errors, we will reconstruct the original file from two sets of storage nodes then compare the results. (Without loss of generality, we assume $0 \leq i \leq k_l$.)

Step 2: For every $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$, DC can reconstruct the matrices related to the original file:

Step 2.1: Let $R' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{k_l,l,t}]^T$.

Step 2.2: Let ${R_1}' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{k_l-1,l,t}]^T$, which are the symbols collected from node 0 to node $k_l - 1$, then we have $\mathbf{W}_{0,k_l-1,l} \cdot M_1 = {R_1}'$. Solve M_1 using the method same to algorithm 5.12.

Step 2.3: Let ${R_2}' = [\tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{k_l,l,t}]^T$, which are the symbols collected from node 1 to node k_l, then we have $\mathbf{W}_{1,k_l,l} \cdot M_2 = {R_2}'$. Solve M_2 using the method same to algorithm 5.12.

Step 2.4: Compare M_1 with M_2. If they are the same, let $M_{l,t} = M_1$. Otherwise, errors are detected in the received symbols. Exit the algorithm and switch to recovery reconstruction mode.

Step 3: No error has been detected for the calculating of the reconstruction so far. So DC can reconstruct the original file from all the matrices $M_{l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$.

$R_1 = [R_{1,1}, R_{1,2}]$ and $Q_1 = [Q_{1,1}, Q_{1,2}]$, where Ω_{DC1}, $R_{1,1}$, $Q_{1,1}$ are $k_l \times k_l$ submatrices and Δ_{DC1}, $R_{1,2}$, $Q_{1,2}$ are $k_l \times (\alpha_l - k_l)$ submatrices.

According to equation (5.69), we have

$$\begin{aligned} \mathbf{W}_{0,k_l-1,l} \cdot M_1 &= [\Omega_{DC1} S_1 + \Delta_{DC1} T_1^T, \Omega_{DC1} T_1] \\ &= [R_{1,1} + Q_{1,1}, R_{1,2} + Q_{1,2}]. \end{aligned} \tag{5.74}$$

Since Ω_{DC1} is a submatrix of a Vandermonde matrix, it is a full rank matrix. So we have

$$\begin{aligned} T_1 &= \Omega_{DC1}^{-1} R_{1,2} + \Omega_{DC1}^{-1} Q_{1,2} = T + \hat{T}_1, && (5.75) \\ S_1 &= \Omega_{DC1}^{-1}(R_{1,1} + Q_{1,1} - \Delta_{DC1} T_1^T) \\ &= \Omega_{DC1}^{-1}(R_{1,1} - \Delta_{DC1} T^T) + \Omega_{DC1}^{-1}(Q_{1,1} - \Delta_{DC1} \hat{T}_1^T) \\ &= S + \Omega_{DC1}^{-1}(Q_{1,1} - \Delta_{DC1} \hat{T}_1^T) = S + \hat{S}_1. && (5.76) \end{aligned}$$

For $R_2{}' = R_2 + Q_2$ in Algorithm 5.13, Let $R_2 = [R_{2,1}, R_{2,2}]$, $Q_2 = [Q_{2,1}, Q_{2,2}]$ and $\mathbf{W}_{1,k_l,l} = [\Omega_{DC2}, \Delta_{DC2}]$, where $R_{2,1}$, $Q_{2,1}$, Ω_{DC2} are $k_l \times k_l$ submatrices and $R_{2,2}$, $Q_{2,2}$, Δ_{DC2} are $k_l \times (\alpha_l - k_l)$ submatrices. Similarly, we have

$$\begin{aligned} T_2 &= \Omega_{DC2}^{-1} R_{2,2} + \Omega_{DC2}^{-1} Q_{2,2} = T + \hat{T}_2, && (5.77) \\ S_2 &= S + \Omega_{DC2}^{-1}(Q_{2,1} - \Delta_{DC2} \hat{T}_2^T) = S + \hat{S}_2. && (5.78) \end{aligned}$$

If $\hat{T}_1 = \hat{T}_2$ and $\hat{S}_1 = \hat{S}_2$, Algorithm 5.13 will fail to detect the bogus symbols. So we will focus on $\hat{T}_1, \hat{T}_2$ and $\hat{S}_1, \hat{S}_2$.

Suppose $\Pi_{1,j} = [e_0, \cdots, e_{k_l-1}]^T, \Pi_{2,j} = [e_1, \cdots, e_{k_l}]^T$ are the $j^{th}, 1 \leq j \leq \alpha_l - k_l$, columns of $Q_{1,2}$ and $Q_{2,2}$ respectively, where $e_i \in GF(q^2)$. Since Ω_{DC1} and Ω_{DC2} are Vandermonde matrices and have the same relationship as that between $\mathbf{V}_{0,d_l-1,l}$ and $\mathbf{V}_{1,d_l,l}$, similar as the proof of Lemma 1, we can prove that when the number of malicious nodes in the $k_l + 1$ nodes is less than $k_l + 1$, the probability of $\Omega_{DC1}^{-1}\Pi_{1,j} = \Omega_{DC2}^{-1}\Pi_{2,j}$ is at most $1/q^2$. Thus the probability for $\hat{T}_1 = \hat{T}_2$ is at most $1/q^{2(\alpha_l - k_l)}$. Through the same procedure, we can derive that the probability of $\hat{S}_1 = \hat{S}_2$ is at most $1/q^{2k_l}$. The probability for both $\hat{S}_1 = \hat{S}_2$ and $\hat{T}_1 = \hat{T}_2$ is at most $1/q^{2\alpha_l}$. So the detection probability is at least $1 - 1/q^{2\alpha_l}$. □

5.4.5.2 Recovery Mode

Once DC detects errors using Algorithm 5.13, it will send integer $j = q - 1$ to all the q^2 nodes in the network requesting symbols. Storage node i will use the way similar to that

of the error-free network in Section 5.4.4 to send symbols. The reconstruct procedures are described in Algorithm 5.14.

Algorithm 5.14 [Recovery Mode] DC Reconstructs the Original File in Hostile Network

Step 1: For every $0 \leq l \leq q-1$, divide the symbol vector $\tilde{\mathbf{y}}'_{i,l}$ into A/α_l equal row vectors: $[\tilde{\mathbf{y}}'_{i,l,1}, \tilde{\mathbf{y}}'_{i,l,2}, \cdots, \tilde{\mathbf{y}}'_{i,l,A/\alpha_l}]$. (Without loss of generality, we assume $0 \leq i \leq q^2-1$.)

Step 2: For every $q-1 \geq l \geq 0$ in descending order and $1 \leq t \leq A/\alpha_l$ in ascending order, DC reconstructs the matrices related to the original file when the errors in the received symbol vectors $\tilde{\mathbf{y}}'_{i,l,t}$ from q^2 storage nodes can be corrected:

Step 2.1: Let $R' = [\tilde{\mathbf{y}}'^T_{0,l,t}, \tilde{\mathbf{y}}'^T_{1,l,t}, \cdots, \tilde{\mathbf{y}}'^T_{q^2-1,l,t}]^T$.

Step 2.2: If the number of corrupted nodes detected is larger than $\min\{q^2 - k_l, \lfloor (q^2 - k_{q-1})/2 \rfloor\}$, then the number of errors have exceeded the error correction capability. So here we will flag the decoding failure and exit the algorithm.

Step 2.3: Since the number of errors is within the error correction capability of the H-MBR code, substitute $\tilde{\mathbf{y}}'_{i,l,t}$ in R' with the symbol $\otimes$ representing an erasure vector if node i has been detected to be corrupted in the previous loops (previous values of l, t).

Step 2.4: Solve $M_{l,t}$ using the method in section 5.4.6. If symbols from node i are detected to be erroneous during the calculation, mark node i as corrupted.

Step 3: DC reconstructs the original file from all the matrices $M_{l,t}$, $0 \leq l \leq q-1$ and $1 \leq t \leq A/\alpha_l$.

5.4.6 Recover Matrices $M_{\alpha_l,t}$ from $q\hat{}2$ Storage Nodes

When there are bogus symbols $\tilde{p}'_{i,l,t}$ sent by the corrupted nodes for certain l, t, we can recover the matrices $M_{\alpha_l,t}$ as follows:

For R' in Algorithm 5.14, we have $\Phi_{DC} \cdot M' = R'$, where $\Phi_{DC} = \mathbf{W}_{0,q^2-1,l} = [\Omega_{DC}, \Delta_{DC}]$, $R' = [R'_1, R'_2]$. Ω_{DC}, R'_1 are $q^2 \times k_l$ submatrices and Δ_{DC}, R'_2 are $q^2 \times (\alpha_l - k_l)$ submatrices.

According to equation (5.69), we have

$$\Phi_{DC} \cdot M' = [\Omega_{DC} S' + \Delta_{DC} T'^T, \Omega_{DC} T'] = [R'_1, R'_2]. \tag{5.79}$$

For $R_2' = \Omega_{DC} T'$, we can view each column of R_2' as a $(q^2, k_l, q^2 - k_l + 1)$ MDS code because Φ_{DC} is a Vandermonde matrix. If the number of erasures σ (corresponding to the previously detected corrupted nodes) and the number of corrupted nodes τ that have not been detected satisfy:

$$\sigma + 2\tau \leq q^2 - k_l, \tag{5.80}$$

then all the columns of T' can be recovered and the error locations (corresponding to the corrupted nodes) can be pinpointed. After T' has been recovered, we can recover S' following the same process because $\Omega_{DC} S' = R_1' - \Delta_{DC} T'^T$. So DC can reconstruct $M_{\alpha_l, t}$.

5.5 Performance Analysis

In this section, we analyze the performance of the H-MSR code and compare it with the performance of the RS-MSR code. We will first analyze their error correction capability then their complexity.

The comparison results between the H-MBR code and the RS-MBR code are the same since the error correction capability and the complexity of the H-MSR code and the H-MBR code are similar while these performance parameters of the RS-MSR code and the RS-MBR code are similar.

5.5.1 Scalable Error Correction

5.5.1.1 Error correction for data regeneration

The RS-MSR code in [83] can correct up to τ errors by downloading symbols from $d + 2\tau$ nodes. However, the number of errors may vary in the symbols sent by helper nodes. When there is no error or the number of errors is far less than τ, downloading symbols from extra nodes will be a waste of bandwidth. When the number of errors is larger than τ, the decoding process will fail without being detected. In this case, the symbols stored in the replacement

node will be erroneous. If this erroneous node becomes a helper node later, the errors will propagate to other nodes.

The H-MSR code can detect the erroneous decodings using Algorithm 5.3. If no error is detected, regeneration of H-MSR only needs to download symbols from one more node than the regeneration in error-free network, while the extra cost for the RS-MSR code is 2τ. If errors are detected in the symbols received from the helper nodes, the H-MSR code can correct the errors using Algorithm 5.4. Moreover, the algorithm can determine whether the decoding is successful, while the RS-MSR code is unable to provide such information.

5.5.1.2 Error correction for data reconstruction

The evaluation result is similar to the data regeneration. The RS-MSR code can correct up to τ errors with support from 2τ additional helper nodes. The H-MSR code is more flexible. For error detection, it only requires symbols from one additional node using Algorithm 5.6. The errors can then be corrected using Algorithm 5.7. The algorithm can also determine whether the decoding is successful.

