Repair-Optimal MDS Array Codes over GF(2)

Eyal En Gad*, **Robert Mateescu**†, **Filip Blagojevic**† , **Cyril Guyot**† and **Zvonimir Bandic**†
*Electrical Engineering, California Institute of Technology, Pasadena, CA 91125.
†HGST Research, San Jose, CA 95135.
**eengad@caltech.edu*, †{*robert.mateescu, filip.blagojevic, cyril.guyot, zvonimir.bandic*}*@hgst.com*

***Abstract*—Maximum-distance separable (MDS) array codes with high rate and an optimal repair property were introduced recently. These codes could be applied in distributed storage systems, where they minimize the communication and disk access required for the recovery of failed nodes. However, the encoding and decoding algorithms of the proposed codes use arithmetic over finite fields of order greater than 2, which could result in a complex implementation.**

In this work, we present a construction of 2-parity MDS array codes, that allow for optimal repair of a failed information node using XOR operations only. The reduction of the field order is achieved by allowing more parity bits to be updated when a single information bit is being changed by the user.

I. Introduction

MDS array codes are highly applicable in modern data storage systems. Array codes are non-binary erasure codes, where each symbol is a column of elements in a two dimensional array, and is stored on a different storage node in the system. In traditional erasure codes, the decoder uses all of the available codeword symbols for the recovery of erased symbols. However, in distributed storage systems, this property requires the transmission of an entire array over the network for the recovery of failed nodes. And since node failures are common, the network load caused by node recovery became a major constraint to the application of erasure codes in such systems [5].

For that reason, a lot of attention has been drawn recently to the minimization of the communication required for node recovery. The total amount of information communicated in the network during recovery is called the *repair bandwidth* [4]. In this work we focus on the practical case of systematic MDS array codes with 2 parity nodes. In this case, when 2 nodes are erased, the entire information must be transmitted in order to repair the erased nodes. However, when only a single node is erased, the required repair bandwidth can be lower. It was shown in [4] that the repair bandwidth must be at least $1/2$ of the entire available information in the array. Subsequently, several constructions were designed to achieve that lower bound [2], [3], [6], [9]. Other constructions were also presented for different parameters, such as for the case of low rate in [7], [8].

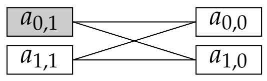
Beside the repair bandwidth, another important parameter of array codes is the *update measure*. In systematic array codes, the elements of the information nodes are called information elements, and those in the parity nodes are called parity elements. The update measure is defined as the number of parity elements that need to change each time an information element is being updated. For MDS array codes, the update measure cannot be smaller than the number of parity nodes. For the codes in [2], [3], [6], [9], the update measure is optimal. Another property of these codes is that the elements of the nodes belong to a finite field of order at least 3. This property can make the codes difficult for hardware implementation. However, it was shown in these papers that for MDS codes with optimal repair bandwidth and optimal update measure, the node elements cannot belong to GF(2). This is the point of departure of this work. Instead of designing codes with optimal update measure, we focus on the design of codes with node elements in GF(2), with the price of a higher update measure. This offers a different trade-off, that can find a wide array of applications.

The main contribution of this work is a construction of systematic MDS array codes with node elements in GF(2). The construction has a similar structure to the ones described in [2], [3], [6], [9]. The codes have 2 parity nodes, and a failure of any information node can be recovered with the access to only $1/2$ of the available information in the array. Note that in general, the amount of *accessed* information in node recovery can be different from the repair bandwidth. Specifically, the total access can be higher than the total bandwidth, but not lower, since there is no reason to communicate more than what is accessed. For that reason, our construction have both optimal access and optimal repair bandwidth in the case of a single information node failure. However, in the case of a parity node failure, the entire information array needs to be transmitted for the recovery. But this is not a major drawback, since a parity node failure does not reduce the availability of the stored data to the user, and thus its recovery can be done offline, and does not affect the system performance significantly. The update measure in our construction is different for different elements. For k information nodes, where k is odd, the expected update is $1/2 \cdot \lfloor k/2 \rfloor + 2$, and the worst-case update is $\lfloor k/2 \rfloor + 2$.

