Overhead-free In-place Recovery Scheme for XOR-based Storage Codes

Ximing Fu*, Zhiqing Xiao†, and Shenghao Yang‡
*Department of Computer Science and Technology, Tsinghua University, Beijing, China
†Department of Electronic Engineering, Tsinghua University, Beijing, China
‡Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, China
fxm13@mails.tsinghua.edu.cn, xzq.xiaozhiqing@gmail.com, shyang@tsinghua.edu.cn

***Abstract*—This paper proposes a novel recovery scheme for the XOR-based storage codes with the increasing-difference property. For a message of kL bits stored in n storage nodes, a data collector connects any k out of the n storage nodes to recover the message. In our scheme, the data collector acquires exactly L bits for each node, so that no transmission overhead exists. Furthermore, we propose an in-place decoding algorithm that acquires less auxiliary space than the existing decoding algorithm, and the decoding computational complexity of our decoding algorithm is the same as the existing decoding algorithm.**

***Index Terms*—Distributed storage system, maximum distance separable code, recovery scheme, in-place decoding.**

I. Introduction

In a distributed storage system, a message is divided into k blocks, each of which consists of L bits. These k blocks are encoded into n packets using a storage code, each of which is stored in a distinct node. When a data collector (called DC) wants to recover the message, it requires data from a subset of the n nodes. A storage codes is Maximum Distance Separable (MDS) when two preconditions are satisfied, the first of which is that each node in the n nodes stores L bits (perhaps with some overheads), and the second is that DC can recover the message from any k out of the n nodes.

Reed-Solomon code is the most celebrated MDS code, and is widely used in storage systems [1]. The encoding and decoding of Reed-Solomon codes require operations over large finite fields, whose complexity is high. Therefore, storage codes using bitwise exclusive-or's (XOR) for encoding and decoding are of interests due to the low computational cost. Paper [2]–[8] proposed some MDS codes that can correct one, two, or three node failures. Later, [9] proposed an MDS code that can decode at $O\left(k^2L^3\right)$ times. In order to further reduce the complexity of encoding and decoding, a new type of storage codes were proposed in [10]. These codes use bit shifting and fewer XOR operations in the encoding and decoding process, and they are able to recover the message from any k out of the n nodes. Specifically, if the generator matrix of the storage code satisfies the *increasing-difference property*, an associated decoding algorithm, called ZigZag decoding, can correctly recover the message using $O\left(k^2L\right)$ XOR's. However, since both the number of stored bits in each node and the number of transmitting bits from each node are larger than L, the codes proposed in [10] are not strictly MDS.

TABLE I
Comparison between our result and the previous result

Recovery Scheme	Recovery Bandwidth for Node i	Extra Decoding Storage	Decoding Time Complexity
ZigZag decoding [10]	$L + i(k-1)$	$O(kL)$	$O\left(k^2L\right)$
Our recovery scheme	L	$O(k\log L)$	$O\left(k^2L\right)$

Moreover, the ZigZag decoding requires considerable auxiliary space at the same time.

This paper considers the XOR-based storage codes in [10], where the encoding and decoding operations are limited to bit-shifting and XOR. We propose a novel recovery scheme, which is different from the ZigZag decoding, for the storage codes in [10]. The characteristics of this recovery scheme include:

1) *No Transmission Overheads:* In order to recover the message of kL bits, only kL bits are needed to transmit from the k nodes to DC. Therefore, our scheme is optimal in terms of the transmission efficiency.

2) *In-place Decodable:* After the transmissions, the kL bits of message are stored in k vectors of length L. Then an in-place decoding algorithm is executed to transform the k vectors into the recovered message. The algorithm overwrites the data when it is being executed, and only $O(k\log L)$ extra space is needed to store the auxiliary variables. After the algorithm execution completes, the k vectors become exactly the desired recovered message.

3) *Exclusive-or Implementable:* Exclusive-or of two bits is one of the fastest operations to implement. In our recovery algorithm, we use no operations but XOR. Additionally, the number of XOR operations is $O\left(k^2L\right)$ when L is large, and it is the lowest complexity as far as we know.

The comparisons between our scheme and the existing recovery scheme using ZigZag decoding are summarized in Table I.

The rest of the paper is organized as follows: The distributed storage system is described in Section II. And Section III presents our recovery scheme. Specifically, Section III-A and Section III-B introduce the transmission procedure and the decoding procedure in the recovery scheme, respectively. We provide a theorem to show the correctness of the recovery scheme, which is proved in Section V. Moreover, an example

978-1-4799-6513-7/14 $31.00 © 2014 IEEE
DOI 10.1109/TrustCom.2014.70

for the recovery is given in Section IV. Finally, in Section VI, we summarize the paper.

