# Product Matrix MSR Codes with Bandwidth Adaptive Exact Repair

Kaveh Mahdaviani*, Soheil Mohajer†, and Ashish Khisti*
*ECE Dept., University of Toronto, Toronto, ON M5S3G4, Canada
†ECE Dept., University of Minnesota, Minneapolis, MN 55404, USA
Email: {kaveh, akhisti}@comm.utoronto.ca, soheil@umn.edu

**Abstract**

In a distributed storage systems (DSS) with $k$ systematic nodes, robustness against node failure is commonly provided by storing redundancy in a number of other nodes and performing repair mechanism to reproduce the content of the failed nodes. Efficiency is then achieved by minimizing the storage overhead and the amount of data transmission required for data reconstruction and repair, provided by coding solutions such as regenerating codes [1]. Common explicit regenerating code constructions enable efficient repair through accessing a predefined number, $d$, of arbitrary chosen available nodes, namely helpers. In practice, however, the state of the system dynamically changes based on the request load, the link traffic, *etc.*, and the parameters which optimize system's performance vary accordingly. It is then desirable to have coding schemes which are able to operate optimally under a range of different parameters simultaneously. Specifically, adaptivity in the number of helper nodes for repair is of interest. While robustness requires capability of performing repair with small number of helpers, it is desirable to use as many helpers as available to reduce the transmission delay and total repair traffic.

In this work we focus on the minimum storage regenerating (MSR) codes, where each node is supposed to store $\alpha$ information units, and the source data of size $k\alpha$ could be recovered from any arbitrary set of $k$ nodes. We introduce a class MSR codes that realize optimal repair bandwidth simultaneously with a set of different choices for the number of helpers, namely $D = \{d_1, \cdots, d_\delta\}$. We show that comparing to the only other explicit constructions with such capabilities, presented in [2], the required value for $\alpha$ is exponentially smaller. In our construction, particularly for a DSS with $n$ nodes, and $k$ systematic nodes, the required value for $\alpha$ is reduced from $s^n$ to $sk$, where $s = \mathrm{lcm}\,(d_1 - k + 1, \cdots, d_\delta - k + 1)$. We also show the required field size is significantly smaller in the proposed construction.

## I. Introduction

Distributed storage systems (DSS) are compelling solutions to the fast growth of the demand in storage and accessibility of digital content. The main feature of such systems is to provide accessibility and durability for the stored data by introducing redundancy. In parallel, the number of storage components as well as the number of users connecting to these servers are dramatically increasing. These increase the chance of failures in the system, due to storage device failures or inaccessible nodes with overloaded traffic. Replication is the simplest approach to make the distributed storage system robust against such failures, which is implemented in systems such as [3]. This approach however, provides simplicity in the cost of huge storage overhead.

In the literature of erasure codes, there exists solutions, such as the Reed-Solomon (RS) code [4], which offer similar fault-tolerance level as the replication does, with significantly less storage overhead. For instance, a replication based DSS needs to accommodate two mirrors for every single storage node in order to achieve reliability guarantee against any simultaneous two node failure, which arises $200\%$ storage overhead, while *maximum distance separable* (MDS) codes, such as RS code, achieve the same level of fault-tolerance guarantee by adding only two redundant storage nodes for the whole system. In other words, an erasure code with *MDS property* can guarantee to recover the whole source data from any subset of stored encoded segments of collective size equal to the source data size.

On the other hand, when a node fails in a DSS, it needs to be replaced by a new node in order to maintain the system's performance. Such procedure is referred to as a *repair*. To perform a repair, the system downloads some data from a subset of the surviving nodes, called *helper nodes*. The amount of data downloaded for a repair is referred to as the *repair bandwidth*. In conventional MDS erasure codes, such as the RS codes, one has to reconstruct the entire source data and re-encode it to recover a single lost segment. Hence, the repair bandwidth of these codes is at least as large as the size of the whole source data.

Considering both storage overhead and repair bandwidth simultaneously, the family of *regenerating codes* [1], [5] offers a very efficient coding mechanism for distributed storage systems. More precisely, a regenerating code on a Galois field $\mathbb{F}_q$ for a DSS with $n$ storage nodes, maps the source data of size $F$ symbols into $n$ pieces of size $\alpha$ symbols each, and stores them in $n$ separate nodes, such that any $k$ out of $n$ nodes suffice to recover the data. Such a system is capable to tolerate up to $(n-k)$ node failures. Moreover, upon failure of one node, it can be replaced by a new node whose content of size $\alpha$ symbols is determined by connecting to an arbitrary set of $d$ (where $d \geq k$) helper nodes, and downloading $\beta$ symbols form each (where $\beta \leq \alpha$). Hence, the total repair bandwidth in regenerating codes is $d\beta$, which is denoted by $\gamma$. Also, the parameter $\alpha$ is referred to as the *sub-packetization level*.

Ideally, one would like to minimize the storage overhead, and repair bandwidth simultaneously. It turns out that for a given file size $F$, there is a trade-off between the sub-packetization level $\alpha$ and the repair bandwidth $\gamma$, and one can be minimized only at the cost of a penalty for the other [1]. In particular, at one extreme point of this trade-off, one could first minimize the sub-packetization level, $\alpha$, and then minimize the *per-node repair bandwidth*, $\beta$, to obtain a *minimum storage regenerating* (MSR) code. As a result, MSR codes have the MDS property, and also minimize the repair bandwidth for the given $\alpha$ [1], which means for an MSR code we have $F = k\alpha$, and

$$\beta = \frac{F}{k(d-k+1)}. \tag{1}$$

The total repair bandwidth ($d\beta$) of an MSR code is then upper-bounded by that of RS code, and only coincides with that when $d = k$. In other word, the total repair bandwidth is decreasing super-linearly as $d$ grows in MSR codes.