The rest of the paper is organized as following: In section II we demonstrate the key principles of the construction by simple examples. Next, the construction is described formally in section III, with an additional example. Lastly, the properties of the constructions are proven in section IV, and concluding remarks are brought in section V.

II. Demonstrating Examples

A basic principle of the construction can be demonstrated in the case of two information nodes, shown in Figure 1. In this case, each of the columns contains two elements. The information element in row i and column j is denoted as $a_{i,j}$. As is the case in the rest of the paper, there are two parity



$a_{0,1}$	$a_{0,0}$	$h_0 = a_{0,1} + a_{0,0}$	$b_0 = a_{0,0} + a_{1,1}$
$a_{1,1}$	$a_{1,0}$	$h_1 = a_{1,1} + a_{1,0}$	$b_1 = a_{1,0} + a_{0,1} + a_{0,0}$

Figure 1. Decoding a butterfly cycle.

nodes. The first parity node is called the horizontal node, as its elements are encoded as the horizontal parities. The horizontal element in row i is denoted as h_i, and its value is the parity of the information elements in row i. The summations in the table of Figure 1 are taken modulo 2 without mention, as are all of the summations of bits in the rest of the paper.

The second parity node is called the butterfly node, and its element in row i is denoted as b_i. The reason for the name will be clear in the next example. In the figure above the table, the horizontal elements correspond to the horizontal lines, and the butterfly elements to the diagonal lines. However, as shown in the table, the encoding of b_1 contains also the element $a_{0,0}$. In the figure, this is symbolized by the dark color of $a_{0,1}$, that signifies that the element to its right is also added to the corresponding butterfly element.

Now consider the case that column 1 is erased. In this case the column can be decoded using the available elements of row 0 only, by setting $a_{0,1} = h_0 + a_{0,0}$, and $a_{1,1} = b_0 + a_{0,0}$. Since the decoder accesses only half of the elements in each available column, and only half of the available information in total, we say that the *access ratio* is $1/2$.

Since we claim that the code is MDS, consider the case that both information nodes are erased. Notice that if $a_{0,0}$ was not included in b_1, the code could not recover from the loss of the two information nodes. However, the addition of $a_{0,0}$ to b_1, which corresponds to the dark element $a_{0,1}$, allows to break the cycle, and create a decoding chain. From $h_0 + b_1$ we obtain $a_{1,0}$. In the decoding chain that remains in Figure 1 if we eliminate the diagonal $a_{0,1}, a_{1,0}$, we now have all the segments and the end element, and therefore all the other three elements can be decoded. Notice that the addition of $a_{0,0}$ also increases the update measure. If the user wants to change the value of $a_{0,0}$, the encoder needs to update the element b_1, in addition to h_0 and b_0. The code in Figure 1 is also the simplest version of the EVENODD code [1].

Now consider the case of 3 information nodes. In this case, the construction requires that the nodes contain 4 elements, where in general, for k information nodes, the number of elements is 2^{k-1}. Although the size of the column is exponential in the number of columns, this is still practical because the usual number of storage nodes is typically between 10 and 20, and the element of a column is a single bit.

The horizontal elements are encoded in the same way as before, as the parity of the rows. The butterfly node is now encoded with correspondence to its name, where each b_i is encoded according to the line in the *butterfly diagram* of Figure 2 that starts at element $a_{i,0}$. Note that we draw the

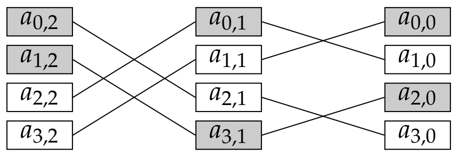


Figure 2. The encoding of the butterfly node.

