Overhead-free In-place Recovery and Repair Schemes of XOR-based Regenerating Codes

Ximing Fu*, Zhiqing Xiao†, and Shenghao Yang‡
*Department of Computer Science and Technology, Tsinghua University
†Department of Electronic Engineering, Tsinghua University
‡Institute of Network Coding, The Chinese University of Hong Kong

***Abstract*—In this paper, refined recovery and repair schemes are proposed for a storage system using the XOR-based MBR regenerating storage code proposed by Hou et al. Our schemes have zero transmission overhead for both recovery and repair, i.e., the total number of transmitted bits for repair/recovery is exactly equal to the total number of bits repaired/recovered. Further, our schemes use mainly XOR operations and have lower complexity than that of the previous schemes. Moreover, our schemes require only a small amount of auxiliary space, which qualifies our schemes as in-place.**

I. Introduction

Regenerating codes were proposed for distributed storage systems in [1]–[3]. In a typical setting, a file of M bits is encoded and stored at n storage nodes. The file can be recovered by accessing any k storage nodes. When a storage node fails, a new node can be regenerated by accessing any d surviving nodes such that the new set of n storage nodes preserves the above recovery and regenerating properties. A regenerating code with the above parameters is also called an $[n, k, d]$ code. Suppose that each storage node stores α bits and the regenerating bandwith is γ bits, i.e., the total number of bits communicated from the d nodes during regenerating. Dimakis et al. [3] characterized a fundamental tradeoff between the storage per node and the regenerating bandwidth.

Two extremal points in the optimal storage-bandwith tradeoff curve are of particular interest, i.e., the minimum-storage regenerating (MSR) point and minimum-bandwidth regenerating (MBR) point. The regenerating codes attaining the MSR point store $\alpha = \frac{M}{k}$ bits in each node. Since each k nodes can be used to recover the original file, such regenerating codes have zero recovery overhead, i.e., the total recovery bandwidth is equal to the file size. The regenerating codes attaining the MBR point have the minimum repair bandwidth $\gamma = \frac{M}{k}\frac{2d}{2d-k+1}$ and $\alpha = \gamma$. Accessing $k\alpha = M\frac{2d}{2d-k+1}$ bits from any k nodes is sufficient to recover the original file. But $k\alpha$ is strictly larger than M when $k > 1$ so that an MBR code may not have zero recovery overhead. We focus on MBR codes in this paper.

Rashmi, Shah and Kumar [4] provided product-matrix constructions of MBR codes for all valid values of $[n, k, d]$ and MSR codes for $d \geq 2k - 2$. The product-matrix MBR codes, however, require matrix operations over finite fields for encoding and recovery, which leads to high complexity for practical systems. To resolve this complexity issue, Hou et al. [5]–[7] proposed BASIC codes using an exclusive-or (XOR) version of the product-matrix constructions. For BASIC codes, encoding and recovery mainly use binary XOR and shift operations, so the computational complexities are significantly reduced.

In a BASIC MBR code, a file of M bits are divided into $B = kd - \binom{k}{2}$ sequences, each of which consists of $L = M/B$ bits. After encoding, each storage node stores d encoded packets. In the recovery algorithm introduced in [5], kd packets are retrieved from k storage nodes and the B sequences are recovered by solving d linear systems of dimension k. Due to the shift operations, the packets encoded may be of different length but all have at least L bits. This *packet overhead* (the number of bits in a packet minus L) would affect the recovery bandwidth. The recovery bandwidth, the extra decoding storage and the computational complexity in the recovery algorithm of [5] are given in Table I. In the worst case, about $2M$ bits are transmitted to recover the M bits.

In this paper, we propose a more efficient recovery scheme for BASIC codes, which works for all valid values of $[n, k, d]$. Our recovery scheme has two stages: the retrieving stage and the decoding stage. In a BASIC MBR code, each node stores more than M/k bits. In the retrieving stage of our scheme, exactly M bits are retrieved from any k storage nodes. Therefore, our scheme achieves the optimal recovery bandwidth exactly. In other words, our scheme implies that it is possible to achieve zero recovery overhead for MBR codes. In the decoding stage of our scheme, the M bits retrieved in the first stage are used to recover the original file. Our algorithm is similar to the ZigZag decoding of Sung and Gong [8] designed for a storage code based on XOR and shift operations. We optimize their algorithm for BASIC MBR codes to gain lower computational and storage complexities. Specifically, the number of XOR operations used in our recovery scheme is $O\left(dk^2L\right)$. After retrieving the data from the storage nodes, our decoding algorithm overwrites the data during execution, and only consumes $O\left(k \log L\right)$ extra storage space for auxiliary variables. After the algorithm executing, the M bits in the memory storing the retrieved data become exactly the desired file. Therefore, our decoding algorithm is *in-place*.