II. System Description

The encoding scheme of the distributed storage system in this paper follows that of [10]. The message consists of k blocks, namely, $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$. The generator matrix $\mathbf{\Psi}$ is an $n \times k$ matrix,

$$\mathbf{\Psi} = \begin{pmatrix} z^{t_{1,1}} & z^{t_{1,2}} & \dots & z^{t_{1,k}} \\ z^{t_{2,1}} & z^{t_{2,2}} & \dots & z^{t_{2,k}} \\ \vdots & \vdots & \ddots & \vdots \\ z^{t_{n,1}} & z^{t_{n,2}} & \dots & z^{t_{n,k}} \end{pmatrix}$$

where $t_{i,j} \in \mathbb{N}\,(1 \le i \le n, 1 \le j \le k)$ satisfies the *increasing-difference property* [10]: for any i, i', j, and j' such that $i < i'$ and $j < j'$,

$$0 < t_{i,j'} - t_{i,j} < t_{i',j'} - t_{i',j}.$$

(Normally, if $t_{i,j} = i\,(j-1)$, the matrix $\mathbf{\Psi}$ is a Vandermonde matrix.)

These k blocks of message are encoded into n packets, $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n$, according to the generator matrix. Specifically, to generate $\mathbf{y}_i$ $(1 \le i \le n)$, pad $t_{i,j}$ zero bits and $(t_{i,k} - t_{i,j})$ zero bits to the leftmost and the rightmost of $\mathbf{x}_j$ $(1 \le j \le k)$, respectively, which results in a sequence of bits of length $(L + t_{i,k})$ denoted by

$$z^{t_{i,j}}\mathbf{x}_j = \left(\mathbf{0}_{1\times t_{i,j}}, \mathbf{x}_j, \mathbf{0}_{1\times (t_{i,k}-t_{i,j})}\right)$$

where $\mathbf{0}_{1\times m}$ is an m-dimensional row vector whose elements are all zero. This padding operation can be regarded as shifting the original block $\mathbf{x}_i$ right by $t_{i,j}$ bit, thus the padded result can be denoted as $z^{t_{i,j}}\mathbf{x}_j$. Then $\mathbf{y}_i = (y_i[1], \cdots, y_i[L + t_{i,k}])$ is formed by

$$\mathbf{y}_i = \sum_{j=1}^{k} z^{t_{i,j}}\mathbf{x}_j. \tag{1}$$

There are n nodes, indexed by $1, 2, \cdots, n$, in the distributed storage system. The ith coded packet $\mathbf{y}_i$ is stored in node i $(1 \le i \le n)$. DC can recover the message from any k out of the n nodes.

III. Recovery Scheme

This section gives our scheme to recover the message from arbitrary k nodes. This scheme consists of two stages, namely, the transmission stage and the decoding stage. Theorem 1 guarantees that DC is able to recover the original message via these two stages.

A. Transmission Stage

Fix a set of k nodes. Now DC needs to recover the original message from the coded packets stored in these k nodes. First, DC sorts the indices of the k nodes in descending order, resulting in the sequence of indices $i_1, i_2, \cdots, i_k$ $(n \ge i_1 > i_2 > ... > i_k \ge 1)$. For $1 \le u \le k$, DC tells node i_u that its index is the uth largest one among those k nodes. Then node i_u obtains the value of $t_{i_u,u}$ and transmits L bits of the packet $\mathbf{y}_{i_u}$ from the $(t_{i_u,u}+1)$th bit to the $(t_{i_u,u}+L)$th bit, to DC. Then DC stores the L bits in $\hat{\mathbf{x}}_u = (\hat{x}_u[1], \cdots, \hat{x}_u[L])$.

Transmission bandwidth: For every node, it needs to transmit L bits. So the total number of bits to transmit is kL.