butterfly with column 0 on the right side. The element b_i is encoded as the parity of the elements in this line, and in addition, if there are *dark* elements in the line, according to Figure 2, extra elements are added to b_i. For each dark element in the line, the element to its right (cyclicly) is also added to b_i. In the general case of k information nodes, the $\lfloor k/2 \rfloor$ elements to the right of a dark elements are added (for odd k, see details in section III). The careful addition of extra elements in the butterfly parity, corresponding to the dark elements, is what allows the computation to be done in GF(2). In this example, $b_0 = a_{0,0} + a_{1,1} + a_{3,2} + a_{0,2}$. The elements $a_{0,0}$, $a_{1,1}$, $a_{3,2}$ come form the butterfly line; additionally, since $a_{0,0}$ is dark, the element to its right (cyclicly), $a_{0,2}$, is also added. Similarly, $b_2 = a_{2,0} + a_{3,1} + a_{1,2} + a_{2,2} + a_{3,0} + a_{1,1}$.

The dark elements in Figure 2 are those $a_{i,j}$ for which the j-th bit in the binary representation of i over $k-1$ bits, is equal to the $(j-1)$-th bit, where the -1-th bit is considered as 0. For example, $a_{0,1}$ is dark since the bit 1 of 0 is equal to bit 0 of 0. Now consider the case that node 1 is failing and needs to be reconstructed. The decoding method for a single node failure is simple: recover the dark elements by the horizontal parities, and the white elements by the butterfly parity. In the example, we set $a_{0,1} = h_0 + a_{0,0} + a_{0,2}$ and $a_{3,1} = h_3 + a_{3,0} + a_{3,2}$, and the dark elements are recovered. For the white elements, we set $a_{1,1} = b_0 + a_{0,0} + a_{3,2} + a_{0,2}$ and $a_{2,1} = b_3 + a_{3,0} + a_{0,2} + a_{0,1}$ (where $a_{0,1}$ was recovered by the horizontal parity). Notice that according to this method, the decoder only access rows 0 and 3, and the access ratio is $1/2$.

Now consider the case that nodes 0 and 1 fail. We can see in Figure 2 that there are two decoding cycles, the cycle of rows 0 and 1, and that of rows 2 and 3. For this decoding, we can ignore the fact the $a_{0,0}$ and $a_{2,0}$ are dark, since the added elements are in column 2, which is available. Therefore, the top cycle becomes identical to the previous example, and can be decoded in the same way. Note that the bottom cycle could not be decoded before the top one. That is since the dark elements of column 2 imply that $a_{0,1}$ and $a_{1,1}$ are added to the butterfly lines of the bottom cycle, and since they are unknown, the cycle cannot be decoded. However, after the decoding of the top cycle, the bottom cycle can be decoded in the same way. In the case of more information columns, the order needed for the decoding of the cycles is related to a binary reflected Gray code, and is described in the next section.

III. Code Construction

For the presentation of the construction we use extra notation. Let $[n] = \{0, 1, \ldots, n-1\}$. For integers i and j,

$i \oplus j$ denotes the bitwise XOR operation between the binary representations of i and j, where the result is converted back to be an integer. The expression $i(j)$ denotes the j-th bit in the binary representation of i, where $i(0)$ is the least significant bit. If j is negative, $i(j)$ is defined to be 0. The construction is now described formally.

Construction 1. *For each pair $(i,j) \in [2^{k-1}] \times [k]$, define a set $B_{i,j}$ as following. If $i(j) \neq i(j-1)$, let $B_{i,j} = \{(i,j)\}$. Else, let*

$$B_{i,j} = \{(i,j') : j' \in [k], j - j' \leqslant \lfloor k/2 \rfloor (mod\, 2\left\lceil \frac{k-1}{2} \right\rceil + 1)\}.$$

(the expression $2\left\lceil \frac{k-1}{2} \right\rceil + 1$ is k for odd k and $k+1$ for even k). Next, let $\ell_{i,j} = i \oplus (2^j - 1)$, and for each $i \in [2^{k-1}]$, define a set

$$B_i = \cup_{j \in [k]} B_{\ell_{i,j},j}.$$

***Encoding**: For each $i \in [2^{k-1}]$, set*

$$h_i = \sum_{j \in [k]} a_{i,j}, \qquad b_i = \sum_{(i',j') \in B_i} a_{i',j'}.$$