5.5.2 Error Correction Capability

For data regeneration described in Algorithm 5.4, H-MSR code can be viewed as q MDS codes with parameters (q^2-1, d_l, q^2-d_l), $l = 0, \cdots, q-1$. Since $\alpha_l \leq \kappa(l)$ and $\kappa(l)$ is strictly decreasing, we can choose the sequence α_l to be strictly decreasing. So d_l is also strictly decreasing. For the q MDS codes, the minimum distance of the $(q^2-1, d_{q-1}, q^2-d_{q-1})$ code is the largest. In Algorithm 5.4, this code is decoded first and it can correct up to $\tau_{q-1} = \lfloor (q^2 - d_{q-1} - 1)/2 \rfloor$ errors, where $\lfloor x \rfloor$ is the floor function of x. Then the code (q^2-1, d_l, q^2-d_l), $l = q-2, \cdots, 0$, will be decoded sequentially. The (q^2-1, d_l, q^2-d_l) code can correct at most $\tau_l = \tau_{q-1}$ errors when $q^2 - d_0 - 1 \geq \tau_{q-1}$. Thus, the total numbers errors that the H-MSR code can correct is $\tau_{H-MSR} = q \cdot \tau_{q-1}$. While the $(q^3 -$

$q, \sum_{l=0}^{q-1} d_l, q^3 - q - \sum_{l=0}^{q-1} d_l + 1)$ RS-MSR code with the same rate can correct $\tau_{RS-MSR} = \lfloor (q^3 - q - \sum_{l=0}^{q-1} d_l)/2 \rfloor$ errors. Therefore, we have the following theorem.

Theorem 5.7. *For data regeneration, the number of errors that the H-MSR code and the RS-MSR code can correct satisfy $\tau_{H-MSR} > \tau_{RS-MSR}$ when $q \geq 3$.*

Proof. For τ_{RS-MSR}, we have

$$\begin{aligned} \tau_{RS-MSR} &= \left\lfloor \left(q^3 - q - \sum_{l=0}^{q-1} d_l \right) /2 \right\rfloor \\ &\leq \left\lfloor (q^3 - q - q \cdot d_{q-1} - \frac{q}{2}(q-1))/2 \right\rfloor \\ &= \left\lfloor q \cdot (q^2 - d_{q-1} - 1)/2 - \frac{q(q-1)}{4} \right\rfloor \\ &\leq q \cdot (q^2 - d_{q-1} - 1)/2 - \frac{q(q-1)}{4}. \end{aligned} \tag{5.81}$$

For τ_{H-MSR}, we have

$$\tau_{H-MSR} = q \cdot \lfloor (q^2 - d_{q-1} - 1)/2 \rfloor. \tag{5.82}$$

When $q = 3$, it is easy to verify that $\tau_{H-MSR} > \tau_{RS-MSR}$.

When $q > 3$, We can rewrite equation (5.82) as

$$\tau_{H-MSR} \geq q \cdot (q^2 - d_{q-1} - 1)/2 - q/2. \tag{5.83}$$

The gap between τ_{H-MSR} and τ_{RS-MSR} is at least

$$\frac{q(q-1)}{4} - \frac{q}{2} = \frac{q^2 - 3q}{4} > 0 \, , q > 3, \tag{5.84}$$

so we have $\tau_{H-MSR} > \tau_{RS-MSR}$. □

Example 1. *Suppose $q = 4$ and $m = 37$, the Hermitian curve is defined by $y^4 + y = x^5$ over $GF(4^2)$. From the previous discussion, we have $\kappa(0) = 10, \kappa(1) = 9, \kappa(2) = 7, \kappa(3) = 6$. Choose $\alpha_0 = 6, \alpha_1 = 5, \alpha_2 = 4, \alpha_3 = 3$. So $d_0 = 12, d_1 = 10, d_2 = 8, d_3 = 6$. According to the analysis above, we have $\tau_{H-MSR} = 4 \cdot \tau_3 = 4 \cdot \lfloor (15-6)/2 \rfloor = 16$, which is larger than $\tau_{RS-MSR} = \lfloor (60-36)/2 \rfloor = 12$.*

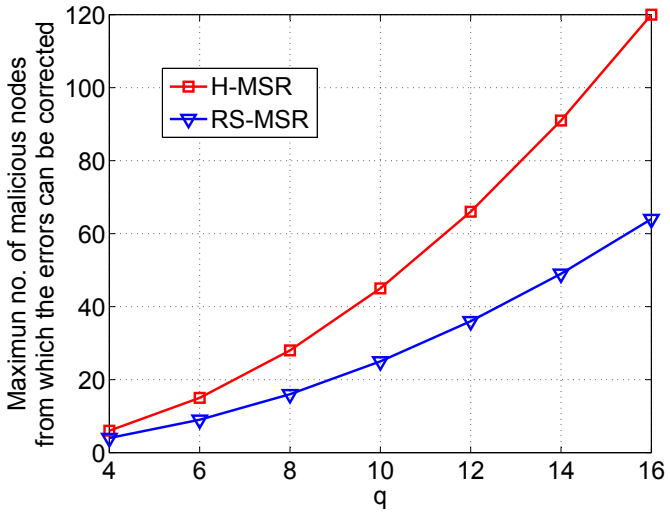

Figure 5.4 Comparison of error correction capability between the H-MSR code and the RS-MSR code

We also show the maximum number of malicious nodes from which the errors can be corrected by the H-MSR code in Figure. 5.4. Here we set the parameter q of the Hermitian code from 4 to 16 with a step of 2. In the figure, the performance of the RS-MSR code with the same code rates as the H-MSR code is also plotted. The comparison result further demonstrates that for data regeneration the H-MSR code has better error correction capability than the RS-MSR code.

For data reconstruction Algorithm 5.7, H-MSR code can be viewed as q MDS codes with parameters $(q^2-1, k_l-1, q^2-k_l+1)$. The decoding for the reconstruction is performed from the code with the largest minimum distance to the code with the smallest minimum distance as in the data regeneration case. Similarly, we can conclude that for data reconstruction the H-MSR code has better error correction capability than the RS-MSR code under the same code rate.

5.5.3 Complexity Discussion

For the complexity of the H-MSR code, we consider two scenarios.

5.5.3.1 H-MSR regeneration

For the H-MSR regeneration, compared with RS-MSR code, the H-MSR code will slightly increase the complexity of the helper nodes. For each helper node, the extra operation is a matrix multiplication between B_i^{-1} and Y_i. The complexity is $O(q^2) = \mathcal{O}((n^{1/3})^2) = \mathcal{O}(n^{2/3})$. Similar to [87], for a replacement node, from Algorithm 5.2 and Algorithm 5.3, we can derive that the complexity to regenerate symbols for RS-MSR is $\mathcal{O}(n^2)$, while the complexity for H-MSR is only $\mathcal{O}(n^{5/3})$. Likewise, for Algorithm 5.4, the complexity to recover the H-MSR code is $\mathcal{O}(n^{5/3})$, and $\mathcal{O}(n^2)$ for RS-MSR code.

5.5.3.2 H-MSR reconstruction

For the reconstruction, compared with RS-MSR code, the additional complexity of the H-MSR code for each storage node is $\mathcal{O}(q^2)$, which is $\mathcal{O}(n^{2/3})$. The computational complexity for DC to reconstruct the data is $\mathcal{O}(n^{5/3})$ for the H-MSR code and $\mathcal{O}(n^2)$ for the RS-MSR code.

CHAPTER 6

DISTRIBUTED STORAGE IN HOSTILE NETWORKS — OPTIMAL CONSTRUCTION OF REGENERATING CODES THROUGH RATE-MATCHING APPROACH

Inspired by the nice performance of Hermitian code based regenerating codes, we will step forward in this chapter to further construct optimal regenerating codes which have similar layered structure like Hermitian code in distributed storage. Compared to the Hermitian based code, these codes have simpler structure and are easier to understand and implement. We will propose two optimal constructions of MSR codes through rate-matching in hostile networks: 2-layer rate-matched MSR code and m-layer rate-matched MSR code. For the 2-layer code, we can achieve the optimal storage efficiency for given system requirements. Our comprehensive analysis shows that our code can detect and correct malicious nodes with higher storage efficiency compared to the RS-MSR code. Then we will propose the m-layer code by extending the 2-layer code and achieve the optimal error correction efficiency by matching the code rate of each layer's MSR code. We will also demonstrate that the optimized parameter can achieve the maximum storage capacity under the same constraint. Compared to the RS-MSR code, our code can achieve much higher error correction efficiency. The optimized m-layer code also has better error correction capability than the H-MSR code.

6.1 System/Adversarial Models and Assumptions

The system/adversarial models and assumptions in this chapter are the same with Chapter 5. We use the notation CH to refer to either the full rate MSR code or a codeword of the full rate MSR code. The exact meaning can be discriminated clearly according to the context.

6.2 Component Codes of Rate-matched MSR Code

In this section, we will introduce two different component codes for rate-matched MSR code on the MSR point with $d = 2k - 2$. The code based on the MSR point with $d > 2k - 2$ can be derived the same way through truncating operations. In the rate-matched MSR code, there are two types of MSR codes with different code rates: full rate code and fractional rate code.

6.2.1 Full Rate Code

6.2.1.1 Encoding

The full rate code is encoded based on the product-matrix code framework in [57]. According to equation (2.7), we have $\alpha_H = d/2$, $\beta_H = 1$ for one block of data with the size $B_H = (\alpha+1)\alpha$. The data will be arranged into two $\alpha \times \alpha$ symmetric matrices S_1, S_2, each of which will contain $B_H/2$ data. The codeword CH is defined as

$$CH = [\Phi \quad \Lambda\Phi] \begin{bmatrix} S_1 \\ \\ S_2 \end{bmatrix} = \Psi S_H, \tag{6.1}$$

where

$$\Phi = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \phi & \phi^2 & \dots & \phi^{\alpha-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \phi^{n-1} & (\phi^{n-1})^2 & \dots & (\phi^{n-1})^{\alpha-1} \end{bmatrix} \tag{6.2}$$

is a Vandermonde matrix and $\Lambda = \text{diag}[\lambda_1, \lambda_2, \cdots, \lambda_\alpha]$ such that $\lambda_i \neq \lambda_j$ for $1 \leq i, j \leq \alpha, i \neq j$, where $\lambda_i \in GF_q$ for $1 \leq i \leq \alpha$, ϕ is a primitive element in GF_q, and any d rows of

Ψ are linear independent. Then each row $\mathbf{ch}_i$, $0 \leq i < n$, of the codeword matrix CH will be stored in storage node i, in which the encoding vector ν_i is the i^{th} row of Ψ.

6.2.1.2 Regeneration

Suppose node z fails, the replacement node z' will send regeneration requests to the rest of $n-1$ helper nodes. Upon receiving the regeneration request, helper node i will calculate and send out the help symbol $p_i = \mathbf{ch}_i \mu_z^T$, where μ_z is the z^{th} row of Φ. z' will perform Algorithm 6.1 to regenerate the contents of the failed node z. For convenience, we define $\mathbf{V}_{i,j} = \left[\nu_i^T, \nu_{i+1}^T \cdots, \nu_j^T\right]^T$, where ν_t, $i \leq t \leq j$, is the t^{th} row of Ψ and $\mathbf{x}^{(j)}$ is the vector containing the first j symbols of $S_H \mu_z^T$.

Algorithm 6.1 z' Regenerates Symbols of the Failed Node z

Suppose $p'_i = p_i + e_i$ is the response from the i^{th} helper node. If p_i has been modified by the malicious node i, we have $e_i \in GF_q \backslash \{0\}$. We can successfully regenerate the symbols in node z when the errors in the received help symbols ${p_i}'$ from $n-1$ helper nodes can be corrected. Without loss of generality, we assume $0 \leq i \leq n-2$.