The packet overhead also affects the repair bandwidth. For the repair scheme of BASIC codes in [4], [5], the total number

This work was supported in part by the National Basic Research Program of China (973 Program) under Grant 2013CB834205 and the National Natural Science Foundation of China (NSFC) under Grant 61133013 and 61471215. This work was partially funded by a grant from the University Grants Committee of the Hong Kong Special Administrative Region (Project No. AoE/E-02/08).

TABLE I. COMPARISON BETWEEN OUR RESULT AND THE PREVIOUS RESULT.

Recovery Scheme	Recovery Bandwidth	Extra Decoding Storage	Decoding Time Complexity
Recovery in [5]	$M\frac{2d}{2d-k+1}+O(nkd^2)$	$O(BL)$	$O(d^2k^3nL)$
Our scheme	M	$O(k\log L)$	$O(dk^2L)$

of bits transmitted for repairing a storage node is larger than the number of bits stored in the node. In this paper, we propose an in-place repair scheme by applying an algorithm developed in [9]. Our repair scheme is overhead free: the total number of bits transmitted for repairing a storage node is exactly equal to the number of bits stored in the node.

The rest of the paper is organized as follows: Section II describes the XOR-based MBR codes. Section III introduces a procedure that is used for both recovery and repair. Section IV and Section V present our repair and recovery schemes respectively.

II. XOR-BASED MBR CODES

In order to give a precise description of our recovery/repair scheme, we introduce the encoding of the XOR-based $[n,k,d]$ MBR codes in [5] with a minor generalization. Without loss of generality, we assume

$$2\le k\le d\le n-1.$$

The encoding of BASIC codes disperses a file into n storage nodes. First, the file is divided into several sequences, and these sequences are rearranged as a message matrix. After that, the data packets in each node is obtained by multiplying an encoding matrix with the message matrix. The details of the encoding is described as follows.

A file of M bits is divided into

$$B=(k+1)k/2+k(d-k)=kd-\binom{k}{2}$$

sequences, each of which consists of $L=M/B$ bits. These B sequences, denoted as $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_B$, are used to form the *message matrix* $\mathbf{M}=(\mathbf{m}_{i,j})_{1\le i,j\le d}$, which is a $d\times d$ matrix with L-bit sequences as components. The message matrix $\mathbf{M}$ is of the form

$$\mathbf{M}=\begin{pmatrix}\mathbf{S} & \mathbf{T}\\ \mathbf{T}^\top & \mathbf{O}\end{pmatrix}, \tag{1}$$

where $\mathbf{S}=(\mathbf{s}_{i,j})$ is a $k\times k$ symmetric matrix, $\mathbf{T}=(\mathbf{t}_{i,j})$ is a $k\times(d-k)$ matrix, and $\mathbf{O}$ is a $(d-k)\times(d-k)$ zero matrix. $\mathbf{T}^\top$ is the transpose of $\mathbf{T}$.

The message matrix $\mathbf{M}$ can be formed by any one-to-one mapping from $\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_B\}$ to the $(k+1)k/2+k(d-k)=B$ free components in $\mathbf{M}$. For example, when $k=3$ and $d=4$, we have $B=9$, and the message matrix can be

$$\mathbf{M}=\begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_4 & \mathbf{x}_7\\ \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_5 & \mathbf{x}_8\\ \mathbf{x}_4 & \mathbf{x}_5 & \mathbf{x}_6 & \mathbf{x}_9\\ \mathbf{x}_7 & \mathbf{x}_8 & \mathbf{x}_9 & \mathbf{0}\end{pmatrix}. \tag{2}$$

The *encoding matrix* $\mathbf{\Psi}$ is an $n\times d$ matrix where the i-th row and j-th column component is $z^{t_{i,j}}$, a polynomial in indeterminate z. Here $t_{i,j}$ is an integer. In [5], $t_{i,j}=(i-1)(j-1)$ such that $\mathbf{\Psi}$ is a Vandermonde matrix. For example, the Vandermonde matrix for $n=6$ and $d=4$ is

$$\mathbf{\Psi}=\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & z & z^2 & z^3\\ 1 & z^2 & z^4 & z^6\\ 1 & z^3 & z^6 & z^9\\ 1 & z^4 & z^8 & z^{12}\\ 1 & z^5 & z^{10} & z^{15}\end{pmatrix}. \tag{3}$$