B. Decoding Stage

Algorithm 1 In-place Decoding

```
Input: coded packets stored in $\hat{\mathbf{x}}_u$, and corresponding node
    indices $i_u$ $(1 \le u \le k)$, which satisfies $i_1 > i_2 > ... > i_k$.
Output: recovered message, stored in $\hat{\mathbf{x}}_u$ $(1 \le u \le k)$.
    Step 1: (Initialize auxiliary variables)
1:  Initialize a vector $(l_1, l_2, \dots, l_k)$ 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)
2:  while $l_k < L$ (Iterate until the message is fully recovered.) do
3:    for $u \leftarrow 1 : k$ do
4:      if $(l_u < L)$ and $(u = 1$ or $l_{u-1} > t_{i_u,u} - t_{i_u,u-1})$
        (Both "and" and "or" here are short-circuit logical
        operators, i.e., the courses in the right operand will
        not be evaluated when the left operand is enough to
        indicate the result.) then
5:        $l_u \leftarrow l_u + 1$; (Freeze one more bit in $\hat{\mathbf{x}}_u$.)
6:        for $v \leftarrow 1 : k$ do
7:          if $v \ne u$ and $0 < l_u + t_{i_v,u} - t_{i_v,v} \le L$ then
8:            $\hat{x}_v[l_u + t_{i_v,u} - t_{i_v,v}] \leftarrow \hat{x}_v[l_u + t_{i_v,u} - t_{i_v,v}]$
                $\oplus \hat{x}_u[l_u]$; (Eliminate the superposed bits.)
9:          end if
10:       end for
11:     end if
12:   end for
13: end while
```

After the transmissions, DC uses an in-place decoding algorithm to recover the message from $\hat{\mathbf{x}}_u$ $(1 \le u \le k)$. The detail of this algorithm is presented in Algorithm 1. In this algorithm, we use a vector $(l_1, \dots, l_k)$ to record the number of decoded bits in $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k)$. Specifically, when $l_u = l$ $(1 \le u \le k)$, the bits $\hat{x}_u[1], \dots, \hat{x}_u[l]$ have been exactly equal to $x_u[1], \dots, x_u[l]$. In the initialization step, all entries in $(l_1, \dots, l_k)$ are set to zero. During the iterations, each entry in $(l_1, \dots, l_k)$ increases gradually to L (see the subsequent description). Finally, all entries in $(l_1, \dots, l_k)$ become L, which means that the message has been fully recovered.

Consider every iteration of the for loop in Line 3. At the beginning of every iteration, DC judges whether $\mathbf{x}_u$ has been fully recovered. If $l_u = L$, the $\mathbf{x}_u$ has been fully recovered, and no further operations are needed. Otherwise $(l_u < L)$, DC tries to decode the bit $x_u[l_u + 1]$. Let l be the current value of $l_u + 1$ $(1 \le l \le L)$. When $u = 1$ or $l_{u-1} > t_{i_u,u} - t_{i_u,u-1}$, the bit $\hat{x}_u[l]$ has become the original message bit $x_u[l_u + 1]$, so DC increases l_u from $l-1$ to l to record the successful recovery of the bit $x_u[l]$. After that, DC eliminates the recovered bit $x_u[l]$ from $\hat{\mathbf{x}}_v$ $(v \ne u)$. According to the encoding scheme, $x_u[l]$ is superposed on $y_{i_v}[l+t_{i_v,u}]$ ($v \neq u$). Note that node i_v transmits $\mathbf{y}_{i_v}$ ranging from $(t_{i_v,v}+1)$ to $(t_{i_v,v}+L)$. Therefore, when $t_{i_v,v} < l+t_{i_v,u} \leq t_{i_v,v}+L$ (or $0 < l+t_{i_v,u}-t_{i_v,v} \leq L$ equivalently), $x_u[l]$ is involved in generating $\hat{x}_v[l+t_{i_v,u}-t_{i_v,v}]$. Line 8 eliminates these superposed bits.

During the execution of the decoding, the entries of $(l_1,\ldots,l_k)$ increase in the following way: At the first several iterations, only l_1 increases while other entries remain zero. After l_1 becomes $(t_{i_2,2}-t_{i_2,1}+1)$, l_2 begins to increase in the same pace with l_1. From then on, (l_2-l_1) is kept to be $(t_{i_2,2}-t_{i_2,1})$ until l_1 reaches L and stops increasing. Similarly, for any $u \neq 1$, l_u begins to increase when l_{u-1} becomes $(t_{i_u,u}-t_{i_u,u-1}+1)$, and (l_u-l_{u-1}) is kept to be $(t_{i_u,u}-t_{i_u,u-1})$ until l_{u-1} reaches L. Finally, all entries in $(l_1,l_2,\ldots,l_k)$ end up in L.