Step 1: Decode $\mathbf{p'}$ to $\mathbf{p}_{cw}$, where $\mathbf{p'} = [p'_0, p'_1, \cdots, p'_{n-2}]^T$ can be viewed as an MDS code with parameters $(n-1, d, n-d)$ since $\mathbf{V}_{0,n-2} \cdot \mathbf{x}^{(n-1)} = \mathbf{p'}$.

Step 2: Solve $\mathbf{V}_{0,n-2} \cdot \mathbf{x}^{(n-1)} = \mathbf{p}_{cw}$ and compute $\mathbf{ch}_z = \mu_z S_1 + \lambda_z \mu_z S_2$ as described in [57].

6.2.1.3 Reconstruction

When DC needs to reconstruct the original file, it will send reconstruction requests to n storage nodes. Upon receiving the request, node i will send out the symbol vector $\mathbf{c}_i$. Suppose $\mathbf{c}'_i = \mathbf{c}_i + \mathbf{e}_i$ is the response from the i^{th} storage node. If $\mathbf{c}_i$ has been modified by the malicious node i, we have $\mathbf{e}_i \in (GF_q)^\alpha \backslash \{\mathbf{0}\}$.

Then DC will reconstruct the file as follows: Let $R' = [{\mathbf{ch}'_0}^T, {\mathbf{ch}'_1}^T, \cdots, {\mathbf{ch}'_{n-1}}^T]^T$, we

have

$$\Psi_{DC}\begin{bmatrix} S_1' \\ S_2' \end{bmatrix} = [\Phi_{DC} \quad \Lambda_{DC}\Phi_{DC}]\begin{bmatrix} S_1' \\ S_2' \end{bmatrix} = \mathbf{V}_{0,n-1}\begin{bmatrix} S_1' \\ S_2' \end{bmatrix} = R',$$

$$\Phi_{DC}S_1'\Phi_{DC}^T + \Delta_{DC}\Phi_{DC}S_2'\Phi_{DC}^T = R'\Phi_{DC}^T. \tag{6.3}$$

Let $C = \Phi_{DC}S_1'\Phi_{DC}^T$, $D = \Phi_{DC}S_2'\Phi_{DC}^T$, and $\widehat{R}' = R'\Phi_{DC}^T$, then

$$C + \Delta_{DC}D = \widehat{R}'. \tag{6.4}$$

Since C, D are both symmetric, we can solve the non-diagonal elements of C, D as follows:

$$\begin{cases} C_{i,j} + \lambda_i \cdot D_{i,j} = \widehat{R}'_{i,j} \\ C_{i,j} + \lambda_j \cdot D_{i,j} = \widehat{R}'_{j,i}. \end{cases} \tag{6.5}$$

Because matrices C and D have the same structure, here we only focus on C (corresponding to S_1'). It is straightforward to see that if node i is malicious and there are errors in the i^{th} row of R', there will be errors in the i^{th} row of $\widehat{R}'$. Furthermore, there will be errors in the i^{th} row and i^{th} column of C. Define $S_1'\Phi_{DC}^T = \widehat{S}_1'$, we have $\Phi_{DC}\widehat{S}_1' = C$. Here we can view each column of C as an $(n-1, \alpha, n-\alpha)$ MDS code because Φ_{DC} is a Vandermonde matrix. The length of the code is $n-1$ since the diagonal elements of C is unknown. Suppose node j is a legitimate node, we can decode the MDS code to recover the j^{th} column of C and locate the malicious nodes. Eventually C can be recovered. So DC can reconstructs S_1 using the method similar to [57]. For S_2, the recovering process is similar.

6.2.2 Fractional Rate Code

6.2.2.1 Encoding

For the fractional rate code, we also have $\alpha_L = d/2$, $\beta_L = 1$ for one block of data with the size

$$B_L = \begin{cases} xd(1+xd)/2, x \in (0, 0.5] \\ \alpha(\alpha+1)/2+(x-0.5)d(1+(x-0.5)d), x \in (0.5,1] \end{cases}, \tag{6.6}$$

where x is the match factor of the rate-matched MSR code. It is easy to see that the fractional rate code will become the full rate code with $x = 1$. The data $\mathbf{m} = [m_1, m_2, \ldots, m_{B_L}] \in GF_q^{B_L}$ will be processed as follows:

When $x \leq 0.5$, the data will be arranged into a symmetric matrix S_1 of the size $\alpha \times \alpha$:

$$S_1 = \begin{bmatrix} m_1 & m_2 & \ldots & m_{xd} & 0 & \ldots & 0 \\ m_2 & m_{xd+1} & \ldots & m_{2xd-1} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ m_{xd} & m_{2xd-1} & \ldots & m_{B_L/2} & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix}. \tag{6.7}$$

The codeword CL is defined as

$$CL = [\Phi \quad \Lambda\Phi] \begin{bmatrix} S_1 \\ \mathbf{0} \end{bmatrix} = \Psi S_L, \tag{6.8}$$

where $\mathbf{0}$ is the $\alpha \times \alpha$ zero matrix and Φ, Λ, Ψ are the same as the full rate code.

When $x > 0.5$, the first $\alpha(\alpha+1)/2$ data will be arranged into an $\alpha \times \alpha$ symmetric matrix S_1. The rest of the data $m_{\alpha(\alpha+1)/2+1}, \ldots, m_{B_L}$ will be arranged into another $\alpha \times \alpha$ symmetric matrix S_2:

$$S_2 = \begin{bmatrix} m_{\alpha(\alpha+1)/2+1} & \cdots & m_{\alpha(\alpha+1)/2+xd} & 0 \ldots 0 \\ m_{\alpha(\alpha+1)/2+2} & \cdots & m_{\alpha(\alpha+1)/2+2xd-1} & 0 \ldots 0 \\ \vdots & \ddots & \vdots & \vdots \ddots \vdots \\ m_{\alpha(\alpha+1)/2+xd} & \cdots & m_{B_L/2} & 0 \ldots 0 \\ 0 & \ldots & 0 & 0 \ldots 0 \\ \vdots & \ddots & \vdots & \vdots \ddots \vdots \\ 0 & \ldots & 0 & 0 \ldots 0 \end{bmatrix}. \tag{6.9}$$

The codeword CL is defined the same as equation (6.1) with the same parameters Φ, Λ and Ψ.

Then each row $\mathbf{cl}_i$, $0 \leq i < n$, of the codeword matrix CL will be stored in storage node i respectively, in which the encoding vector ν_i is the i^{th} row of Ψ.

6.2.2.2 Regeneration

The regeneration for the fractional rate code is the same as the regeneration for the full rate code described in Section 6.2.1.2 with only a minor difference. If we define $\mathbf{x}^{(j)}$ as the vector containing the first j symbols of $S_L \mu_z^T$, there will be only xd nonzero elements in the vector. According to $\mathbf{V}_{0,n-2} \cdot \mathbf{x}^{(n-1)} = \mathbf{p'}$, the received symbol vector $\mathbf{p'}$ for the fractional rate code in **Step 1** of Algorithm 6.1 can be viewed as an $(n-1, xd, n-xd)$ MDS code. Since $x < 1$,

we can detect and correct more errors in data regeneration using the fractional rate code than using the full rate code.

6.2.2.3 Reconstruction

The reconstruction for the fractional rate code is similar to that for the full rate code described in Section 6.2.1.3. Let $R' = [\mathbf{cl'}_0{}^T, \mathbf{cl'}_1{}^T, \cdots, \mathbf{cl'}_{n-1}{}^T]^T$.

When the match factor $x > 0.5$, reconstruction for the fractional rate code is the same to that for the full rate code.

When $x \leq 0.5$, equation (6.3) can be written as:

$$\Phi_{DC} S'_1 = R'. \tag{6.10}$$

So we can view each column of R' as an $(n, xd, n - xd + 1)$ MDS code. After decoding R' to R_{cw}, we can recover the data matrix S_1 by solving the equation $\Phi_{DC} S_1 = R_{cw}$. Meanwhile, if the i^{th} rows of R' and R_{cw} are different, we can mark node i as corrupted.

6.3 2-Layer Rate-matched MSR Code

In this section, we will show our first optimization of the rate-matched MSR code: 2-layer rate-matched MSR code. In the code design, we utilize two layers of the MSR code: the fractional rate code for one layer and the full rate code for the other. The purpose of the fractional rate code is to correct the erroneous symbols sent by malicious nodes and locate the corresponding malicious nodes. Then we can treat the errors in the received symbols as erasures when regenerating with the full rate code. However, the rates of the two codes must match to achieve an optimal performance. Here we mainly focus on the rate-matching for data regeneration. We can see in the later analysis that the performance of data reconstruction can also be improved with this design criterion.

The main idea of this optimization is: first fixing the error correction capabilities of the fractional rate code and the full rate code by making their rate matched, then optimizing the data storage efficiency by adjusting the number of data blocks of different codes.

6.3.1 Rate Matching

From the analysis above, we know that during regeneration the fractional rate code can correct up to $\lfloor (n - xd - 1)/2 \rfloor$ errors, which are more than $\lfloor (n - d - 1)/2 \rfloor$ errors that the full rate code can correct. In the 2-layer rate-matched MSR code design, our goal is to match the fractional rate code with the full rate code. The main task for the fractional rate code is to detect and correct errors, while the main task for the full rate code is to maintain the storage efficiency. So if the fractional rate code can locate all the malicious nodes, the full rate code can simply treat the symbols sent from these malicious nodes as erasures, which requires the minimum redundancy for the full rate code. The full rate code can correct up to $n - d - 1$ erasures. Thus we have the following optimal rate-matching equation:

$$\lfloor (n - xd - 1)/2 \rfloor = n - d - 1, \tag{6.11}$$

from which we can derive the match factor x.

6.3.2 Encoding

To encode a file with size B_F using the 2-layer rate-matched MSR code, the file will first be divided into θ_H blocks of data with the size B_H and θ_L blocks of data with the size B_L, where the parameters should satisfy

$$B_F = \theta_H B_H + \theta_L B_L. \tag{6.12}$$

Then the θ_H blocks of data will be encoded into code matrices $CH_1, \ldots, CH_{\theta_H}$ using the full rate code and the θ_L blocks of data will be encoded into code matrices $CL_1, \ldots, CL_{\theta_L}$ using the fractional rate code. To prevent the malicious nodes from corrupting the fractional

rate code only, the secure server will randomly concatenate all the matrices together to form the final $n \times \alpha(\theta_H + \theta_L)$ codeword matrix:

$$CM = [\text{Perm}(CH_1, \ldots, CH_{\theta_H}, CL_1, \ldots, CL_{\theta_L})], \tag{6.13}$$

where Perm$(\cdot)$ is the random matrices permutation operation. The secure sever will also record the order of the permutation for future code regeneration and reconstruction. Then each row $\mathbf{c}_i = [\text{Perm}(\mathbf{ch_{1},}_i, \ldots, \mathbf{ch}_{\theta_\mathbf{H},i}, \mathbf{cl_{1},}_i, \ldots, \mathbf{cl}_{\theta_\mathbf{L},i})]$, $0 \leq i \leq n-1$, of the codeword matrix CM will be stored in storage node i, where $\mathbf{ch_{j},}_i$ is the i^{th} row of CH_j, $1 \leq j \leq \theta_H$, and $\mathbf{cl_{j},}_i$ is the i^{th} row of CL_j, $1 \leq j \leq \theta_L$. The encoding vector ν_i for storage node i is the i^{th} row of Ψ in equation (6.1).