In general, we may assume that $\{t_{i,j}\}$ satisfies the *refined increasing difference property*, i.e., for any i, i', j, and j' such that $i<i'$ and $j<j'$,

$$0\le t_{i,j'}-t_{i,j}<t_{i',j'}-t_{i',j}, \tag{4}$$

where the equality holds only when $i=1$. (Note that in the *increasing difference property* introduced in [8], the first inequality in (4) is strict.) The encoding of BASIC codes generates d packets for each of the n storage nodes using the message matrix $\mathbf{M}$ and the encoding matrix $\mathbf{\Psi}$. For a binary sequence $\mathbf{m}$ and integer t, define the shift operation

$$z^t\mathbf{m}=(\mathbf{0}_t,\mathbf{m})$$

where $\mathbf{0}_t$ is a sequence of t zeros attached before $\mathbf{m}$. Define the formal multiplication of $\mathbf{\Psi}$ and $\mathbf{M}$ as

$$\mathbf{Y}=(\mathbf{y}_{i,j})_{1\le i\le n,1\le j\le d}=\mathbf{\Psi}\mathbf{M}, \tag{5}$$

where

$$\mathbf{y}_{i,j}=\sum_{u=1}^{d}z^{t_{i,u}}\mathbf{m}_{u,j}.$$

Note that in the above summation, the addition is bit-wise XOR. If two sequences are not of the same length, zeros are appended after the shorter sequence before the addition operation. It can be checked that $\mathbf{y}_{i,j}$ has $L+t_{i,d}$ bits.

Using the above notations, the d packets stored in the i-th node are $\mathbf{y}_{i,j}$, $j=1,\ldots,d$. In the aforementioned example with the message matrix in (2) and the encoding matrix in (3), the coded packets stored in node i ($1\le i\le 6$) are

$$\begin{aligned}\mathbf{y}_{i,1}&=\mathbf{x}_1+z^{i-1}\mathbf{x}_2+z^{2(i-1)}\mathbf{x}_4+z^{3(i-1)}\mathbf{x}_7,\\ \mathbf{y}_{i,2}&=\mathbf{x}_2+z^{i-1}\mathbf{x}_3+z^{2(i-1)}\mathbf{x}_5+z^{3(i-1)}\mathbf{x}_8,\\ \mathbf{y}_{i,3}&=\mathbf{x}_4+z^{i-1}\mathbf{x}_5+z^{2(i-1)}\mathbf{x}_6+z^{3(i-1)}\mathbf{x}_9,\\ \mathbf{y}_{i,4}&=\mathbf{x}_7+z^{i-1}\mathbf{x}_8+z^{2(i-1)}\mathbf{x}_9.\end{aligned} \tag{6}$$

III. A BASIC PROCEDURE

Before describing our recovery and repair schemes, we give a procedure that will be used repeatedly in our schemes.

Consider a file of kL bits represented by $\mathbf{m}_i$, $i=1,\ldots,k$, where $\mathbf{m}_i$ is an L-bit sequence. Let $\mathbf{\Phi}=(z^{t_{i,j}})$ be an $n\times k$ matrix where z is an indeterminate and $t_{i,j}$, $1\le i\le n$, $1\le j\le k$, are integers satisfying the refined increasing difference property. Let

$$[\mathbf{y}_1,\ldots,\mathbf{y}_n]^\top=\mathbf{\Phi}[\mathbf{m}_1,\ldots,\mathbf{m}_k]^\top,$$

where the matrix multiplication is defined as in (5). Note that due to the shift operation, $\mathbf{y}_i$ has $L+t_{i,k}$ bits.

Algorithm 1 In-place Decoding for MDS Codes

Input: coded packets stored in $\hat{\mathbf{m}}_u$, and corresponding node indices i_u ($1 \le u \le k$), which satisfy $i_1 > i_2 > \cdots > i_k$.
Output: recovered messages, stored in $\hat{\mathbf{m}}_u$ ($1 \le u \le k$).
Step 1: *(Initialize auxiliary variables)*
Initialize a vector $(l_1, l_2, \ldots, l_k)$ to a zero vector: $l_u \leftarrow 0$ ($1 \le u \le k$). (This vector will store the number of bits that have been recovered successfully.)
Step 2: *(In-place decoding)*
while $l_k < L$ (Iterate until the message is fully recovered.) **do**
  for $u \leftarrow 1 : k$ **do**
    if ($l_u < L$) and ($u = 1$ or $l_{u-1} > t_{i_u,u} - t_{i_u,u-1}$) **then**
      $l_u \leftarrow l_u + 1$; (Freeze one more bit in $\hat{\mathbf{m}}_u$.)
      for $v \leftarrow 1 : k$ **do**
        if $v \neq u$ and $0 < l_u + t_{i_v,u} - t_{i_v,v} \le L$ **then**
          $\hat{\mathbf{m}}_v[l_u + t_{i_v,u} - t_{i_v,v}] \leftarrow \hat{\mathbf{m}}_v[l_u + t_{i_v,u} - t_{i_v,v}] \oplus \hat{\mathbf{m}}_u[l_u]$; (Eliminate the superposed bits.)
        end if
      end for
    end if
  end for
end while

Suppose that we have a storage system of n nodes, where node i stores $\mathbf{y}_i$.

The following scheme, proposed in [9], can recover the file from any k out of the n storage nodes, where exactly kL bits are retrieved from these k nodes.