Remarkably, for any snapshot of the decoding process, $l_u > l_{u-1}$ $(1 < u \leq k)$. Therefore, we have

$$L \geq l_1 \geq l_2 \geq \cdots \geq l_k \geq 0$$

at any time. Furthermore, when l_k becomes L, $l_u = L$ $(1 \leq u \leq k)$, and the stored vectors have become the source message, i.e.,

$$\hat{\mathbf{x}}_u = \mathbf{x}_u, \quad 1 \leq u \leq k.$$

The decodability of the in-place decoding algorithm is ensured by the following theorem:

Theorem 1: Consider the MDS storage system with a generator matrix $\mathbf{\Psi}$ satisfying the increasing-difference property. DC collects k nodes indexed by $\mathcal{I} = \{i_u : 1 \leq u \leq k\}$ (where $i_1 > i_2 > \cdots > i_k$ without loss of generality). The node i_u $(1 \leq u \leq k)$ transmits each packet within the range of $[t_{i_u,u}+1, t_{i_u,u}+L]$, and DC stores them in $\hat{\mathbf{x}}_u = (\hat{x}_u[1], \cdots, \hat{x}_u[L])$. DC can decode message using the in-place decoding algorithm, and the recovered k blocks are in $\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_k$.

Theorem 1 is proved in Section V.

C. Complexity Analysis

Space Complexity: In the algorithm, no extra space is needed to store the temporary data, so this algorithm is in-place. Actually, the only auxiliary space to allocate is for the vector $(l_1,l_2,\ldots,l_k)$. Since every entry in $(l_1,l_2,\ldots,l_k)$ should be able to store an integer within the range of $[0,L]$, it needs to occupy $O(\log L)$ bits. Therefore, the overall auxiliary decoding space is $O(k \log L)$ bits.

Time Complexity: In the algorithm, the while-loop (in Line 2) will iterate

$$L + \sum_{u=2}^{k} (t_{i_u,u} - t_{i_u,u-1})$$

times. Considering that (proved in Lemma 1 in Section V)

$$\sum_{u=2}^{k} (t_{i_u,u} - t_{i_u,u-1}) < t_{i_1,k} - t_{i_1,1},$$

TABLE II
PARAMETERS IN THE EXAMPLE

u	i_u	$t_{i_u,u}$	Transmitted bits	Stored bits in DC
1	4	0	$y_4[l]$ $(1 \leq l \leq L)$	$\hat{x}_1[l] = y_4[l]$ $(1 \leq l \leq L)$
2	3	3	$y_3[l]$ $(4 \leq l \leq L+3)$	$\hat{x}_2[l] = y_3[l+3]$ $(1 \leq l \leq L)$
3	1	2	$y_1[l]$ $(3 \leq l \leq L+2)$	$\hat{x}_3[l] = y_1[l+2]$ $(1 \leq l \leq L)$

we have

$$L + \sum_{u=2}^{k} (t_{i_u,u} - t_{i_u,u-1}) < L + t_{i_1,k}.$$

Normally when the generator matrix is the Vandermonde matrix, $t_{i_1,k} < kn$ and $L \gg kn$, the iteration of the while-loop is executed in $O(L)$ times. Additionally, either of the two inner for-loops in Line 3 and Line 6 iterate $O(k)$ times. Therefore, the overall time complexity of this decoding algorithm is $O(k^2 L)$.

D. Comparison with Previous Recovery Scheme

The recovery scheme in [10] also consists of two stages. In the first stage, the storage nodes need to transmit all their packets to DC. For node i, the length of packet is $L+i(k-1)$. Therefore, the number of bits to transmit from node i to DC is $L+i(k-1)$. Compared to this scheme, our scheme only acquires node i to transmit L bit to DC, which is optimal in terms of transmission efficiency.

Furthermore, the ZigZag decoding in [10] needs an $O(kL)$ auxiliary space to record the current decoded process, then does the cancellation according to the records. The auxiliary space is even larger than the recovered message. Compared to the ZigZag decoding, our in-place decoding uses an integer l_u $(1 \leq u \leq k)$ to record the number of decoded bits in the vector $\hat{\mathbf{x}}_u$, and does not require other extra space. In this way, our decoding algorithm uses much less auxiliary space to complete the decoding.

IV. An Example of the Recovery Scheme

This section provides an example to illustrate the recovery scheme. In this example, there are $n = 5$ packets encoded from $k = 3$ message blocks. Here we use the Vandermonde matrix as the generator matrix, which is

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

where $t_{i,j} = i(j-1)$. The source message consists of three blocks $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. After the encoding, the packet stored in node i $(1 \leq i \leq 5)$ is shown in (1).

Next, we will show how to recover $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$. Without loss of generality, let DC connect node 1, node 3, and node 4. The related parameters are shown in Table II. Using the recovery scheme, the packets transmitted to DC are shown in Table III, IV, and V.