6.3.3 Regeneration

Suppose node z fails, the security server will initialize a replacement node z' with the order information of the fractional rate code and the full rate code in the 2-layer rate-matched MSR code. Then the replacement node z' will send regeneration requests to the rest of $n-1$ helper nodes. Upon receiving the regeneration request, helper node i will calculate and send out the help symbol $p_i = \mathbf{c}_i \mu_z^T$. z' will perform Algorithm 6.2 to regenerate the contents of the failed node z. After the regeneration is finished, z' will erase the order information. So even if z' was compromised later, the adversary would not get the permutation order of the fractional rate code and the full rate code.

Algorithm 6.2 z' Regenerates Symbols of the Failed Node z for the 2-layer Rate-matched MSR Code

Step 1: According to the order information, regenerate all the symbols related to the θ_L data blocks encoded by the fractional rate code, using Algorithm 6.1. If errors are detected in the symbols sent by node i, it will be marked as a malicious node.

Step 2: Regenerate all the symbols related to the θ_H data blocks encoded by the full rate code, using Algorithm 6.1. During the regeneration, all the symbols sent from nodes marked as malicious nodes will be replaced by erasures $\bigotimes$.

It is easy to see that Algorithm 6.2 can correct errors and locate malicious node using the fractional rate code while achieve high storage efficiency using the full rate code.

6.3.4 Parameters Optimization

We have the following design requirements for a given distributed storage system applying the 2-layer rate-matched MSR code:

- The maximum number of malicious nodes M that the system can detect and locate using the fractional rate code. We have

$$\lfloor (n - xd - 1)/2 \rfloor = M. \tag{6.14}$$

- The probability P_{det} that the system can detect all the malicious nodes. The detection will be successful if each malicious node modifies at least one help symbol corresponding to the fractional rate code and sends it to the replacement node. Suppose the malicious nodes modify each help symbol to be sent to the replacement node with probability P, we have

$$(1 - (1 - P)^{\theta_L})^M \geq P_{det}. \tag{6.15}$$

So there is a trade-off between θ_L and θ_H: the number of data blocks encoded by the fractional rate code and the number of data blocks encoded by the full rate code. If we encode using too much full rate code, we may not meet the detection probability P_{det} requirement. If too much fractional rate code is used, the redundancy may be too high.

The storage efficiency is defined as the ratio between the actual size of data to be stored and the total storage space needed by the encoded data:

$$\delta_S = \frac{\theta_H B_H + \theta_L B_L}{(\theta_H + \theta_L) n\alpha} = \frac{B_F}{(\theta_H + \theta_L) n\alpha}. \tag{6.16}$$

Thus we can calculate the optimized parameters x, d, θ_H, θ_L by maximizing equation (6.16) under the constraints defined by equations (6.11), (6.12), (6.14), (6.15).

d and x can be determined by equation (6.11) and (6.14):

$$d = n - M - 1, \tag{6.17}$$

$$x = (n - 2M - 1)/(n - M - 1). \tag{6.18}$$

Since B_F is constant, to maximize δ_S is equal to minimize $\theta_H + \theta_L$. So we can rewrite the optimization problem as follows:

$$\text{Minimize } \theta_H + \theta_L, \text{ subject to (6.12) and (6.15).} \tag{6.19}$$

This is a simple linear programming problem. Here we will show the optimization results directly:

$$\theta_L = log_{(1-P)}(1 - P_{det}^{1/M}), \tag{6.20}$$

$$\theta_H = (B_F - \theta_L B_L)/B_H. \tag{6.21}$$

We assume that we are storing large files, which means $B_F > \theta_L B_L$. So an optimal solution for the 2-layer rate-matched MSR code can always be found. We have the following theorem:

Theorem 6.1. *When the number of blocks of the fractional rate code θ_L equals to $log_{(1-P)}(1-P_{det}^{1/M})$ and the number of blocks of the full rate code θ_H equals to $(B_F - \theta_L B_L)/B_H$, the 2-layer rate-matched MSR code can achieve the optimal storage efficiency.*

6.3.5 Reconstruction

When DC needs to reconstruct the original file, it will send reconstruction requests to n storage nodes. Upon receiving the request, node i will send out the symbol vector $\mathbf{c}_i$. Suppose $\mathbf{c}_i' = \mathbf{c}_i + \mathbf{e}_i$ is the response from the i^{th} storage node. If $\mathbf{c}_i$ has been modified by the malicious node i, we have $\mathbf{e}_i \in (GF_q)^{\alpha(\theta_L+\theta_H)} \backslash \{\mathbf{0}\}$. Since DC has the permutation information of the fractional rate code and the full rate code, similar to the regeneration of the 2-layer rate-matched MSR code, DC will perform the reconstruction using Algorithm 6.3.

Algorithm 6.3 DC Reconstructs the Original File for the 2-layer Rate-matched MSR Code

Step 1: According to the order information, reconstruct each of the θ_L data blocks encoded by the fractional rate code and locate the malicious nodes.

Step 2: Reconstruct each of the data blocks encoded by the full rate code. During the reconstruction, all the symbols sent from malicious nodes will be replaced by erasures $\bigotimes$.

6.3.5.1 Optimized Parameters

In Section 6.3.4, we optimized the parameters for the data regeneration, considering the trade-off between the successful malicious node detection probability and the storage efficiency. Here we will show that the same parameters can guarantee that the same constraints be satisfied for the data reconstruction.

- The maximum number of malicious nodes can be detected for the data reconstruction is no smaller that M: if $x > 0.5$, the number is $\lfloor (n-\alpha-1)/2 \rfloor$. We have $\lfloor (n-\alpha-1)/2 \rfloor \geq \lfloor (n-xd-1)/2 \rfloor = M$. If $x \leq 0.5$, the number is $\lfloor (n-xd)/2 \rfloor$. We have $\lfloor (n-xd)/2 \rfloor \geq \lfloor (n-xd-1)/2 \rfloor = M$

- The successful malicious node detection probability for the data reconstruction is larger than P_{det}: the probability is $(1-(1-P)^{\alpha\theta_L})^M$, so we have $(1-(1-P)^{\alpha\theta_L})^M > (1-(1-P)^{\theta_L})^M \geq P_{det}$.

Although the rate-matching equation (6.11) does not apply to the data reconstruction, the reconstruction strategy in Algorithm 6.3 can still benefit from the different rates of the two codes. When $x \leq 0.5$, the fractional rate code can detect and correct $\lfloor (n-xd)/2 \rfloor$ malicious nodes, which are more than $\lfloor (n-d/2-1)/2 \rfloor$ malicious nodes the full rate code can detect. When $x > 0.5$, the full rate code and the fractional rate code can detect and correct the same number of malicious nodes: $\lfloor (n-\alpha-1)/2 \rfloor$.

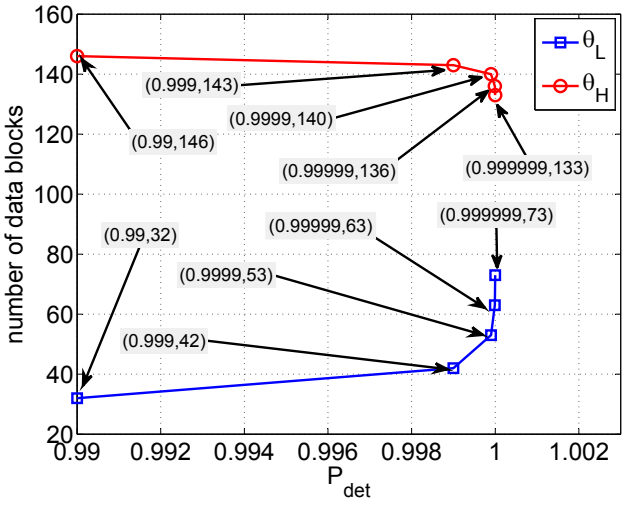

Figure 6.1 The number of fractional/full rate code blocks for different P_{det}

From the analysis above we can see that the same optimized parameters, which are obtained for the data regeneration, can maintain the optimized trade-off between the malicious node detection and storage efficiency for the data reconstruction.

6.3.6 Performance Evaluation

From the analysis above, we know that for a distributed storage system with n storage nodes out of which at most M nodes are malicious, the 2-layer rate-matched MSR code can guarantee detection and correction of the malicious nodes during the data regeneration and reconstruction with the probability at least P_{det}.

For a distributed storage system with $n = 30$, $M = 11$ and $P = 0.2$, suppose we have a file with the size $B_F = 14000M$ symbols to be stored in the system. The number of the fractional rate code blocks θ_L and the number of the full rate code blocks θ_H for different detection probabilities P_{det} are shown in Figure. 6.1. From the figure we can see that the number of fractional rate code blocks will increase when the detection probability becomes larger. Accordingly, the number of full rate code blocks will decrease.

For the RS-MSR code constructed in [83], the efficiency of the code with the same re-

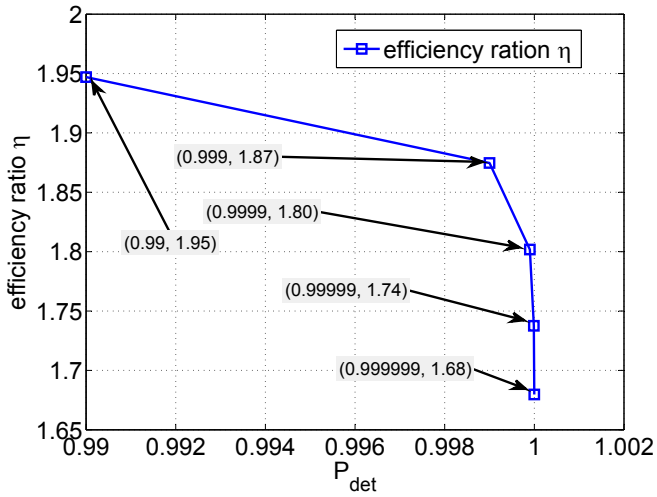

Figure 6.2 Efficiency ratios between the 2-layer rate-matched MSR code and the RS-MSR code for different P_{det}

generation performance as the 2-layer rate-matched MSR code is defined as

$$\delta'_S = \frac{\alpha'(\alpha'+1)}{\alpha' n} = \frac{\alpha'+1}{n} = \frac{xd/2+1}{n}. \tag{6.22}$$

In Figure. 6.2 we will show the efficiency ratios $\eta = \delta_S/\delta'_S$ between the 2-layer rate-matched MSR code and the RS-MSR code under different detection probabilities P_{det}. From the figure we can see that the 2-layer rate-matched MSR code has higher efficiency than the RS-MSR code. When the successful malicious nodes detection probability is 0.999999, the efficiency of the 2-layer rate-matched MSR code is about 70% higher.

6.4 m-Layer Rate-matched MSR Code

In this section, we will show our second optimization of the rate-matched MSR code: m-layer rate-matched MSR code. In the code design, we extend the design concept of the 2-layer rate-matched MSR code. Instead of encoding the data using two MSR codes with different match factors, we utilize m layers of the full rate MSR codes with different parameter d, written as d_i for layer L_i, $1 \leq i \leq m$, which satisfy

$$d_i \leq d_j, \ \forall 1 \leq i \leq j \leq m. \tag{6.23}$$

The data will be divided into m parts and each part will be encoded by a distinct full rate MSR code. According to the analysis above, the code with a lower code rate has better error correction capability.