Particularly, a data collector accesses k nodes indexed by $i_u, 1 \le u \le k$, where $i_1 > i_2 > \cdots > i_k$ without loss of generality. Let $[i, j]$ be the set of integers $\{i, i+1, \ldots, j\}$ when $i \le j$. For a sequence of bits $\mathbf{m}$, denote by $\mathbf{m}[i]$ the i-th bit of the sequence and denote by $\mathbf{m}[i, j]$ the sub-sequence of $\mathbf{m}$ in the range $[i, j]$. Node i_u ($1 \le u \le k$) transmits to the data collector the sub-sequence $\mathbf{y}_{i_u}[t_{i_u,u} + 1, t_{i_u,u} + L]$, denoted by $\hat{\mathbf{m}}_u$, i.e.,

$$\hat{\mathbf{m}}_u = \mathbf{m}_u + \sum_{j=1}^{u-1} \left(\mathbf{m}_j\left[t_{i_u,u} - t_{i_u,j} + 1, L\right], \mathbf{0}_{t_{i_u,u} - t_{i_u,j}}\right) + \sum_{j=u+1}^{k} \left(\mathbf{0}_{t_{i_u,j} - t_{i_u,u}}, \mathbf{m}_j\left[1, L + t_{i_u,u} - t_{i_u,j}\right]\right).$$

So the packet node i_u transmitted to the data collector involves all bits of sequence $\mathbf{m}_u$. All superposed bits from other sequences can be eliminated by executing Algorithm 1 on $\hat{\mathbf{m}}_i$, resulting in $\hat{\mathbf{m}}_i = \mathbf{m}_i$, $i = 1, 2, \ldots, k$ according to [9]. The decoding algorithm is in-place.

IV. Our Repair Scheme

For an $m \times n$ matrix $\mathbf{A}$, $I \subset [1, m]$ and $J \subset [1, n]$ we denote by $\mathbf{A}_I$ (resp. $\mathbf{A}^J$) the submatrices of $\mathbf{A}$ formed by all the columns (resp. rows) with indices in I (resp. J). When $I = i$ (resp. $J = j$), we also write $\mathbf{A}_i$ (resp. $\mathbf{A}^j$) for convenience.

Let us first consider repair. For the storage code generated above, a failed node can be repaired using any d out of the remaining $n - 1$ nodes.

Suppose that node i fails, and we want to generate a new storage node that stores d packets $\mathbf{Y}^i = \mathbf{\Psi}^i\mathbf{M}$, which is the same as the d packets stored in node i. Note that each packet of node i has $L + t_{i,d}$ bits. Fix d surviving nodes with indices i_j, $1 \le j \le d$, assuming $i_1 > i_2 > \cdots > i_d$. In the repair scheme of [4], [5], node i_j transmits

$$\mathbf{r}_{i_j} = \mathbf{\Psi}^{i_j}\mathbf{M}(\mathbf{\Psi}^i)^\top = \mathbf{\Psi}^{i_j}(\mathbf{Y}^i)^\top$$

to the new storage node. The length of $\mathbf{r}_{i_j}$ is $L + t_{i,d} + t_{i_j,d}$ bits. Thus the *repair bandwidth overhead*, defined as the number of bits transmitted minus the number of bits to repair, is $\sum_{j=1}^{d} t_{i_j,d}$.

A better repair scheme is to apply the algorithm introduced in the last section. Node i_j transmits $\mathbf{r}_{i_j}$ within the range of $\left[t_{i_j,j} + 1, t_{i_j,j} + L + t_{i,d}\right]$ for repair. Let $\hat{\mathbf{m}}_j = \mathbf{r}_{i_j}[t_{i_j,j} + 1, t_{i_j,j} + L + t_{i,d}]$. $\mathbf{Y}^i$ can be decoded by executing Algorithm 1 on $\hat{\mathbf{m}}_j, j = 1, \ldots, d$. By this scheme, the repair bandwidth is exactly equal to the number of bits of the repaired packets.

According to the analysis of Algorithm 1 in [9], the time complexity of repair is $O\left(d^2\left(L + t_{i,d}\right)\right) = O\left(d^2 L\right)$.

V. Our Recovery Scheme

To recover the original file, in the method described in [4], [5], the d packets stored in any k nodes are first retrieved, and then the linear system formed by the retrieved packets is solved to recover the original file. Note that the linear system has dk equalities (formed by the dk packets) with B variables $\mathbf{x}_1, \ldots, \mathbf{x}_B$. Since $B < dk$, there is redundancy in the linear system. In the following of this paper, we will present a new recovery scheme for the BASIC codes, which only retrieves M bits from any k nodes.

A. Overview of our Scheme

Now let us consider the recovery scheme of BASIC MBR codes. Our scheme recovers the message matrix $\mathbf{M}$ by first recovering the submatrix $\mathbf{T}$ and then recovering the submatrix $\mathbf{S}$. Since the submatrix $\mathbf{S}$ is symmetric, only its upper triangle components need to be recovered.