TABLE III
DATA STORED IN $\hat{\mathbf{x}}_1$ AFTER THE TRANSMISSIONS

$\hat{\mathbf{x}}_1$	$\hat{x}_1[1]$ $(y_4[1])$	$\hat{x}_1[2]$ $(y_4[2])$	$\hat{x}_1[3]$ $(y_4[3])$	$\hat{x}_1[4]$ $(y_4[4])$	$\hat{x}_1[5]$ $(y_4[5])$	$\cdots$	$\hat{x}_1[8]$ $(y_4[8])$	$\hat{x}_1[9]$ $(y_4[9])$	$\cdots$	$\hat{x}_1[L]$ $(y_4[L])$
	$x_1[1]$	$x_1[2]$	$x_1[3]$	$x_1[4]$	$x_1[5]$	$\ldots$	$x_1[8]$	$x_1[9]$	$\ldots$	$x_1[L]$
$\oplus$					$x_2[1]$	$\ldots$	$x_2[4]$	$x_2[5]$	$\ldots$	$x_2[L-4]$
								$x_3[1]$	$\ldots$	$x_3[L-8]$

TABLE IV
DATA STORED IN $\hat{\mathbf{x}}_2$ AFTER THE TRANSMISSIONS

$\hat{\mathbf{x}}_2$	$\hat{x}_2[1]$ $(y_3[4])$	$\hat{x}_2[2]$ $(y_3[5])$	$\hat{x}_2[3]$ $(y_3[6])$	$\hat{x}_2[4]$ $(y_3[7])$	$\cdots$	$\hat{x}_2[L-3]$ $(y_3[L])$	$\hat{x}_2[L-2]$ $(y_3[L+1])$	$\hat{x}_2[L-1]$ $(y_3[L+2])$	$\hat{x}_2[L]$ $(y_3[L+3])$
	$x_1[4]$	$x_1[5]$	$x_1[6]$	$x_1[7]$	$\ldots$	$x_1[L]$			
$\oplus$	$x_2[1]$	$x_2[2]$	$x_2[3]$	$x_2[4]$	$\ldots$	$x_2[L-3]$	$x_2[L-2]$	$x_2[L-1]$	$x_2[L]$
				$x_3[1]$	$\ldots$	$x_3[L-4]$	$x_3[L-3]$	$x_3[L-2]$	$x_3[L-1]$

TABLE V
DATA STORED IN $\hat{\mathbf{x}}_3$ AFTER THE TRANSMISSIONS

$\hat{\mathbf{x}}_3$	$\hat{x}_3[1]$ $(y_1[3])$	$\hat{x}_3[2]$ $(y_1[4])$	$\hat{x}_3[3]$ $(y_1[5])$	$\cdots$	$\hat{x}_3[L-2]$ $(y_1[L])$	$\hat{x}_3[L-1]$ $(y_1[L+1])$	$\hat{x}_3[L]$ $(y_1[L+2])$
	$x_1[3]$	$x_1[4]$	$x_1[5]$	$\ldots$	$x_1[L]$		
$\oplus$	$x_2[2]$	$x_2[3]$	$x_2[4]$	$\ldots$	$x_2[L-1]$	$x_2[L]$	
	$x_3[1]$	$x_3[2]$	$x_3[3]$	$\ldots$	$x_3[L-2]$	$x_3[L-1]$	$x_3[L]$

Decoded Algorithm: After the transmissions, the transmitted bits are stored in $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$, and $\hat{\mathbf{x}}_3$. Now we execute the decoding algorithm. In the beginning, we can recover $x_1[1]$, $x_1[2]$, and $x_1[3]$ directly, and l_1 changes from 0 to 3. Because $t_{i_3,1} = 0$ and $t_{i_3,3} = 2$, we have $0 < l_1 + t_{i_3,1} - t_{i_3,3} < L$. Let $\hat{x}_3[1] \leftarrow \hat{x}_3[1] \oplus \hat{x}_1[3]$, and $x_1[3]$ is eliminated from $\hat{x}_3[1]$. Next, $l_1 \leftarrow l_1 + 1 = 4$, which indicates that $\hat{x}_1[4] = x_1[4]$. After this update, $l_1 + t_{i_2,1} - t_{i_2,2} = 1$, $l_1 + t_{i_3,1} - t_{i_3,3} = 2$, so we can eliminate $x_1[4]$ from $\hat{x}_2[1]$ and $\hat{x}_3[2]$. Because $l_1 > t_{i_2,2} - t_{i_2,1}$, we perform the operation $l_2 \leftarrow l_2 + 1 = 1$, which results in $\hat{x}_2[1] = x_2[1]$. Then we can eliminate $x_2[1]$ from $\hat{\mathbf{x}}_1$ since $0 < l_2 + t_{i_1,2} - t_{i_1,1} \le L$. As the decoding process goes on, we can decode all the bits of $\hat{\mathbf{x}}_1$ and the bits stored in $\hat{\mathbf{x}}_1$ are exactly the message bits of $\mathbf{x}_1$. In this situation, we have decoded $\hat{\mathbf{x}}_1$, and the bits of $\hat{\mathbf{x}}_1$ have been eliminated from $\hat{\mathbf{x}}_2$ and $\hat{\mathbf{x}}_3$. We can continue the decoding process without considering the first message block.