***Single failure decoding**: If the failed node is a parity node, use the encoding method. If information node j failed, for each $i \in [2^{k-1}]$ recover $a_{i,j}$ as following: If $i(j-1) = i(j)$, set $a_{i,j} = h_i + \sum_{j' \neq j} a_{i,j'}$. Else, set*

$$a_{i,j} = b_{\ell_{i,j}} + \sum_{(i',j') \in B_{\ell_{i,j}} \setminus \{(i,j)\}} a_{i',j'}.$$

***Double failure decoding**: If both failed nodes are parity nodes, use the encoding method. If one of them is the butterfly node and the other is the information node j, then for each $i \in [2^{k-1}]$, set $a_{i,j} = h_i + \sum_{j' \neq j} a_{i,j'}$, and then encode the butterfly node.*

If the horizontal node fails together with the information node j, decode as following: For $i = 0, 1, \ldots, 2^{k-1} - 1$, find i' according to Algorithm 1, and set

$$a_{i',j} = b_{\ell_{i',j}} + \sum_{(i'',j'') \in B_{\ell_{i',j}} \setminus \{(i',j)\}} a_{i'',j''}.$$

After node j is decoded, encode the horizontal node.

Finally, if two information nodes failed, denote their indices as j_0, j_1, such that $j_1 - j_0 \leqslant \lfloor k/2 \rfloor$ (mod $2\left\lceil \frac{k-1}{2} \right\rceil + 1$). Next, for $i = 0, 1, \ldots, 2^{k-2} - 1$, find i_0, i_1 according to Algorithm 2, and set, sequentially,

$$\begin{aligned} a_{i_1,j_0} =& h_{i_0} + \sum_{j' \in [k] \setminus \{j_0,j_1\}} a_{i_0,j'} \\ &+ b_{\ell_{i_1,j_0}} + \sum_{(i',j') \in B_{\ell_{i_1,j_0}} \setminus \{(i_1,j_0),(i_0,j_0),(i_0,j_1)\}} a_{i',j'} \end{aligned} \quad (1)$$

$$a_{i_1,j_1} = h_{i_1} + \sum_{j' \in [k] \setminus \{j_1\}} a_{i_1,j'} \quad (2)$$

$$a_{i_0,j_0} = b_{\ell_{i_0,j_0}} + \sum_{(i',j') \in B_{\ell_{i_0,j_0}} \setminus \{(i_0,j_0)\}} a_{i',j'} \quad (3)$$

$$a_{i_0,j_1} = h_{i_0} + \sum_{j' \in [k] \setminus \{j_1\}} a_{i_0,j'} \quad (4)$$

Algorithm 1 Find i'.

```
1: Inputs: $i \in [2^{k-1}]$
2: Output: $i' \in [2^{k-1}]$
3: $i'(k-1) \leftarrow 0$
4: for $j' = k-2$ to $j' = j$ do
5:     $i'(j') \leftarrow i'(j'+1) + i(j')$
6: end for
7: for $j' = 0$ to $j' = j-1$ do
8:     $i'(j') \leftarrow i'(j'-1) + i(j')$
9: end for
```

Algorithm 2 Find i_m.

```
1: Inputs: $m \in \{0,1\}, i \in [2^{k-2}]$
2: Output: $i_m \in [2^{k-1}]$
3: $i_m(k-1) \leftarrow 0$
4: $s \leftarrow \arg\max_{i' \in \{0,1\}} \{j_{i'}\}$
5: for $j = k-2$ to $j = j_s$ do
6:     $i_m(j) \leftarrow i_m(j+1) + i(j)$
7: end for
8: for $j = 0$ to $j = j_{1-s} - 1$ do
9:     $i_m(j) \leftarrow i_m(j-1) + i(j)$
10: end for
11: $i_m(j_1 - s) \leftarrow i_m(j_1 + s - 1) + m$
12: if $j_s - j_{1-s} > 1$ then
13:     for $j = j_1 + 1 - 3s$ to $j_0 + s - 1$ do
14:         $i_m(j) \leftarrow i_m(j + 2s - 1) + i(j + s - 1)$
15:     end for
16: end if
```

Algorithms 1 and 2 seem a bit complex, but are in fact only a slight modification to the standard method of decoding a binary reflected Gray code into a binary number. We see this in the following example.