Recall the matrix $\mathbf{Y}$ defined in (5). We have

$$\mathbf{Y}_{j+k} = \mathbf{\Psi}_{[1,k]}\mathbf{T}_j, j = 1, \ldots, d - k.$$

Since $\mathbf{\Psi}_{[1,k]}$ satisfies the refined increasing difference property, according to our discussion in Section III, $\mathbf{T}_j$ can be recovered by first retrieving a sub-sequence of the $(j + k)$-th packet from any k out of the n storage nodes and then executing Algorithm 1. The $d - k$ columns of $\mathbf{T}$ can be recovered one by one or in parallel.

After recovering $\mathbf{T}$, we continue to recover the k columns of $\mathbf{S}$ sequentially with the column indices in descending order. Fix u with $1 \le u \le k$. We have

$$\begin{aligned}\mathbf{Y}_u &= \mathbf{\Psi}_{[1,k]}\mathbf{S}_u + \mathbf{\Psi}_{[k+1,d]}(\mathbf{T}^u)^\top \\ &= \mathbf{\Psi}_{[1,u]}\mathbf{S}_u^{[1,u]} + \mathbf{\Psi}_{[u+1,k]}\mathbf{S}_u^{[u+1,k]} + \mathbf{\Psi}_{[k+1,d]}(\mathbf{T}^u)^\top. \quad (7)\end{aligned}$$

To recover $\mathbf{S}_u^{[1,u]}$, we use a modified procedure of the one in Section III (to be elaborated in the next subsection): When $u = k$, $\mathbf{S}_k$ can be recovered by first retrieving a sub-sequence of the k-th packet from any k out of the n storage nodes,

substituting the values of $\mathbf{T}^u$, and then executing Algorithm 1. When $u < k$, suppose that the columns of $\mathbf{S}$ with indices larger than u have all been recovered. After retrieving a sub-sequence of the u-th packet from any u out of the n nodes, we substitute the values of $\mathbf{S}_u^{[u+1,k]}$ and $\mathbf{T}^u$ before executing Algorithm 1.

B. Overhead-free In-place Recovery Scheme

Our recovery scheme of a BASIC MBR code is able to recover the file by retrieving data from any k out of the n nodes. Without loss of generality, let the indices of the k nodes be $i_1, i_2, \ldots, i_k$ ($i_1 > i_2 > \cdots > i_k$). Our scheme consists of two stages: the retrieving stage and the decoding stage.

1) Retrieving Stage: In this stage, for $v \in [1, k]$, totally $(d - v + 1)$ packets of L bits are retrieved from node i_v. Particularly, for $u \in [v, d]$, sub-sequence $\mathbf{y}_{i_v,u}[t_{i_v,v} + 1, t_{i_v,v} + L]$ is transmitted to the data collector, and stored as $\hat{\mathbf{m}}_{v,u}$.

Consider the aforementioned example (see equations in (6)). When the data collector connects to node 1, 3 and 4, we have $i_1 = 4$, $i_2 = 3$, and $i_3 = 1$. The bits stored in the data collector are illustrated as follows:

$$\begin{aligned}
&\hat{\mathbf{m}}_{1,1} = \mathbf{y}_{4,1}[1, L],\\
&\hat{\mathbf{m}}_{1,2} = \mathbf{y}_{4,2}[1, L], \hat{\mathbf{m}}_{2,2} = \mathbf{y}_{3,2}[3, L + 2],\\
&\hat{\mathbf{m}}_{1,3} = \mathbf{y}_{4,3}[1, L], \hat{\mathbf{m}}_{2,3} = \mathbf{y}_{3,3}[3, L + 2], \hat{\mathbf{m}}_{3,3} = \mathbf{y}_{1,3}[1, L],\\
&\hat{\mathbf{m}}_{1,4} = \mathbf{y}_{4,4}[1, L], \hat{\mathbf{m}}_{2,4} = \mathbf{y}_{3,4}[3, L + 2], \hat{\mathbf{m}}_{3,4} = \mathbf{y}_{1,4}[1, L],
\end{aligned}$$

where $\mathbf{y}_{i,j}$ are given in (6).

2) Decoding Stage: The details of decoding algorithm are shown in Algorithm 2, which includes two steps. In the first step, matrix $\mathbf{T}$ is recovered using $\hat{\mathbf{m}}_{v,u}$, $1 \le v \le k, k + 1 \le u \le d$. In the second step, matrix $\mathbf{S}$ is recovered using $\hat{\mathbf{m}}_{v,u}$, $1 \le v \le u \le k$. These two steps are explained in detail as follows.

Step 1: Recover $\mathbf{T}$. This step contains $(d - k)$ iterations. For each $u \in [k + 1, d]$, the data collector first uses Algorithm 1 to convert the sequences stored in $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{k,u}$ to sequences $\mathbf{m}_{1,u}, \mathbf{m}_{2,u}, \ldots, \mathbf{m}_{k,u}$ (ref. Line 2). For any v with $1 \le v \le k$, $\mathbf{m}_{v,u}$ is involved in generating $\mathbf{y}_{i_w,v}$, $1 \le w \le v$ (ref. (7)). So the value of $\mathbf{m}_{v,u}$ is substituted in $\hat{\mathbf{m}}_{w,v}$, which is a sub-sequence of $\mathbf{y}_{i_w,v}$ (ref. Line 3–8). After the substitution, $\hat{\mathbf{m}}_{w,v}$ is only related to $\mathbf{m}_{v',u'}, 1 \le v' \le u' \le k$.