V. PROOF OF THE THEOREM

This section proves Theorem 1. First, we prove a lemma.

Lemma 1: Assume that the generator matrix satisfies the increasing-difference property. $L \ge i_1 > i_2 > \cdots > i_k \ge 1$. For any u, u' such that $1 \le u < u' \le k$,

$$t_{i_{u'},u'} - t_{i_{u'},u} \overset{(1)}{\le} \sum_{w=u+1}^{u'} \left(t_{i_w,w} - t_{i_w,w-1}\right) \overset{(2)}{<} t_{i_u,u'} - t_{i_u,u}.$$

Proof: (1) Due to the increasing-difference property, for w such that $w < u'$,

$$i_w > i_{u'},$$

so

$$t_{i_w,w} - t_{i_w,w-1} > t_{i_{u'},w} - t_{i_{u'},w-1},$$

and for w such that $w = u'$,

$$i_w = i_{u'},$$

so

$$t_{i_w,w} - t_{i_w,w-1} = t_{i_{u'},w} - t_{i_{u'},w-1}.$$

Therefore, for w such that $w \le u'$,

$$t_{i_w,w} - t_{i_w,w-1} \ge t_{i_{u'},w} - t_{i_{u'},w-1}.$$

Consequently,

$$\begin{aligned}\sum_{w=u+1}^{u'} \left(t_{i_w,w} - t_{i_w,w-1}\right) &\ge \sum_{w=u+1}^{u'} \left(t_{i_{u'},w} - t_{i_{u'},w-1}\right)\\ &= t_{i_{u'},u'} - t_{i_{u'},u}.\end{aligned}$$

(2) Due to the increasing-difference property, for w such that $w > u$,

$$i_w < i_u,$$

and

$$t_{i_w,w} - t_{i_w,w-1} < t_{i_u,w} - t_{i_u,w-1}.$$

Consequently,

$$\begin{aligned}\sum_{w=u+1}^{u'} \left(t_{i_w,w} - t_{i_w,w-1}\right) &< \sum_{w=u+1}^{u'} \left(t_{i_u,w} - t_{i_u,w-1}\right)\\ &= t_{i_u,u'} - t_{i_u,u}.\end{aligned}$$

That proves the lemma. ■

Proof of Theorem 1: (1) Extend the original message from $x_u[l]$ ($1 \le u \le k$, $1 \le l \le L$) to $x'_u[l]$ ($1 \le u \le k$, $-\infty < l < +\infty$) by assuming

$$x'_u[l] = \begin{cases} x_u[l], & 1 \le l \le L \\ 0, & \text{otherwise}. \end{cases}$$

Then the encoding can be reformulated as

$$y_i[l] = \sum_{j=1}^{k} x'_j[l - t_{i,j}],$$

since the padding zeros in $\mathbf{x}'_u$ ($1 \le u \le k$) do not change the value of $\mathbf{y}_i$ ($1 \le i \le n$).

After the $[t_{i_u,u} + 1, t_{i_u,u} + L]$th bits of $\mathbf{y}_{i_u}$ have been transmitted to DC, what $\hat{\mathbf{x}}_u$ stored is

$$\begin{aligned}\hat{x}_u[l] &= y_{i_u}[l + t_{i_u,u}]\\ &= \sum_{v=1}^{k} x'_v[l + t_{i_u,u} - t_{i_u,v}]\\ &= \underbrace{x'_u[l]}_{\text{desired message bit}} + \underbrace{\sum_{\substack{v=1\\ v\ne u}}^{k} x'_v[l + t_{i_u,u} - t_{i_u,v}]}_{\text{superposed bits}}, \qquad (2)\\ &\qquad \text{for } 1 \le l \le L.\end{aligned}$$

(2) Now we to show that, when the value of l_u changes from $(l-1)$ to l in the decoding algorithm ($1 \le u \le k$, $1 \le l \le L$),

the value of $\hat{x}_u[l]$ has been $x'_u[l]$. That's to say, all superposed bits in Eq. (2), namely,

$$x'_v\left[l + t_{i_u,u} - t_{i_u,v}\right], \qquad v \neq u,$$

have been eliminated from $\hat{x}_u[l]$.