Example 1. *Let $k = 4$, and assume that the information node $j = 2$ fails together with the horizontal parity node. By Construction 1, we first decode node j in 2^{k-1} iterations, and then decode the horizontal node. In each iteration i, we first find the index i' of the element $a_{i',j}$ to be decoded in that iteration. In iteration $i = 0$ of the decoding, it is easy to see that Algorithm 1 sets $i' = 0$, so $a_{0,2}$ is the first element to be decoded. To decode $a_{0,2}$ we need to find the set $B_{\ell_{0,2}}$, so we first calculate $\ell_{0,2} = 0 \oplus (2^2 - 1) = 3$. By definition, $B_3 = \cup_{j' \in [4]} B_{\ell_{3,j'},j'}$. For $j' = 0$, $\ell_{3,0} = 3 \oplus (2^0 - 1) = 3$. To find $B_{3,0}$, we notice that bit 0 in the binary representation of 3 is 1 while bit -1 is 0 (for any integer by our convention). So $B_{3,0} = \{(3,0)\}$. The fact that $B_{3,0}$ has only 1 element can also be seen in Figure 3, since $a_{3,0}$ is white (not dark). Similarly, the elements $a_{2,1}$ and $a_{4,3}$ are also in the same butterfly line with $a_{0,2}$ ($\ell_{3,1} = 2$ and $\ell_{3,3} = 4$), and are also white. So $B_{2,1} = \{(2,1)\}$, and $B_{4,3} = \{(4,3)\}$. For $j' = 2$, we need to* *find $B_{0,2}$. Unlike all the other columns, here bit 2 of 0 is equal to bit 1 of 0 ($a_{0,2}$ is dark), so $B_{0,2}$ contains multiple elements. Following the definition we see that $B_{0,2} = \{(0,0),(0,1),(0,2)\}$, since $2-0 \leqslant 2 \pmod 5$, $2-1 \leqslant 2 \pmod 5$ and $2-2 \leqslant 2 \pmod 5$, but $2-3 = 4 > 2 \pmod 5$. So in summary, $B_3 = \{(3,0),(2,1),(0,2),(0,1),(0,0),(4,3)\}$. To decode $a_{0,2}$, we need all of the rest of the elements in B_3 to be available for us, since b_3 was encoded to be the sum of the elements with indices in B_3. Remember that column 2 is the only information node that failed (together with the horizontal node), and notice that none of the elements in B_3 (except $(0,2)$) is from column 2. So all of them are available, and we can decode $a_{0,2}$ successfully, by calculating*