The codewords will be decoded layer by layer in the order from layer L_1 to layer L_m. That is, the codewords encoded by the full rate MSR code with a lower d will be decoded prior to those encoded by the full rate MSR code with a higher d. If errors were found by the full rate MSR code with a lower d, the corresponding nodes would be marked as malicious. The symbols sent from these nodes would be treated as erasures in the subsequent decoding of the full rate MSR codes with higher d's. The purpose of this arrangement is to try to correct as many as erroneous symbols sent by malicious nodes and locate the corresponding malicious nodes using the full rate MSR code with a lower rate. However, the rates of the m full rate MSR codes must match to achieve an optimal performance. Here we mainly focus on the rate-matching for data regeneration. We can see in the later analysis that the performance of data reconstruction can also be improved with this design criterion.

The main idea of this optimization is to optimize the overall error correction capability by matching the code rates of different full rate MSR codes.

6.4.1 Rate Matching and Parameters Optimization

According to Section 6.2.1.2, the full rate MSR code CH_i for layer L_i can be viewed as an $(n-1, d_i, n-d_i)$ MDS code for $1 \leq i \leq m$. During the optimization, we set the summation of the d's of all the layers to a constant $\overline{d}$:

$$\sum_{i=1}^{m} d_i = \overline{d}. \tag{6.24}$$

Here we will show the optimization through an illustrative example first. Then we will present the general result.

6.4.1.1 Optimization for $m = 3$

There are three layers of full rate MSR codes for $m = 3$: CH_1, CH_2 and CH_3.

The first layer code CH_1 can correct t_1 errors:

$$t_1 = \lfloor (n - d_1 - 1)/2 \rfloor = (n - d_1 - 1 - \varepsilon_1)/2, \tag{6.25}$$

where $\varepsilon_1 = 0$ or 1 depending on whether $(n - d_1 - 1)/2$ is even or odd.

By regarding the symbols from the t_1 nodes where errors are found by CH_1 as erasures, the second layer code CH_2 can correct t_2 errors:

$$\begin{aligned} t_2 &= \lfloor (n - d_2 - 1 - t_1)/2 \rfloor + t_1 \\ &= (n - d_2 - 1 - t_1 - \varepsilon_2)/2 + t_1 \\ &= (2(n - d_2) + n - d_1 - 2\varepsilon_2 - \varepsilon_1 - 3)/4, \end{aligned} \tag{6.26}$$

where $\varepsilon_2 = 0$ or 1, with the restriction that $n - d_2 - 1 \geq t_1$, which can be written as:

$$-d_1 + 2d_2 \leq n + \varepsilon_1 - 1. \tag{6.27}$$

The third layer code CH_3 also treat the symbols from the t_2 nodes as erasures. CH_3 can correct t_3 errors:

$$\begin{aligned} t_3 &= \lfloor (n - d_3 - 1 - t_2)/2 \rfloor + t_2 \\ &= (n - d_3 - 1 - t_2 - \varepsilon_2)/2 + t_2 \\ &= (4(n - d_3) + 2(n - d_2) + n - d_1 - 4\varepsilon_3 - 2\varepsilon_2 - \varepsilon_1 - 7)/8, \end{aligned} \tag{6.28}$$

where $\varepsilon_3 = 0$ or 1, with the restriction that $n - d_3 - 1 \geq t_2$, which can be written as:

$$-d_1 - 2d_2 + 4d_3 \leq n + \varepsilon_1 + 2\varepsilon_2 - 1. \tag{6.29}$$

According to the analysis above, the d's of the three layers satisfy:

$$d_1 - d_2 \leq 0, \tag{6.30}$$

$$d_2 - d_3 \leq 0. \tag{6.31}$$

And we can rewrite equation (6.24) as:

$$d_1 + d_2 + d_3 \leq \overline{d}, \tag{6.32}$$

$$-d_1 - d_2 - d_3 \leq -\overline{d}. \tag{6.33}$$

To maximize the error correction capability of the m-layer rate-matched MSR code for $m = 3$, we have to maximize t_3, the number of errors that the third layer code CH_3 can correct, since t_3 has included all the malicious nodes from which errors are found by the codes of first two layers. With all the constraints listed above, the optimization problem can written as:

$$\begin{aligned} &\text{Maximize } t_3 \text{ in (6.28)}, \\ &\text{subject to (6.27), (6.29), (6.30), (6.31), (6.32), (6.33).} \end{aligned} \tag{6.34}$$

Now we have changed this optimization problem into a typical linear programming problem. After verifying this linear programming problem has a feasible solution, we solve it using the SIMPLEX algorithm [112]. When $d_1 = d_2 = d_3 = \text{Round}(\overline{d}/3) = \widetilde{d}$, the m-layer rate-matched MSR code can correct errors from at most

$$\begin{aligned} \widetilde{t}_3 &= (7n - 7\widetilde{d} - 4\varepsilon_3 - 2\varepsilon_2 - \varepsilon_1 - 7)/8 \\ &\geq (7n - 7\widetilde{d} - 14)/8 \text{ (worst case)} \end{aligned} \tag{6.35}$$

malicious nodes, where Round$(\cdot)$ is the rounding operation.

6.4.1.2 Evaluation of the Optimization for $m = 3$

Similar to the storage efficiency δ_S defined in Section 6.3, here we can define the error correction efficiency δ_C of the m-layer rate-matched MSR code as the ratio between the maximum number of malicious nodes that can be found and the total number of storage nodes in the network:

$$\delta_C = (7n - 7\widetilde{d} - 14)/(8n). \tag{6.36}$$

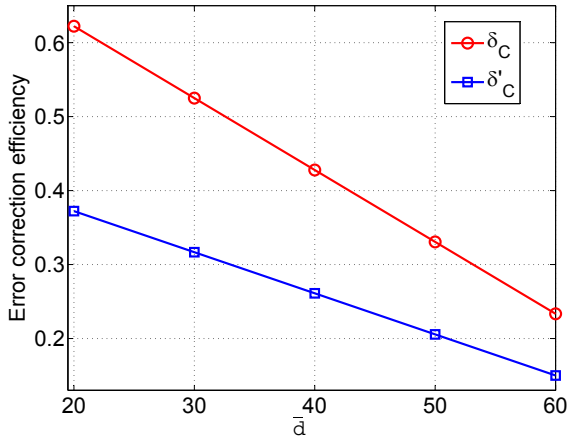

Figure 6.3 Comparison of the error correction capability between m-layer rate-matched MSR code for $m = 3$ and RS-MSR code

The RS-MSR code with the same code rate can be viewed as an $(n-1, \widetilde{d}, n-\widetilde{d})$ MDS code which can correct errors from at most $(n-\widetilde{d}-1)/2$ malicious nodes (best case). So the error correction efficiency δ'_C is

$$\delta'_C = (n - \widetilde{d} - 1)/(2n). \tag{6.37}$$

The comparison of the error correction capability between m-layer rate-matched MSR code for $m = 3$ and RS-MSR code is shown in Figure. 6.3. In this comparison, we set the number of storage nodes in the network $n = 30$. From the figure we can see that the m-layer rate-matched MSR code for $m = 3$ improves the error correction efficiency more than 50%.

6.4.1.3 General Optimization Result

For the general m-layer rate-matched MSR code, the optimization process is similar.

The first layer code CH_1 can correct t_1 errors as in equation (6.25). By regarding the symbols from the t_{i-1} nodes where errors are found by CH_{i-1} as erasures, the i^{th} layer code

can correct t_i errors for $2 \leq i \leq m$:

$$\begin{aligned} t_i &= \lfloor (n - d_i - 1 - t_{i-1})/2 \rfloor + t_{i-1} \\ &= (n - d_i - 1 - t_{i-1} - \varepsilon_i)/2 + t_{i-1} \\ &= (\textstyle\sum_{j=1}^{i} 2^{j-1}(n - d_j) - \sum_{j=1}^{i} 2^{j-1}\varepsilon_j - 2^i + 1)/2^i, \end{aligned} \tag{6.38}$$

where $\varepsilon_i = 0$ or 1, with the restriction that $n - d_i - 1 \geq t_{i-1}$, which can be written as:

$$-\sum_{j=1}^{i-1} 2^{j-1} d_j + 2^{i-1} d_i \leq n + \sum_{j=1}^{i-1} 2^{j-1}\varepsilon_j - 1. \tag{6.39}$$

Similarly, the parameter d of the i^{th} layer for $2 \leq i \leq m$ must satisfy

$$d_{i-1} - d_i \leq 0. \tag{6.40}$$

And equation (6.24) can be written as:

$$\sum_{j=1}^{m} d_j \leq \overline{d}, \tag{6.41}$$

$$-\sum_{j=1}^{m} d_j \leq -\overline{d}. \tag{6.42}$$

We can maximize the error correction capability of the m-layer rate-matched MSR code by maximizing t_m. With all the constrains listed above, the optimization problem can be written as:

$$\begin{aligned} &\text{Maximize} \quad t_i \text{ for } i = m \text{ in (6.38)}, \\ &\text{subject to} \quad \text{(6.39) and (6.40) for } 2 \leq i \leq m, \text{ (6.41), (6.42)}. \end{aligned} \tag{6.43}$$

After verifying that this linear programming problem has a feasible solution, we can use the SIMPLEX algorithm to solve it. The optimization result can be summarized as follows:

Theorem 6.2. *For the m-layer rate-matched MSR code, when*

$$d_i = \mathit{Round}(\overline{d}/m) = \widetilde{d} \quad \textit{for } 1 \leq i \leq m, \tag{6.44}$$

it can correct errors from at most

$$\begin{aligned}\widetilde{t}_m &= ((2^m-1)(n-\widetilde{d}) - \sum_{j=1}^{m} 2^{j-1}\varepsilon_j - 2^m + 1)/2^m \\ &\geq ((2^m-1)(n-\widetilde{d}) - 2^{m+1} + 2)/2^m \ \ (worst\ \ case). \end{aligned} \tag{6.45}$$

malicious nodes.

The error correction efficiency for the m-layer rate-matched MSR code is

$$\delta_C = ((2^m-1)(n-\widetilde{d}) - 2^{m+1} + 2)/(2^m n). \tag{6.46}$$

This is a monotonically increasing function for m, so we have:

Corollary 1. *The error correction efficiency of the m-layer rate-matched MSR code increases with m, which is the number of layers.*

Remark 6. *During the optimization, we set the code rate of the rate-matched MSR code to a constant value and maximize the error correction capability. To optimizing the rate-matched MSR code, we can also set the error correction capability* t_i *for* $i = m$ *in (6.38) to a constant value*

$$t_m = \bar{t} \tag{6.47}$$

and maximize the code rate. The problem can be written as:

$$\begin{aligned} &Maximize \quad \textstyle\sum_{j=1}^{m} d_j \\ &subject\ to \quad (6.39)\ and\ (6.40)\ for\ 2 \leq i \leq m,\ (6.47). \end{aligned} \tag{6.48}$$

The optimization result is the same as that of (6.43): when all the $d_i's$ *for* $1 \leq i \leq m$ *are the same, the code rate is maximized.* d_i, $1 \leq i \leq m$, *satisfies the following equation:*

$$d_i \geq n - \frac{2^m \bar{t} + 2^{m+1} - 2}{2^m - 1} \ \ (worst\ case). \tag{6.49}$$

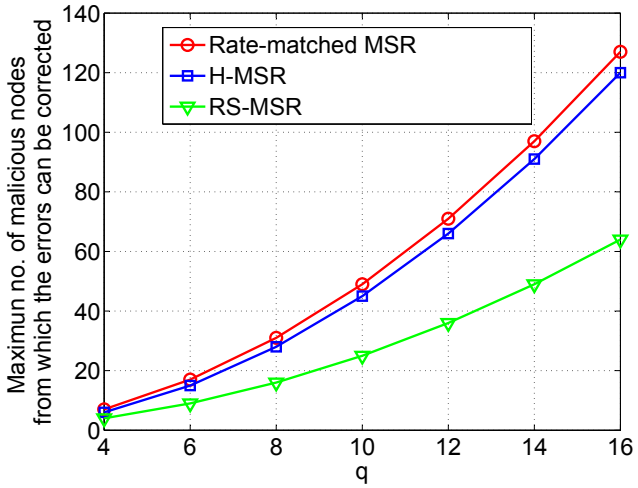

Figure 6.4 Comparison of error correction capability between the m-layer rate matched MSR code and the H-MSR code

6.4.1.4 Evaluation of the Optimization

Although at the beginning of this section we propose to decode the code with a lower rate first in the m-layer rate-matched MSR code, equation (6.44) shows that we can get the optimized error correction capability when all the rates of the codes in the m-layer code are equal. However, this result is not in conflict with our assumption in equation (6.23).