Step 2: Recover $\mathbf{S}$. This step contains $k - 1$ iterations. In an iteration indexed by u, Algorithm 1 is used to convert $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{u,u}$ to $\mathbf{m}_{1,u}, \mathbf{m}_{2,u}, \ldots, \mathbf{m}_{u,u}$ (ref. Line 10). Since $\mathbf{m}_{v,u}$ $(1 \le v \le u - 1)$ are involved in generating the bits in $\mathbf{y}_{i_w,v}$ $(1 \le w \le v)$, the values of $\mathbf{m}_{v,u}$ $(1 \le v \le u-1)$ are then substituted into $\hat{\mathbf{m}}_{w,v}(1 \le w \le v)$, which is a sub-sequence of $\mathbf{y}_{i_w,v}$ (ref. Line 11–16). After the substitution, $\hat{\mathbf{m}}_{w,v}$ $(1 \le w \le v \le u - 1)$ are only affected by $\mathbf{m}_{v',u'}, 1 \le v' \le u' \le u - 1$.

After Step 2, $\hat{\mathbf{m}}_{1,1}$ becomes exactly $\mathbf{m}_{1,1}$.

Example 1. Continue considering the aforementioned example with $n = 6$, $k = 3$ and $d = 4$, where the data collector connects to node 1, 3 and 4.

Since $d = k + 1$, Step 1 of Algorithm 2 has only one iteration in this example, which converts $\hat{\mathbf{m}}_{1,4}$, $\hat{\mathbf{m}}_{2,4}$ and $\hat{\mathbf{m}}_{3,4}$ to $\mathbf{m}_{1,4}$, $\mathbf{m}_{2,4}$ and $\mathbf{m}_{3,4}$, respectively. See (14), (15) and (16), where the backslashes ($\backslash$) illustrate the terms cancelled by the decoding algorithm in Step 1. After that, the data collector eliminates $\mathbf{m}_{1,4}$ from $\hat{\mathbf{m}}_{1,1}$ (see (8)), eliminates $\mathbf{m}_{2,4}$ from $\hat{\mathbf{m}}_{1,2}$ and $\hat{\mathbf{m}}_{2,2}$ (see (9) and (10)), and eliminates $\mathbf{m}_{3,4}$ from $\hat{\mathbf{m}}_{1,3}$, $\hat{\mathbf{m}}_{2,3}$ and $\hat{\mathbf{m}}_{3,3}$ (see (11), (12) and (13)).

Step 2 of Algorithm 2 has two iterations with $u = 3, 2$ respectively. In the iteration with $u = 3$, $\hat{\mathbf{m}}_{1,3}$, $\hat{\mathbf{m}}_{2,3}$, and $\hat{\mathbf{m}}_{3,3}$ are converted into $\mathbf{m}_{1,3}$, $\mathbf{m}_{2,3}$ and $\mathbf{m}_{3,3}$. See (11), (12) and (13), where the slashes ($/$) illustrate the terms cancelled by the decoding algorithm in Step 2 with $u = 3$. The data collector eliminates $\mathbf{m}_{1,3}$ from $\hat{\mathbf{m}}_{1,1}$ (see (8)), eliminates $\mathbf{m}_{2,3}$ from $\hat{\mathbf{m}}_{1,2}$ (see (9)) and $\hat{\mathbf{m}}_{2,2}$ (see (10)).

In the iteration with $u = 2$, $\hat{\mathbf{m}}_{1,2}$ and $\hat{\mathbf{m}}_{2,2}$ are converted into $\mathbf{m}_{1,2}$ and $\mathbf{m}_{2,2}$. See (9) and (10), where the crosses ($\times$) illustrate the terms cancelled in this iteration. The data collector eliminates $\mathbf{m}_{1,2}$ from $\hat{\mathbf{m}}_{1,1}$ (see (8)).

After these two steps, the remaining bits are exactly the original file.