$$a_{0,2} = b_3 + a_{3,0} + a_{2,1} + a_{0,1} + a_{0,0} + a_{4,3}.$$

We continue the example for one more iteration. Here $i = 1$, and we turn to Algorithm 1 in order to find i'. In line 3 we set $i'(k-1) = i'(4) = 0$. In line 4 to 6, we set $i'(3) = 0 + i(3) = 0$, and then $i'(2) = 0$. In lines 7-9, we first set $i'(0) = i'(-1) + i(0) = 0 + 1 = 1$, since $i'(-1)$ is defined to be 0. Next we set $i'(1) = 1 + 0 = 1$, and in conclusion we get that $i' = 11_2 = 3$, so in iteration 1 we decode the element $a_{3,2}$. We note that Algorithm 1 can be interpreted visually by observing Figure 3. In each row, consider the dark elements as $'0'$s, and the white elements as $'1'$s, and ignore column j. i' is set to be the index of the row for which the binary number resulting from this observation is equal to i. In the current example, row 3 has a white element in column 0, and dark elements in columns 1 and 3, corresponding to $i = 001_2 = 1$. So in iteration $i = 1$, i' is set to be 3, and we see again that element $a_{3,2}$ is decoded in this iteration. In fact, in order to find i' we need to perform an operation very similar to the decoding of a binary reflected Gray code into a binary number. Next, we can see that elements $a_{0,0}$, $a_{1,1}$ and $a_{7,3}$ are in the same butterfly line with $a_{3,2}$, so they are needed for the decoding of $a_{3,2}$. However, since they are all in columns other than 2, they are all available for the decoder. In addition, we see that $a_{1,1}$ and $a_{7,3}$ are both white, so they do not contribute additional elements to the parity element used in this iteration. However, since $a_{0,0}$ is dark, it contributes an additional element from row 0. But since all the elements in row 0 are dark, we know that row 0 was decoded in iteration 0, and therefore all of its elements are already known, and the decoding can take place successfully. Finally, we note that Algorithm 2 is based on the same idea, and can be interpreted visually in the same way.

IV. Code Properties

In this section we show that the codes have optimal access, that they are MDS, and present their update measure. The first Theorem proves that the single failure decoding function of Construction 1 accesses only half of the elements in each surviving node, and thus Construction 1 is said to be "repair-optimal".

Theorem 1. ***Optimal Repair:*** *The single failure decoding function of Construction 1 decodes any failed information node correctly, and it accesses only $1/2$ of the elements in each of the surviving nodes.*

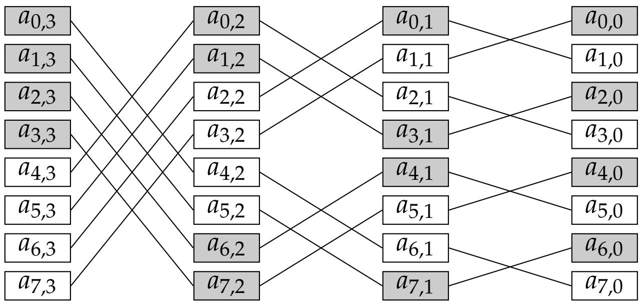


Figure 3. The butterfly construction with 4 information nodes

Proof: Let j be the failed node. First, note that the fraction of elements $i \in [2^{k-1}]$ s.t. $i(j-1) = i(j)$ is $1/2$, and therefore the decoder accesses half of the elements in the horizontal node. Next, note that when j is fixed, the function $f(i) = i \oplus (2^j - 1)$ is a permutation, and therefore the decoder also access only half of the elements in the butterfly node. Finally, we will show that for each accessed element $a_{i',j'}$ in the information nodes, $i'(j-1) = i'(j)$, and thus the decoder only access half of the elements in each node, and the repair ratio is $1/2$.

Let $a_{i,j}$ be a decoded element, and $a_{i',j'}$ be an element that is accessed in the decoding process. If $i(j-1) = i(j)$, then $i' = i$ by the decoding function, and thus $i'(j-1) = i'(j)$. Else, note that (i',j') is in $B_{\ell_{i,j}} = B_{i \oplus (2^j - 1)}$, and since $B_i = \cup_{j \in [k]} B_{\ell_{i,j},j}$, it follows that $i' = i \oplus (2^j - 1) \oplus (2^{j''} - 1)$ for some $j'' \neq j$. If $j'' > j$, then $i'(j-1) = i(j-1)$, and $i'(j) = i(j) + 1$. And if $j'' < j$, then $i'(j-1) = i(j-1) + 1$, and $i'(j) = i(j)$. In both cases,

$$i'(j-1) + i'(j) = i(j-1) + i(j) + 1 = 1 + 1 = 0,$$

and therefore $i'(j) = i'(j-1)$, and the proof is completed. ■

The next theorem verifies the MDS property of the Construction.