Comparison with the Hermitian code based MSR code in [88] The Hermitian code based MSR code (H-MSR code) in [88] has better error correction capability than the RS-MSR code. However, because the structure of the underlying Hermitian code is predetermined, the error correction capability might not be optimal. In figure 6.4, the maximum number of malicious nodes from which the errors can be corrected by the H-MSR code is shown. Here we set the parameter q of the Hermitian code [87] from 4 to 16 with a step of 2. In the figure, we also plot the performance of the m-layer rate-matched MSR code with the same code rates as the H-MSR code. The comparison result demonstrates that the rate-matched MSR code has better error correction capability than the H-MSR code. Moreover, the rate-matched code is easier to understand and has more flexibility than the H-MSR code.

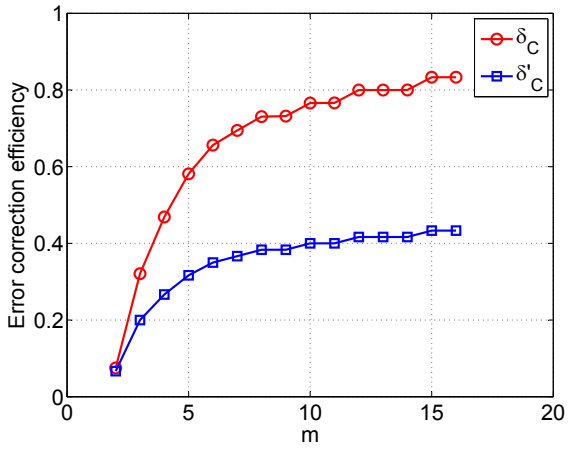

Figure 6.5 The optimal error correction efficiency of the m-layer rate-matched MSR code under different m for $2 \leq m \leq 16$

Relationship between the no. of layers and error correction efficiency Since we have seen the advantage of the rate-matched MSR code over the RS-MSR code in Section 6.4.1.2, here we will mainly discuss how the number of layers can affect the error correction efficiency. The error correction efficiency of the m-layer rate-matched MSR code is shown is Figure. 6.5, where we set $n = 30$ and $\overline{d} = 50$. We also plot the error correction efficiency ${\delta'}_C$ of the RS-MSR code with same code rates for comparison. From the figure we can see that when n and $\overline{d}$ are fixed, the optimal error correction efficiency will increase with the number of layers m as in Corollary 1.

Optimized storage capacity Moreover, the optimization condition in equation (6.44) also leads to maximum storage capacity besides the optimal error correction capability. We have the following theorem:

Theorem 6.3. *The m-layer rate-matched MSR code can achieve the maximum storage capacity if the parameter d's of all the layers are the same, under the constraint in equation (6.24).*

Proof. The code of the i^{th} layer can store one block of data with the size $B_i = \alpha_i(\alpha_i + 1) = (d_i/2)(d_i/2+1)$. So the m-layer code can store data with the size $B = \sum_{i=1}^{m}(d_i/2)(d_i/2+1)$.

Our goal here is to maximize B under the constraint in equation (6.24).

We can use Lagrange multipliers to find the point of maximum B. Let

$$\Lambda_L(d_1, \ldots, d_m, \lambda) = \sum_{i=1}^{m} (d_i/2)(d_i/2+1) + \lambda(\sum_{i=1}^{m} d_i - \overline{d}). \tag{6.50}$$

We can find the maximum value of B by setting the partial derivatives of this equation to zero:

$$\frac{\partial \Lambda_L}{\partial d_i} = \frac{d_i + 1}{2} - \lambda = 0, \ \ \forall 1 \leq i \leq m. \tag{6.51}$$

Here we can see that when all the parameter d's of all the layers are the same, we can get the maximum storage capacity B. This maximization condition coincides with the optimization condition for achieving the goal of this section: optimizing the overall error correction capability of the rate-matched MSR code. □

6.4.2 Practical Consideration of the Optimization

So far, we implicitly presume that there is only one data block of the size $B_i = \alpha_i(\alpha_i + 1)$ for each layer i. In practical distributed storage, it is the parameter d_i that is fixed instead of $\overline{d}$, the summation of d_i. However, as long as we use m layers of MSR codes with the same parameter $d = \widetilde{d}$, we will still get the optimal solution for $\overline{d} = m\widetilde{d}$. In fact, the m-layer rate-matched MSR code here becomes a single full rate MSR code with parameter $d = \widetilde{d}$ and m data blocks. And based on the dependent decoding idea we describe at the beginning of Section 6.4, we can achieve the optimal performance.

So when the file size B_F is larger than one data block size $\widetilde{B}$ of the single full rate MSR code with parameter $d = \widetilde{d}$, we will divide the file into $\lceil B_F/\widetilde{B} \rceil$ data blocks and encode them separately. If we decode these data blocks dependently, we can get the optimal error correction efficiency.

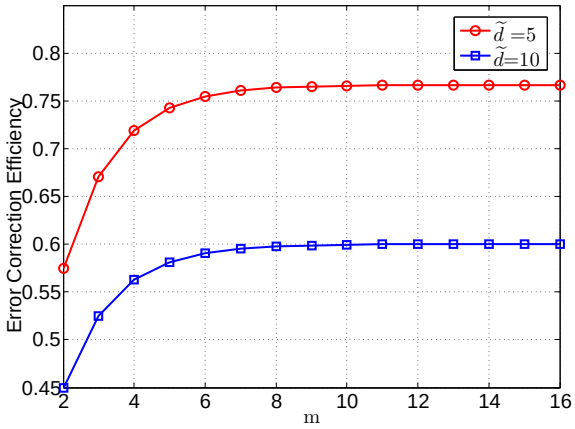

Figure 6.6 The optimal error correction efficiency for $2 \leq m \leq 16$

6.4.2.1 Evaluation of the Optimal Error Correction Efficiency

In the practical case, $\widetilde{d}$ in equation (6.46) is fixed. So here we will study the relationship between the number of dependently decoding data blocks m and the error correction efficiency δ_C, which is shown in Figure. 6.6. We set $n = 30$ and $\widetilde{d} = 5, 10$. From the figure we can see that although δ_C will become higher with the increasing of dependently decoding data blocks m, the efficiency improvement will be negligible for $m \geq 8$. Actually when $m = 7$ the efficiency has already become 99% of the upper bound of δ_C.

On the other hand, there exist parallel algorithms for fast MDS code decoding [113]. We can decode blocks of MDS codewords parallel in a pipeline fashion to accelerate the overall decoding speed. The more blocks of codewords we decode parallel, the faster we will finish the whole decoding process. For large files that could be divided into a large amount of data blocks (θ blocks), we can get a trade-off between the optimal error correction efficiency and the decoding speed by setting the number of dependently decoding data blocks m and the number of parallel decoding data blocks ρ under the constraint $\theta = m\rho$.

6.4.3 Encoding

From the analysis above we know that to encode a file with size B_F using the optimal m-layer rate-matched MSR code is to encode the file using a full rate MSR code with predetermined

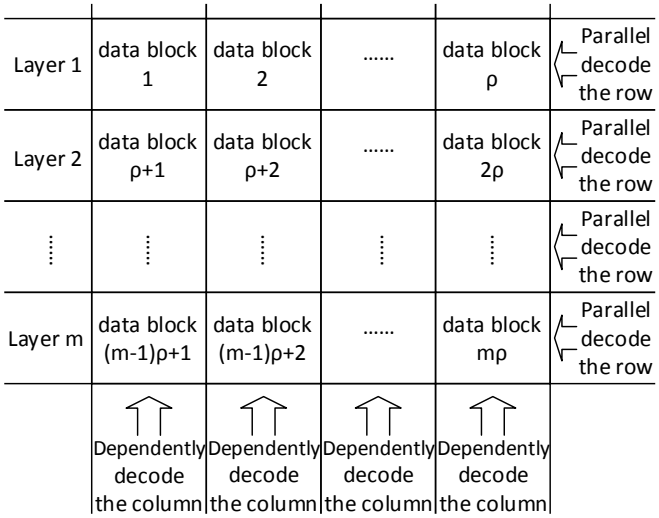

Figure 6.7 Lattice of received help symbols for regeneration

parameter $d = 2\alpha = \widetilde{d}$. First the file will be divided into θ blocks of data with size $\widetilde{B}$, where $\theta = \lceil B_F/\widetilde{B} \rceil$. Then the θ blocks of data will be encoded into code matrices $CH_1, \ldots, CH_\theta$ and form the final $n \times \alpha\theta$ codeword matrix: $CM = [CH_1, \ldots, CH_\theta]$. Each row $\mathbf{c}_i = [\mathbf{ch_1}_{,i}, \ldots, \mathbf{ch}_{\theta,i}]$, $0 \leq i \leq n-1$, of the codeword matrix CM will be stored in storage node i, where $\mathbf{ch_j}_{,i}$ is the i^{th} row of CH_j, $1 \leq j \leq \theta$. The encoding vector ν_i for storage node i is the i^{th} row of Ψ in equation (6.1).

6.4.4 Regeneration

Suppose node z fails, the replacement node z' will send regeneration requests to the rest of $n-1$ helper nodes. Upon receiving the regeneration request, helper node i will calculate and send out the help symbol $p_i = \mathbf{c}_i \mu_z^T$.

As we discuss above, combining both dependent decoding and parallel decoding can achieve the trade-off between optimal error correction efficiency and decoding speed. Although all θ blocks of data are encoded with the same MSR code, z' will place the received help symbols into a 2-dimension lattice with size $m \times \rho$ as shown in Figure. 6.7. In each

grid of the lattice there are $n-1$ help symbols corresponding to one data block, received from $n-1$ helper nodes. We can view each row of the lattice as related to a layer of an m-layer rate-matched MSR code with ρ blocks of data, which will be decoded parallel. We also view each column of the lattice as related to m layers of an m-layer rate-matched MSR code with one block of data each layer, which will be decoded dependently. z' will perform Algorithm 6.4 to regenerate the contents of the failed node z.