Algorithm 2 In-place Decoding for MBR Codes

Input: coded packets stored in $\hat{\mathbf{m}}_{u,v}$ $(1 \le u \le d,\ 1 \le v \le \min(u, k))$, and corresponding node indices i_j $(1 \le j \le k)$, which satisfies $i_1 > i_2 > \cdots > i_k$.
Output: recovered message, stored in $\hat{\mathbf{m}}_{u,v}$.
Step 1: *(Recover* $\mathbf{T}$*)*
1: **for** $u \leftarrow d$ downto $(k + 1)$ **do**
2: Execute Algorithm 1 on $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{k,u}$ with k node indices $i_1, i_2, \ldots, i_k$.
3: **for** $v \leftarrow 1 : k$ **do**
4: **for** $w \leftarrow 1 : v$ **do**
5: $\hat{\mathbf{m}}_{w,v}[t_{i_w,u} - t_{i_w,w} + 1, L] \leftarrow \hat{\mathbf{m}}_{w,v}[t_{i_w,u} - t_{i_w,w} + 1, L] + \hat{\mathbf{m}}_{v,u}[1, L - t_{i_w,u} + t_{i_w,w}]$. (Eliminate superposed bits in recovered sequences)
6: **end for**
7: **end for**
8: **end for**
Step 2: *(Recover* $\mathbf{S}$*)*
9: **for** $u \leftarrow k$ downto 2 **do**
10: Execute Algorithm 1 on $\hat{\mathbf{m}}_{1,u}, \hat{\mathbf{m}}_{2,u}, \ldots, \hat{\mathbf{m}}_{u,u}$ with u node indices $i_1, i_2, \ldots, i_u$.
11: **for** $v \leftarrow 1 : (u - 1)$ **do**
12: **for** $w \leftarrow 1 : v$ **do**
13: $\hat{\mathbf{m}}_{w,v}[t_{i_w,u} - t_{i_w,w} + 1, L] \leftarrow \hat{\mathbf{m}}_{w,v}[t_{i_w,u} - t_{i_w,w} + 1, L] + \hat{\mathbf{m}}_{v,u}[1, L - t_{i_w,u} + t_{i_w,w}]$. (Eliminate superposed bits in recovered sequences)
14: **end for**
15: **end for**
16: **end for**

C. Justification

In Step 1 of Algorithm 2, each column of $\mathbf{T}$ is recovered by executing Algorithm 1, according to [9]. Due to the symmetry of the message matrix, the effect of $\mathbf{T}$ in $\hat{\mathbf{m}}_{i,j}$, $1 \le i \le j \le k$ can be eliminated.

In Step 2 of Algorithm 2, our algorithm guarantees that for $u \in [1, k]$, when recovering the u-th column of $\mathbf{S}$, all sequences in the v-th $(v > u)$ column have been recovered and hence can

$$\hat{\mathbf{m}}_{1,1} = \mathbf{x}_1[1,L] + (\mathbf{0}_3, \cancel{\mathbf{x}_2[1,L-3]}) + (\mathbf{0}_6, \cancel{\mathbf{x}_4[1,L-6]}) + (\mathbf{0}_9, \cancel{\mathbf{x}_7[1,L-9]}) \tag{8}$$

$$\hat{\mathbf{m}}_{1,2} = \mathbf{x}_2[1,L] + (\mathbf{0}_3, \cancel{\mathbf{x}_3[1,L-3]}) + (\mathbf{0}_6, \cancel{\mathbf{x}_5[1,L-6]}) + (\mathbf{0}_9, \cancel{\mathbf{x}_8[1,L-9]}) \tag{9}$$

$$\hat{\mathbf{m}}_{2,2} = \mathbf{x}_3[1,L] + (\cancel{\mathbf{x}_2[3,L]}, \mathbf{0}_2) + (\mathbf{0}_2, \cancel{\mathbf{x}_5[1,L-2]}) + (\mathbf{0}_4, \cancel{\mathbf{x}_8[1,L-4]}) \tag{10}$$

$$\hat{\mathbf{m}}_{1,3} = \mathbf{x}_4[1,L] + (\mathbf{0}_3, \cancel{\mathbf{x}_5[1,L-3]}) + (\mathbf{0}_6, \cancel{\mathbf{x}_6[1,L-6]}) + (\mathbf{0}_9, \cancel{\mathbf{x}_9[1,L-9]}) \tag{11}$$

$$\hat{\mathbf{m}}_{2,3} = \mathbf{x}_5[1,L] + (\cancel{\mathbf{x}_4[3,L]}, \mathbf{0}_2) + (\mathbf{0}_2, \cancel{\mathbf{x}_6[1,L-2]}) + (\mathbf{0}_4, \cancel{\mathbf{x}_9[1,L-4]}) \tag{12}$$

$$\hat{\mathbf{m}}_{3,3} = \mathbf{x}_6[1,L] + \cancel{\mathbf{x}_4[1,L]} + \cancel{\mathbf{x}_5[1,L]} + \cancel{\mathbf{x}_9[1,L]}, \tag{13}$$

$$\hat{\mathbf{m}}_{1,4} = \mathbf{x}_7[1,L] + (\mathbf{0}_3, \cancel{\mathbf{x}_8[1,L-3]}) + (\mathbf{0}_6, \cancel{\mathbf{x}_9[1,L-6]}) \tag{14}$$

$$\hat{\mathbf{m}}_{2,4} = \mathbf{x}_8[1,L] + (\cancel{\mathbf{x}_7[3,L]}, \mathbf{0}_2) + (\mathbf{0}_2, \cancel{\mathbf{x}_9[1,L-2]}) \tag{15}$$