Theorem 2. ***MDS:*** *The double failure decoding function of Construction 1 decodes the failure of any two nodes correctly.*

Proof: In the case that one of the failed nodes is the butterfly node, the proof is trivial by the the encoding method. If the horizontal node failed together with the information node j, we need to show that in each iteration i, all of the elements $a_{i'',j}$, where $(i'',j) \in B_{\ell_{i',j}} \setminus \{(i',j)\}$, were decoded in a previous iteration. To prove that, note that by the definition of the set, if (i'',j) is in $B_{\ell_{i',j}} \setminus \{(i',j)\}$, then there exists j' such that $i'' = i' \oplus (2^j - 1) \oplus (2^{j'} - 1)$ and $i''(j') = i''(j'-1)$. So it is enough to show that for each $j' \neq j$, such that $i''(j') = i''(j'-1)$, $a_{i'',j}$ was decoded in a previous iteration.

We prove this by induction on the iteration i. In the base case, $i = 0$. and according to Algorithm 1, $i' = 0$ as well. Now by the definition of i'', $i''(j') \neq i''(j'-1)$, and the base case is proven.

For the induction step, assume that $i''(j') = i''(j'-1)$. By the definition of i'', $i'(j') + i'(j'-1) = i''(j') + i''(j'-1) + 1 = 1$. In addition, for any $j'' \neq j'$, $i'(j'') + i'(j''-1) = i''(j'') + i''(j''-1)$. Therefore, according to Algorithm 1, the iteration in which $a_{i'',j}$ needs to be decoded differ from i in exactly one bit. And since $i'(j') \neq i'(j'-1)$, the value of that bit in i is 1, and therefore i is a later iteration, and $a_{i'',j}$ was decoded before. So by the induction hypothesis, $a_{i'',j}$ is known, and the induction is proven. So, by the encoding of the butterfly elements, column j is decoded successfully, and the horizontal node can be encoded afterwards.

In the case that both failed nodes are information nodes, the proof is very similar. First we need to show that all of the terms in Equation (1) are known when a_{i_1,j_0} is being decoded. For each $j' \in [k] \setminus \{j_0, j_1\}$, $a_{i_0,j'}$ is known since it's an element of a surviving node. For $a_{i'',j''}$ where $(i'', j'') \in B_{\ell_{i_1,j_0}} \setminus \{(i_1, j_0), (i_0, j_0), (i_0, j_1)\}$, we use induction on i again.

First, notice that (i_0, j_1) is actually in $B_{\ell_{i_1,j_0}}$. That is since according to Algorithm 2, $i_0 = i_1 \oplus (2^{j_0} - 1) \oplus (2^{j_1} - 1)$, where the difference between i_0 and i_1 comes form lines $11 - 16$ in the Algorithm. Therefore, $\ell_{i_1,j_0} = i_1 \oplus (2^{j_0} - 1) = i_0 \oplus (2^{j_1} - 1) = \ell_{i_0,j_1}$. In addition, by line 11 of Algorithm 2, $i_0(j_1) = i_0(j_1 - 1)$, and therefore, $B_{i_0,j_1} = \{(i_0, j' : j_1 - j' \leqslant \lfloor k/2 \rfloor \pmod{2\left\lceil \frac{k-1}{2} \right\rceil + 1})\}$, which implies that $(i_0, j_0) \in B_{\ell_{i_1,j_0}}$ as well. The inductive argument follows the same lines as in the previous case, and is therefore omitted.

At this point, we know that all of the terms in Equation 1 are known. Now notice that $h_{i_0} + \sum_{j' \in [k] \setminus \{j_0, j_1\}} a_{i_0,j'} = a_{i_0,j_0} + a_{i_0,j_1}$, and $b_{\ell_{i_1,j_0}} + \sum_{(i',j') \in B_{\ell_{i_1,j_0}} \setminus \{(i_1,j_0),(i_0,j_0),(i_0,j_1)\}} a_{i',j'} = a_{i_1,j_0} + a_{i_0,j_0} + a_{i_0,j_1}$, and therefore, a_{i_1,j_0} is decoded correctly. After a_{i_1,j_0} is decoded, it can be seen directly by Equation (2) that a_{i_1,j_1} can be decoded correctly as well. As for a_{i_0,j_0}, it can be shown by the same argument that we used for a_{i_1,j_0}, that it could be decoded successfully. And finally, the decoding of a_{i_0,j_1} also follows immediately. ■

Lastly, we present the update measure of the Construction.