Algorithm 6.4 z' Regenerates Symbols of the Failed Node z for the m-layer Rate-matched MSR Code

---

Arrange the received help symbols according to Figure. 6.7. Repeat the following steps from Layer 1 *to Layer* m*:*

Step 1: For a certain grid, if errors are detected in the symbols sent by node i in previous layers of the same column, replace the symbol sent from node i by an erasure $\bigotimes$.

Step 2: Parallel regenerate all the symbols related to ρ data blocks using the algorithm similar to Algorithm 6.1 with only one difference: parallel decode all the ρ MDS codes in **Step 1** of Algorithm 6.1.

6.4.5 Reconstruction

When DC needs to reconstruct the original file, it will send reconstruction requests to n storage nodes. Upon receiving the request, node i will send out the symbol vector $\mathbf{c}_i$. Suppose $\mathbf{c}'_i = \mathbf{c}_i + \mathbf{e}_i$ is the response from the i^{th} storage node. If $\mathbf{c}_i$ has been modified by the malicious node i, we have $\mathbf{e}_i \in (GF_q)^{\alpha\theta} \backslash \{\mathbf{0}\}$. The strategy of combining dependent decoding and parallel decoding for reconstruction is similar to that for regeneration. *DC* will place the received symbols into a 2-dimension lattice with size $m \times \rho$. The only difference is that in a grid of the lattice there are n symbol vectors $\mathbf{ch}'_{\mathbf{j},0}, \ldots, \mathbf{ch}'_{\mathbf{j},n-1}$ corresponding to data block j, received from n storage nodes. DC will perform the reconstruction using Algorithm 6.5.

Algorithm 6.5 DC Reconstructs the Original File for the m-layer Rate-matched MSR Code

Arrange the received symbols similar to Figure. 6.7. Here we place received codeword matrix CH'_j into grid j instead of help symbols received from n-1 help nodes. Repeat the following steps from Layer 1 *to Layer* m*:*

Step 1: For a certain grid, if errors are detected in the symbols sent by node i in previous layers of the same column, replace symbols sent from node i by erasures $\bigotimes$.

Step 2: Parallel reconstruct all the symbols of the ρ data blocks using the algorithm similar to Section 6.2.1.3 with only one difference: parallel decode all the MDS codes in Section 6.2.1.3.

6.4.5.1 Optimized Parameters

From Section 6.4.1 we know that for regeneration of an optimal m-layer rate-matched MSR code, the parameter d's of all the layers are the same, which implies the parameter α's of all layers are also the same. Since the optimization of regeneration is derived based on the decoding of $(n-1, d, n-d)$ MDS codes and in reconstruction we have to decode $(n-1, \alpha, n-\alpha)$ MDS codes, if the parameter α's of all the layers are the same, we can achieve the same optimization results for reconstruction.

CHAPTER 7

CONCLUSIONS

In this dissertation, we study the secure problems in network coding and distributed storage. We propose and analysis schemes for combating pollution attacks in network coding and combating malicious attacks in distributed storage.

For combating pollution attacks in fixed network coding, we analyze the relationship between the error control coding and the network coding in unicast case and prove that the two codes are essentially correlated. Furthermore, we extend this correlation to multicast case. This research provides a methodology to design efficient network coding scheme based on the communication channel and error control coding schemes to combat the communication errors and node compromising attacks.

At the same time, we analyze the relationship between the cascaded error control codes and the network code in unicast case and prove that the two codes are essentially correlated. Then we extend this correlation to multicast case. This research provides a new methodology that can combat the communication errors and node compromising attacks by designing efficient network coding scheme based on cascaded error control codes and fully utilizing the inner structure of network codes.

For combating pollution attacks in random network coding, our purpose is to guarantee a minimum throughput even for heavily polluted network environments. We first introduced an error detection and error correction (EDEC) scheme. By utilizing the information available in the corrupted packets, the network throughput can be maintained with only a slight increase of the computational overhead when moderate pollution attacks present. To deal with network environment with heavy pollution, we introduced LEDEC scheme that enables channel information be exploited and belief propagation algorithm (BPA) be used for the packet symbol recovery. This scheme can guarantee the throughput under the heavy

pollution. We formulated the throughput of the LEDEC scheme through both theoretical analysis and comprehensive evaluation. Our extensive simulation results derived in ns-2 platform show that the theoretical results are achievable in practical environments.

For combating malicious attacks in distributed storage, we develop a Hermitian code based minimum storage regeneration (H-MSR) code and a Hermitian code based minimum bandwidth regeneration (H-MBR) code for distributed storage. Due to the structure of Hermitian code, our proposed codes can significantly improve the performance of the regenerating code under malicious attacks. In particular, these codes can deal with errors beyond the maximum distance separable (MDS) code. Our theoretical analyses demonstrate that the H-MSR/H-MBR codes have lower complexity than the Reed-Solomon based minimum storage regeneration (RS-MSR) code and the Reed-Solomon based minimum bandwidth regeneration (RS-MBR) code in both regeneration and reconstruction.

We also develop two rate-matched minimum storage regeneration (MSR) codes for malicious nodes detection and correction in hostile networks: 2-layer rate-matched MSR code and m-layer rate-matched MSR code. We propose the encoding, regeneration and reconstruction algorithms for both codes. For the 2-layer rate-matched code, we optimize the parameters for the data regeneration, considering the trade-off between the malicious nodes detection probability and the storage efficiency. Theoretical analysis shows that the code can successfully detect and correct malicious nodes using the optimized parameters. Our analysis also shows that the code has higher storage efficiency compared to the RS-MSR code (70% higher for the detection probability 0.999999). Then we extend the 2-layer code to m-layer code and optimize the overall error correction efficiency by matching the code rate of each layer's MSR code. Theoretical analysis shows that the optimized parameter could also achieve the maximum storage capacity under the same constraint. Furthermore, analysis shows that compared to the RS-MSR code, our code can improve the error correction efficiency more than 50%.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] R. Ahlswede, N. Cai, S.-Y.Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, pp. 1205–1216, July 2000.

[2] S.-Y.Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, 2003.

[3] R. Koetter and M. Medard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, 2003.

[4] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks using randomized network coding," in *International Symposium on Information Theory (ISIT) 2004*, July 2004, p. 144.

[5] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE INFOCOM 2006*, Apr. 2006, pp. 1–13.

[6] N. Cai and R. W. Yeung, "Network error correctionvpart ii: Lower bounds," *Communications in Information and Systems*, vol. 6, pp. 37–54, 2006.

[7] R. W. Yeung and N. Cai, "Network error correction, part i: Basic concepts and upper bounds," *Communications in Information and Systems*, vol. 6, pp. 19–36, 2006.

[8] N. Cai, "Network localized error correction: For non-coherent coding," in *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2011, pp. 1146–1150.

[9] N. Cai and R. W. Yeung, "Network coding and error correction," in *Proc. of IEEE Information Theory Workshop (ITW 2002)*, 2002, pp. 119–122.

[10] R. Matsumoto, "Construction algorithm for network error-correcting codes attaining the singleton bound," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. Volume E90-A, pp. 1729–1735, 2007.

[11] S. Yang, C.-K. Ngai, and R. Yeung, "Construction of linear network codes that achieve a refined singleton bound," in *ISIT 2007. IEEE International Symposium on Information Theory, 2007*, 2007, pp. 1576–1580.

[12] Z. Zhang, "Linear network error correction codes in packet networks," *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 209–218, 2008.

[13] H. Balli, X. Yan, and Z. Zhang, "On randomized linear network codes and their error correction capabilities," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3148–3160, 2009.

[14] Z. Zhang, X. Yan, and H. Balli, "Some key problems in network error correction coding theory," in *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, 2007, pp. 1–5.

[15] Z. Zhang, "Some recent progresses in network error correction coding theory," in *NetCod 2008. Fourth Workshop on Network Coding, Theory and Applications, 2008*, 2008, pp. 1–5.

[16] ——, "Network error correction coding in packetized networks," in *Information Theory Workshop, 2006. ITW '06 Chengdu. IEEE*, 2006, pp. 433–437.

[17] H. Balli and Z. Zhang, "On the limiting behavior of random linear network codes," in *NetCod '09. Workshop on Network Coding, Theory, and Applications, 2009*, 2009, pp. 1–5.

[18] N. Cai and R. W. Yeung, "The singleton bound for network error-correcting codes," in *2006 4th International Symposium on Turbo Codes Related Topics; 6th International ITG-Conference on Source and Channel Coding (TURBOCODING)*, 2006, pp. 1–6.

[19] S. Yang, R. Yeung, and C.-K. Ngai, "Refined coding bounds and code constructions for coherent network error correction," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1409–1424, 2011.

[20] M. Siavoshani, C. Fragouli, and S. Diggavi, "On locating byzantine attackers," in *NetCod 2008. Fourth Workshop on Network Coding, Theory and Applications, 2008*, 2008, pp. 1–6.

[21] H. Bahramgiri and F. Lahouti, "Block network error control codes and syndrome-based maximum likelihood decoding," in *ISIT 2008. IEEE International Symposium on Information Theory, 2008.*, 2008, pp. 807–811.

[22] R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3579–3591, 2008.

[23] D. Silva, F. Kschischang, and R. Koetter, "A rank-metric approach to error control in random network coding," in *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, 2007, pp. 1–5.

[24] D. Silva and F. Kschischang, "Adversarial error correction for network coding: Models and metrics," in *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, 2008, pp. 1246–1253.

[25] M. Gadouleau and Z. Yan, "Decoder error probability of bounded distance decoders for constant-dimension codes," in *ISIT 2009. IEEE International Symposium on Information Theory, 2009*, 2009, pp. 2226–2230.

[26] D. Silva and F. Kschischang, "Universal secure network coding via rank-metric codes," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 1124–1135, 2011.

[27] N. Chen, Z. Yan, M. Gadouleau, Y. Wang, and B. Suter, "Rank metric decoder architectures for random linear network coding with error control," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 296–309, 2012.

[28] S. Jaggi, M. Langberg, T. Ho, and M. Effros, "Correction of adversarial errors in networks," in *Proc. of International Symposium on Information Theory (ISIT 2005)*, 2005, pp. 1455–1459.

[29] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *IEEE Transactions on Information Theory*, vol. 54, no. 6, pp. 2596–2603, 2008.

[30] S. Kim, T. Ho, M. Effros, and A. Avestimehr, "Network error correction with unequal link capacities," *IEEE Transactions on Information Theory*, vol. 57, no. 2, pp. 1144–1164, 2011.

[31] J. Kurihara, T. Uyematsu, and R. Matsumoto, "New parameters of linear codes expressing security performance of universal secure network coding," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 533–540.

[32] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing xor network coding against pollution attacks," in *IEEE INFOCOM 2009*, Apr. 2009, pp. 406–414.

[33] M. Krohn, M. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," in *IEEE Symposium on Security and Privacy 2004*, May 2004, pp. 226–240.

[34] D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *2006 40th Annual Conference on Information Sciences and Systems*, 2006, pp. 857–863.

[35] M. Kim, M. Medard, M. Medard, and J. Barros, "An algebraic watchdog for wireless network coding," in *ISIT 2009. IEEE International Symposium on Information Theory, 2009*, 2009, pp. 1159–1163.

[36] F. Oggier and H. Fathi, "An authentication code against pollution attacks in network coding," *IEEE/ACM Transactions on Networking*, vol. 19, no. 6, pp. 1587–1596, 2011.

[37] D. Boneh, D. Freeman, J. Katz, and B. Waters, "Signing a linear subspace: Signature schemes for network coding," in *12th International Conference on Practice and Theory in Public Key Cryptography*, 2009, pp. 68–87.