The mathematical induction is to be used here: (i) First, we will show that the statement holds when $(l_1, l_2, \ldots, l_k)$ changes for the first time (i.e. l_1 changes from 0 to 1); (ii) Next, we will show that, for every change of $(l_1, l_2, \ldots, l_k)$, if the same statement holds for all previous changes, the statement still holds for this change.

(2.i) The first change of $(l_1, l_2, \ldots, l_k)$ is that the value of l_1 changes from 0 to 1. Due to the increasing-difference property,

$$t_{i_1,1} - t_{i_1,v} < 0, \qquad v \neq 1,$$

which is equivalent to

$$1 + t_{i_1,1} - t_{i_1,v} \leq 0, \qquad v \neq 1.$$

Consequently,

$$x'_j\left[1 + t_{i_1,1} - t_{i_1,v}\right] = 0, \qquad v \neq 1.$$

Therefore, what $\hat{x}_1[1]$ stores after the transmissions is

$$\hat{x}_1[1] = x'_1[1] + \sum_{\substack{j=1 \\ j\neq u}}^{k} x'_j\left[1 + t_{i_u,u} - t_{i_u,j}\right] = x'_1[1].$$

Note that $\hat{x}_1[1]$ does not change before l_1 changes to 1, so $\hat{x}_u[l] = x'_u[l]$ for the first change of $(l_1, l_2, \ldots, l_k)$.

(2.ii) Consider a particular change of $(l_1, l_2, \ldots, l_k)$, say, the change of l_u from $(l-1)$ to l, where $1 \leq l \leq L$. Assume that all previous changes of $(l_1, l_2, \ldots, l_k)$ satisfy the statement that the all the first l_u bits of $\hat{\mathbf{x}}_u$ have been determined and are equal to the first l_u bits of $\mathbf{x}'_u$.

In the decoding algorithm, $L \geq l_1 \geq l_2 \geq \cdots \geq l_k \geq 0$ always holds. Accordingly, we can define e as

$$e = \begin{cases} 0, & l_1 < L \\ \max\{u : l_u = L\}, & l_1 = L, \end{cases}$$

which indicates that e is the largest number within the range of $[0, k]$ such that $l_1 = l_2 = \cdots = l_e = L$. Similarly, we can define γ as

$$\gamma = \max\{u : l_u > 0\}.$$

According to the definition, γ is the smallest number within $[1, k]$ such that $l_{\gamma+1} = \cdots = l_k = 0$. It is obvious that $e \leq u \leq \gamma$, so all v's such that $v \neq u$ fall into one of the four following categories: (a) $1 < v \leq e$; (b) $e < v < u$; (c) $u < v \leq \gamma$; and (d) $\gamma < v \leq k$.

(2.ii.a) For $1 < v \leq e$ (when such v exists), all of the bits in $\mathbf{x}_v$ have been decoded, and their superposed bits have been eliminated completely. Therefore, they do not affect the current decoded bit $\hat{x}_u[l]$ any more.

(2.ii.b) For $e < v < u$ (when such v exists), $1 \leq l_v < L$. In the decoding algorithm, the first time that l_{v+1} increases is when $l_v > t_{i_{v+1},v+1} - t_{i_{v+1},v}$, and both l_{v+1} and l_v increase in the sequent iterations until $l_v \leq L$ no longer holds. Therefore, after the change of l_u,

$$l_v - l_{v+1} = t_{i_{v+1},v+1} - t_{i_{v+1},v},$$

which results in

$$\begin{aligned} l_v - l &= \sum_{w=v}^{u-1} (l_w - l_{w+1}) \\ &= \sum_{w=v}^{u-1} \left(t_{i_{w+1},w+1} - t_{i_{w+1},w}\right) \\ &= \sum_{w=v+1}^{u} (t_{i_w,w} - t_{i_w,w-1}). \end{aligned}$$

Considering Lemma 1,

$$\sum_{w=v+1}^{u} (t_{i_w,w} - t_{i_w,w-1}) \geq t_{i_u,u} - t_{i_u,v},$$

which results in

$$l_v - l \geq t_{i_u,u} - t_{i_u,v},$$

and leads to

$$l + t_{i_u,u} - t_{i_u,v} \leq l_v.$$

Therefore, $x_v\left[l + t_{i_u,u} - t_{i_u,v}\right]$ has been eliminated from $\hat{x}_u[l]$.