Theorem 3. ***Update:*** *The expected update measure of Construction* 1 *is* $1/2 \cdot \lfloor k/2 \rfloor + 2$*, and the worst-case is* $\lfloor k/2 \rfloor + 2$.

Proof: For a uniformly-distributed randomly-picked pair $(i, j) \in [2^{k-1}] \times [k]$, the probability that $i(j) = i(j-1)$ is $1/2$. Therefore, in addition to $B_{\ell_{i,j},j}$, the expected number of sets $B_{i',j'}$ that contain (i, j) is $1/2 \cdot \lfloor k/2 \rfloor + 1$. In the case that the value of $a_{i,j}$ is changed, each of these sets require the update of an element in the butterfly node, in addition to a single element in the horizontal node. Therefore, the expected number of updated elements is $\lfloor k/2 \rfloor \cdot 1/2 + 2$.

In the worst case, consider the update of an element $a_{0,j}$, for $j \in [k]$. For each $j' \in [k] \setminus \{j\}$ such that $j' - j \leqslant \lfloor k/2 \rfloor \pmod{2\left\lceil \frac{k-1}{2} \right\rceil + 1}$, $i(j') = i(j'-1)$, and therefore, $(0, j) \in B_{\ell_{0,j'}}$. For that reason, $\lfloor k/2 \rfloor + 1$ elements of the butterfly node need to be updated, in addition to a single element in the horizontal node, and the total is $\lfloor k/2 \rfloor + 2$. ■

V. Conclusions

In this paper, we described a construction of repair-optimal MDS array codes, whose array elements are bits, and the operations are performed over GF(2). Several problems are still open in this topic. First, it could be interesting to find out whether there exist repair optimal MDS codes with lower update measure. Second, a generalization of the construction to more parity nodes could be very useful. And finally, it would be important to know whether such codes exist whose number of rows is polynomial in the number of columns.

VI. Acknowledgments

This work was done while Eyal En Gad was at HGST Research. We would like to thank Yuval Cassuto for his help in reducing the worst-case update measure of the codes.

References

[1] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: an efficient scheme for tolerating double disk failures in raid architectures," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 192 –202, feb 1995.

[2] V. R. Cadambe, S. A. Jafar, and H. Maleki, "Minimum repair bandwidth for exact regeneration in distributed storage," in *Wireless Network Coding Conference (WiNC), 2010 IEEE*, june 2010, pp. 1 –6.

[3] V. Cadambe, C. Huang, and J. Li, "Permutation code: Optimal exact-repair of a single failed node in mds code based distributed storage systems," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, 31 2011-aug. 5 2011, pp. 1225 –1229.

[4] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539 –4551, sept. 2010.

[5] O. Khan, R. Burns, J. Plank, and C. Huang, "In search of i/o-optimal recovery from disk failure," in *Hot Storage 2011, 3rd Workshop on Hot Topics in Storage and File Systems, Portland, OR*, Jun. 2011.

[6] D. Papailiopoulos, A. Dimakis, and V. Cadambe, "Repair optimal erasure codes through hadamard designs," in *Communication, Control, and Computing, 49th Annual Allerton Conference on*, sept. 2011, pp. 1382 –1389.

[7] 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," *Information Theory, IEEE Transactions on*, vol. 57, no. 8, pp. 5227 –5239, aug. 2011.

[8] N. Shah, K. Rashmi, P. Kumar, and K. Ramchandran, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *Information Theory, IEEE Transactions on*, vol. 58, no. 4, pp. 2134 –2158, april 2012.

[9] I. Tamo, Z. Wang, and J. Bruck, "Mds array codes with optimal rebuilding," in *Proceedings of the IEEE International Symposiom on Information Theory (ISIT), Saint Petersburg, Russia*, Jun. 2011, pp. 1493–1495.