$$\hat{\mathbf{m}}_{3,4} = \mathbf{x}_9[1,L] + \cancel{\mathbf{x}_7[1,L]} + \cancel{\mathbf{x}_8[1,L]}. \tag{16}$$

be eliminated from the retrieved sequences $\hat{\mathbf{m}}_{i,u}$, $i \in [1,u]$. Then recovering the u-th column of $\mathbf{S}$ can be performed by executing Algorithm 1. As the decoding process goes on with u from k to 2, all the components in the upper-triangular part of $\mathbf{S}$ can be recovered. Because $\mathbf{S}$ is symmetric, all components of $\mathbf{S}$ are obtained.

From the above analysis, we have the following theorem.

Theorem 1. *Consider a BASIC MBR code with an encoding matrix $\mathbf{\Psi}$ satisfying the refined increasing-difference property. Fix any k storage nodes indexed by $i_u, 1 \le u \le k$ with $i_1 > i_2 > \ldots > i_k$. For any $v \in [1,k]$ and $u \in [v,d]$, let $\hat{\mathbf{m}}_{v,u} = \mathbf{y}_{i_v,u}[t_{i_v,v}+1, t_{i_v,v}+L]$. Then the file encoded by the BASIC MBR code can be recovered by executing Algorithm 2 on $\hat{\mathbf{m}}_{v,u}$, $1 \le u \le d$, $1 \le v \le \min(u,k)$.*

D. Complexity Analysis

1) Recovery Bandwidth: In the first stage of our scheme, totally $BL = M$ bits are transmitted, which is the minimum possible number of bits required to recover a message of M bits. In [5], however, totally

$$\sum_{j=1}^{k} d\left(L + t_{i_j,d}\right) = BL + \binom{k}{2}L + d\sum_{j=1}^{k} t_{i_j,d}$$

bits are transmitted for recovery when nodes i_j, $j = 1,\ldots,k$ are used for recovery. Our new scheme saves $\binom{k}{2}L + d\sum_{j=1}^{k} t_{i_j,d}$ bits.

2) Time Complexity: XOR operations are primary operations in the decoding stage, so the number of XOR operations can be used to indicate the time complexity.

In Step 1, recovering each column of $\mathbf{T}$ costs k^2L XOR operations and eliminating each column of $\mathbf{T}$ from other retrieved sequences takes $\frac{1}{2}k(k+1)L$ XOR operations. There are $(d-k)$ iterations in this step, so the number of XOR operations required in Step 1 is

$$T_1 = (d-k)\left(k^2 + \frac{1}{2}k(k+1)\right)L.$$

In Step 2, iteration u ($u \in [2,k]$) needs u^2L XOR operations to execute Algorithm 1 and $\frac{1}{2}u(u-1)L$ XOR operations to eliminate the recovered sequences. Therefore, the total number of XOR operations in the second step is

$$T_2 = \sum_{u=2}^{k}\left(u^2L + \frac{1}{2}u(u-1)L\right).$$

Therefore, the total computational complexity is

$$T_1 + T_2 = \left(\left(\frac{3}{2}d - k\right)k^2 + \frac{1}{2}dk - 1\right)L = O\left(dk^2L\right).$$

The number of XOR's operations for recovery is $O\left(d^2k^3nL\right)$ in [5]. Therefore, our decoding algorithm reduces the time complexity in the order of n, d and k.

3) Space Complexity: The only auxiliary space is used to store the temporary data when executing Algorithm 1, which takes $k\log L$ bits. Because the space can be reused for each execution of Algorithm 1, the total auxiliary space is $k\log L$ bits. In contrast, the original recovery algorithm in [5] costs $O\left(BL + \frac{1}{2}k(k-1)L\right) = O(BL)$ auxiliary space.

REFERENCES

[1] A. Dimakis, P. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *IEEE Int. Conf. Computer Communications*, May 2007, pp. 2000–2008.

[2] Y. Wu, R. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Allerton Conf. Control, Comput. Commun.*, Sep. 2007, pp. 242–249.

[3] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[4] 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 Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

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

[6] H. Hou, K. Shum, and H. Li, "Construction of exact-BASIC codes for distributed storage systems at the MSR point," in *IEEE Int. Conf. Big Data*, Oct. 2013, pp. 33–38.

[7] X. Huang, H. Li, T. Zhou, H. Guo, H. Hou, H. Zhang, and K. Lei, "Minimum storage BASIC codes: A system perspective," in *IEEE Int. Conf. Big Data*, Oct. 2013, pp. 39–43.

[8] C. Sung and X. Gong, "A ZigZag-decodable code with the MDS property for distributed storage systems," in *IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 341–345.

[9] X. Fu, Z. Xiao, and S. Yang, "Overhead-free in-place recovery scheme for XOR-based storage codes," in *IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications*, Sep. 2014, pp. 552–557.