(2.ii.c) For $u < v \leq \gamma$ (when such v exists), $1 \leq l_v < L$. In the decoding algorithm, the first time that l_v increases is when $l_{v-1} > t_{i_v,v} - t_{i_v,v-1}$, and both l_v and l_{v-1} increase in the sequent iterations until $l_{v-1} < L$ no longer holds. Therefore, before the change of l_u,

$$l_{v-1} - l_v = t_{i_v,v} - t_{i_v,v-1},$$

and

$$\begin{aligned} (l-1) - l_v &= \sum_{w=u+1}^{v} (l_{w-1} - l_w) \\ &= \sum_{w=u+1}^{v} (t_{i_w,w} - t_{i_w,w-1}). \end{aligned}$$

Considering Lemma 1,

$$\sum_{w=u+1}^{v} (t_{i_w,w} - t_{i_w,w-1}) < t_{i_u,v} - t_{i_u,u}.$$

Hence,

$$l - l_v < t_{i_u,v} - t_{i_u,u} + 1,$$

which leads to

$$l + t_{i_u,u} - t_{i_u,v} \leq l_v.$$

Therefore, $x_v\left[l + t_{i_u,u} - t_{i_u,v}\right]$ has been eliminated from $\hat{x}_u[l]$.

(2.ii.d) For $\gamma < v \leq k$ (when such v exists), $l_v = 0$. $\mathbf{x}_v$ does not affect the current bit $\hat{x}_u[l]$. Therefore, we do not need to consider packet v in this case.

Combining these four cases, when l_u changes to l, for any $v \neq u$, $x'_v\left[l + t_{i_u,u} - t_{i_u,v}\right]$ has been eliminated before, which proves $\hat{x}_u[l] = x'_u[l]$.

(3) Now we show that the value of $\hat{x}_u[l]$ never change after l_u has become l $(1 \leq u \leq k,\ 1 \leq l \leq L)$.

Fix u and l. According to the encoding and decoding algorithm, consider arbitrary v such that $v \neq u$. If there exists a $l_v \in [1, L]$ such that $l = l_v + t_{i_v,u} - t_{i_v,v}$, the vth packet affects $\hat{\mathbf{x}}_u$ once and only once, and the impact is to eliminate the superposed bit $x'_v[l_v]$ out of $\hat{x}_u[l]$. Otherwise, the vth packet has no impact on $\hat{x}_u[l]$. In the previous part of the proof, we have shown that this prospective modification occurs before l_u changes to l. Therefore, it does not affect $\hat{x}_u[l]$ after l_u changes to l, and $\hat{x}_u[l]$ would never change henceforth.

(4) Since the value of $\hat{x}_u[l]$ is $x'_u[l]$ when the value of l_u changes to l, and this value of $\hat{x}_u[l]$ never changes afterwards, the value of $\hat{x}_u[l]$ is $x'_u[l]$ when the decoding algorithm is completed. Note that $x'_u[l] = x_u[l]$ for all $1 \leq u \leq k$, $1 \leq l \leq L$, so what $\hat{\mathbf{x}}_1, \cdots, \hat{\mathbf{x}}_k$ store now are exactly the original message. ■

VI. Conclusion

This paper proposes a recovery scheme for XOR-based MDS storage codes that satisfies the increasing-difference property. Our scheme consists of two stages, namely, a transmission stage and an in-place decoding stage. In the transmission stage, our scheme totally eliminates the transmission overheads. In the decoding stage, our in-place decoding algorithm can decode the received packets with little auxiliary space.

Acknowledgment

This work was supported by the National Basic Research Program of China (973 Program) under Grant 2013CB834205 and the National Natural Science Foundation of China under Grant 61133013.

References

[1] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, pp. 300–304, Jun. 1960.

[2] J. Blmer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," *IGSI Technical Report No. TR-95-048*, Aug. 1995.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.

[4] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 529–542, Mar. 1996.

[5] L. Xu, V. Bohossian, J. Bruck, and D. Wagner, "Low-density MDS codes and factors of complete graphs," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1817–1826, Sep. 1999.

[6] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *USENIX Conf. File and Storage Technologies*, Mar. 2004, pp. 1–14.

[7] G. Feng, R. Deng, F. Bao, and J. Shen, "New efficient MDS array codes for RAID, Part I: Reed-Solomon-like codes for tolerating three disk failures," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1071–1080, Sep. 2005.

[8] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 889–901, Jul. 2008.

[9] G. Feng, R. Deng, F. Bao, and J. Shen, "New efficient MDS array codes for RAID, Part II: Rabin-like codes for tolerating multiple (≥ 4) disk failures," *IEEE Trans. Comput.*, vol. 54, no. 12, pp. 1473–1483, Dec. 2005.

[10] 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.