[38] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008, pp. –.

[39] F. Zhao, T. Kalker, M. Medard, and K. Han, "Signatures for content distribution with network coding," in *ISIT 2007. IEEE International Symposium on Information Theory, 2007*, 2007, pp. 556–560.

[40] S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman, "Preventing pollution attacks in multi-source network coding," in *13th International Conference on Practice and Theory in Public Key Cryptography*, 2010, pp. 161–176.

[41] A. Le and A. Markopoulou, "Cooperative defense against pollution attacks in network coding using spacemac," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 2, pp. 442–449, 2012.

[42] Y. Xu and K. Sakurai, "Cooperatively securing network coding against pollution attacks with incentive mechanism," in *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, no. 52, 2012.

[43] J. Dong, R. Curtmola, C. Nita-Rotaru, and D. Yau, "Pollution attacks and defenses in wireless inter-flow network coding systems," in *Wireless Network Coding Conference (WiNC), 2010 IEEE*, 2010, pp. 1–6.

[44] E. Kehdi and B. Li, "Null keys: Limiting malicious attacks via null space properties of network coding," in *IEEE INFOCOM 2009*, Apr. 2009, pp. 1224–1232.

[45] M. Yang and J. An, "Combined fountain code with network coding for error-tolerant transmission network," in *WiCom '09. 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009*, 2009, pp. 1–4.

[46] Z. Liu and S. Jin, "An interaction between network coding and end-host coding," in *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, 2011, pp. 885–890.

[47] H. Yao, T. Ho, and C. Nita-Rotaru, "Key agreement for wireless networks in the presence of active adversaries," in *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, 2011, pp. 792–796.

[48] M. Kim, M. Medard, and J. Barros, "Counteracting byzantine adversaries with network coding: An overhead analysis," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, 2008, pp. 1–7.

[49] J. Zhang, K. Letaief, and P. Fan, "A distributed product coding approach for robust network coding," in *ICC '08. IEEE International Conference on Communications, 2008*, 2008, pp. 176–180.

[50] Q. Wang, S. Jaggi, and S.-Y. Li, "Binary error correcting network codes," in *Information Theory Workshop (ITW), 2011 IEEE*, 2011, pp. 498–502.

[51] M. Najeem and C. Siva Ram Murthy, "On enhancing the random linear network coding," in *17th IEEE International Conference on Networks (ICON), 2011*, 2011, pp. 246–251.

[52] M. Gadouleau and A. Goupil, "Binary codes for packet error and packet loss correction in store and forward," in *2010 International ITG Conference on Source and Channel Coding (SCC)*, 2010, pp. 1–6.

[53] S. Vyetrenko, A. Khosla, and T. Ho, "On combining information-theoretic and cryptographic approaches to network coding security against the pollution attack," in *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, 2009, pp. 788–792.

[54] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, vol. 5, pp. 40 – 49, 2001.

[55] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *roc. Symp. Netw. Syst. Design Implementation*, 2004, pp. 337–350.

[56] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, pp. 4539 – 4551, 2010.

[57] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, pp. 5227–5239, 2011.

[58] D. Cullina, A. G. Dimakis, and T. Ho, "Searching for minimum storage regenerating codes," *Available:arXiv:0910.2245*, 2009.

[59] N. Shah, K. Rashmi, P. Kumar, and K. Ramchandran, "Explicit codes minimizing repair bandwidth for distributed storage," in *Information Theory Workshop (ITW), 2010 IEEE*, 2010, pp. 1–5.

[60] C. Suh and K. Ramchandran, "Exact-repair mds codes for distributed storage using interference alignment," in *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2010, pp. 161–165.

[61] Y. Wu, "A construction of systematic mds codes with minimum repair bandwidth," *IEEE Transactions on Information Theory*, vol. 57, no. 6, pp. 3738–3741, 2011.

[62] D. Papailiopoulos, J. Luo, A. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *INFOCOM, 2012 Proceedings IEEE*, 2012, pp. 2801–2805.

[63] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 1510–1517.

[64] I. Tamo, Z. Wang, and J. Bruck, "Mds array codes with optimal rebuilding," in *2011 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2011, pp. 1240–1244.

[65] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, "Optimal repair of mds codes in distributed storage via subspace interference alignment," *Available:arXiv:1106.1250*, 2011.

[66] D. Papailiopoulos, A. Dimakis, and V. Cadambe, "Repair optimal erasure codes through hadamard designs," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3021–3037, 2013.

[67] N. Shah, K. V. Rashmi, and P. Kumar, "A flexible class of regenerating codes for distributed storage," in *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2010, pp. 1943–1947.

[68] K. Shum and Y. Hu, "Existence of minimum-repair-bandwidth cooperative regenerating codes," in *2011 International Symposium on Network Coding (NetCod)*, 2011, pp. 1–6.

[69] A. Wang and Z. Zhang, "Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage," in *INFOCOM, 2013 Proceedings IEEE*, 2013, pp. 400–404.

[70] H. Hou, K. W. Shum, M. Chen, and H. Li, "Basic regenerating code: Binary addition and shift for exact repair," in *2013 IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2013, pp. 1621–1625.

[71] Y.-L. Chen, G.-M. Li, C.-T. Tsai, S.-M. Yuan, and H.-T. Chiao, "Regenerating code based p2p storage scheme with caching," in *ICCIT '09. Fourth International Conference on Computer Sciences and Convergence Information Technology, 2009*, 2009, pp. 927–932.

[72] Y. Wu, A. G. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *45th Annu. Allerton Conf. Control, Computing, and Communication*, 2007.

[73] A. Duminuco and E. Biersack, "A practical study of regenerating codes for peer-to-peer backup systems," in *ICDCS '09. 29th IEEE International Conference on Distributed Computing Systems, 2009*, June 2009, pp. 376 – 384.

[74] K. Shum, "Cooperative regenerating codes for distributed storage systems," in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–5.

[75] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *IEEE International Symposium on Information Theory, 2009. ISIT 2009.*, 2009, pp. 2276–2280.

[76] N. Shah, K. Rashmi, P. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Transactions on Information Theory*, vol. 58, pp. 2134 – 2158, 2012.

[77] F. Oggier and A. Datta, "Byzantine fault tolerance of regenerating codes," in *2011 IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2011, pp. 112–121.

[78] S. Pawar, S. El Rouayheb, and K. Ramchandran, "Securing dynamic distributed storage systems against eavesdropping and adversarial attacks," *IEEE Transactions on Information Theory*, vol. 57, pp. 6734 – 6753, 2011.

[79] Y. Han, R. Zheng, and W. H. Mow, "Exact regenerating codes for byzantine fault tolerance in distributed storage," in *Proceedings IEEE INFOCOM*, 2012, pp. 2498 – 2506.

[80] H. Chen and P. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage," in *2012 IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*, 2012, pp. 51–60.

[81] C. Cachin and S. Tessaro, "Optimal resilience for erasure-coded byzantine distributed storage," in *DSN 2006. International Conference on Dependable Systems and Networks, 2006*, 2006, pp. 115–124.

[82] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie, "Lazy verification in fault-tolerant distributed storage systems," in *SRDS 2005. 24th IEEE Symposium on Reliable Distributed Systems, 2005*, 2005, pp. 179–190.

[83] K. Rashmi, N. Shah, K. Ramchandran, and P. Kumar, "Regenerating codes for errors and erasures in distributed storage," in *International Symposium on Information Theory (ISIT) 2012*, 2012, pp. 1202–1206.

[84] L. Nutman and M. Langberg, "Adversarial models and resilient schemes for network coding," in *ISIT 2008. IEEE International Symposium on Information Theory, 2008.*, July 2008, pp. 171–175.

[85] O. Kosut and L.-W. Kao, "On generalized active attacks by causal adversaries in networks," in *Information Theory Workshop (ITW), 2014 IEEE*, Nov 2014, pp. 247–251.

[86] P. Wang and R. Safavi-Naini, "An efficient code for adversarial wiretap channel," in *Information Theory Workshop (ITW), 2014 IEEE*, Nov 2014, pp. 40–44.

[87] J. Ren, "On the structure of hermitian codes and decoding for burst errors," *IEEE Transactions on Information Theory*, vol. 50, pp. 2850– 2854, 2004.

[88] J. Li, T. Li, and J. Ren, "Beyond the mds bound in distributed cloud storage," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 307–315.

[89] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Prentice Hall, June 2004.

[90] L. Chen, R. Carrasco, and M. Johnston, "Soft-decision list decoding of hermitian codes," *IEEE Transactions on Communications*, vol. 57, no. 8, pp. 2169–2176, 2009.

[91] R. Blahut, "On codes containing hermitian codes," in *1995 IEEE International Symposium on Information Theory*, 1995, pp. 101–.

[92] J. Li, C. Yang, D. Tang, T. Li, and J. Ren, "Characterization of linear network coding for pollution detection," in *Global Communications Conference (GLOBECOM), 2012 IEEE*, 2012, pp. 1066–1071.

[93] T. Dierks and C. Allen, "The TLS protocol, version 1.0," RFC-2246, January 1999.

[94] W. NS2, http://nsnam.isi.edu/nsnam/index.php/Main$_P$age, 2010.

[95] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. IT-8, no. 1, pp. 21–28, January 1962.

[96] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 533–547,623–656, July and October 1948.

[97] C. Jones, A. Matache, T. Tian, J. Villasenor, and R. Wesel, "The universality of ldpc codes on wireless channels," in *Military Communications Conference, 2003. MILCOM '03. 2003 IEEE*, vol. 1, 2003, pp. 440–445 Vol.1.

[98] M. Franceschini, G. Ferrari, and R. Raheli, "Does the performance of ldpc codes depend on the channel?" *IEEE Transactions on Communications*, vol. 54, no. 12, pp. 2129–2132, 2006.

[99] D. Spielman, "Linear-time encodable and decodable error-correcting codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1723–1731, 1996.

[100] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in *STOC '97 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, 1997, pp. 150–159.

[101] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585–598, 2001.

[102] X.-Y. Hu, E. Eleftheriou, D.-M. Arnold, and A. Dholakia, "Efficient implementations of the sum-product algorithm for decoding ldpc codes," in *Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE*, vol. 2, 2001, pp. 1036–1036E vol.2.

[103] H. Pishro-Nik and F. Fekri, "On decoding of low-density parity-check codes over the binary erasure channel," *IEEE Transactions on Information Theory*, vol. 50, no. 3, pp. 439–454, 2004.

[104] L. Bazzi, T. Richardson, and R. Urbanke, "Exact thresholds and optimal codes for the binary-symmetric channel and gallager's decoding algorithm a," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2010–2021, 2004.

[105] G. Liva, E. Paolini, B. Matuz, and M. Chiani, "A decoding algorithm for ldpc codes over erasure channels with sporadic errors," in *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 458–465.

[106] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theoryn*, vol. 47, no. 2, pp. 619–637, 2001.

[107] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Analysis of random processes via and-or tree evaluation," in *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1998, pp. 364–373.

[108] M. Luby, M. Mitzenmacher, A. Shokrollah, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in *STOC '98 Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 249 – 258.

[109] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 599–618, 2001.

[110] L. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[111] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D.Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, pp. 569–584, 2001.

[112] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[113] D. Dabiri and I. Blake, "Fast parallel algorithms for decoding reed-solomon codes based on remainder polynomials," *IEEE Transactions on Information Theory*, vol. 41, no. 4, pp. 873–885, Jul 1995.