Reversing the order of minimization between $\alpha$, and $\beta$ results in another extreme point of the trade-off. Such codes are not MDS and hence have more storage overhead but provide the smallest possible repair bandwidth and referred to as *minimum repair bandwidth* (MBR) codes. Our focus in this work is on MSR codes as they minimize the storage cost.

In general, two distinct types of repair can be identified: in an *exact repair* scenario the replacement generated node will contain the same data as stored in the failed node. In the so called *functional-repair*, however, the replacement node may store a different content, provided that the entire new system maintain the same properties as of the original one. In practice, exact repair regenerating codes are much more appealing, mainly due to the fact that they could provide systematic encoding, which is a significant advantage in practice.

It is shown that for functional-repair MSR code design could be translated into a linear network coding problem [1]. Design of such codes with exact repair property is more challenging, due to a large number of constraints need to be simultaneously satisfied. Various types of MSR code constructions are proposed for various set of parameters [2], [6]–[16].

Among the numerous available regenerating code designs, the common adopted model considers a predetermined number $d$ (where $k \leq d \leq n-1$) of helpers required for any repair procedure. Each of these helpers is also assumed to provide $\beta = \gamma/d$ repair symbols. This sets a single threshold for the system's capability to perform the repair. On the other hand, in practice the state of the system dynamically changes as a function of various factors including availability of nodes, traffic load, available bandwidth, *etc.* It has been shown that when such characteristics changes in the system the optimal number of helper nodes for minimizing the cost of repair changes [17]. Therefore, runtime adaptation would be of great value towards optimizing the performance. For instance, when the system is heavily loaded by many read requests, many nodes in the network might not be capable to provide the required repair bandwidth and hence would be considered unavailable for the repair. This may result in system's failure to perform the repair, while there might be a few nodes, less than $d$, which are capable of providing even more than $\beta$ repair symbols. A natural question is whether one can download more data from the available helpers, and accomplish the repair without the busy helpers. Moreover, from (1) it is clear that in optimal repair of MSR codes, increasing the number of helpers reduces both $\beta$, and $\gamma = d\beta$. One could consider a situation in which there are many nodes, more than $d$, which are capable of contributing to the repair. It would be of great practical value then if the system could increase the number of helpers and reduce both the per-node and total repair traffic. Note that this will also reduce the transmission delay, which is one of the main bottlenecks in the DSS's performance. We refer to such property as *bandwidth adaptivity*. Note that the dynamic capability of service for storage nodes is a well-known characteristic for many practical distributed systems such as peer-to-peer systems or heterogeneous network topologies [18]–[23].

The design of such codes has been of interest and the significance of bandwidth adaptivity in the performance of the system has been emphasised in [7], [24]–[26]. However, it is a challenging problem to design such coding scheme with a large flexibility degree since it needs to satisfy many optimality conditions simultaneously. As a result, this problem has only been considered for the MSR [2], [7], and MBR [27] extreme points of the tradeoff. For the MBR case, [27] provided a solution for a wide range of practical parameters based on the *Product Matrix* framework introduced in [6]. In [7] a solution is provided based on interference alignment, which only achieves the MSR characteristics when both $\alpha$ and $\beta$ tend to infinity. The first explicit exact repair MSR code constructions which satisfy the bandwidth adaptivity are introduced in [2]. These constructions work for any parameters $k$, $n$, and all values of $d$ such that $k < d < n$. Although these constructions can achieve optimality for finite values of $\alpha$ and $\beta$, but the required value for these parameters are still very huge (*i.e.* exponentially large in $n$), and hence they only achieve optimality for extremely large contents. Recently, [15] introduced a modified version of the code constructions in [2] which achieves MSR optimality for much lower values of $\alpha$, at the cost of loosing bandwidth adaptivity. Indeed the MSR code in [15] works only for $d = n-1$.

In this work we will address the design problem of MSR codes with bandwidth adaptive exact repair for small $\alpha$, and $\beta$, following the Product Matrix framework [6]. The code allows us to choose the number of helper nodes for each repair scenario based on availability of the nodes, network traffic and load state of the nodes, and it is capable to adjust the per-node repair bandwidth, $\beta$, to its optimum value based on the number of selected helpers as in (1). Compared to the constructions proposed in [2] for a DSS with $n$ storage nodes the required values for $\alpha$ and $\beta$ in the presented code is reduced exponentially for the same set of other parameters. We also show that the required filed size is smaller in the presented coding scheme. The main contributions of this work are explained in the next section, after formally defining the problem setup.

The rest of this paper is organized as follows: The following section formally introduced the problem setup and summarizes the main contributions. Section III briefly reviews the most relevant works in the literature. Coding scheme and a few examples are presented in Section IV, which is followed by a discussion on the properties of the code in Section V. Conclusion and abstracts which contain more lengthy proofs and examples are provided at the end.

## II. Model and Main Results

### A. Model

In this section we will briefly introduce a setup for the distributed storage system and the coding scheme of our interest. This model is a modified version of the original setup considered in [1], and is very similar to the model considered in [27].

The first element we consider for the model of our bandwidth adaptive distributed storage system is a predefined Galois filed alphabet, $\mathbb{F}_q$ of size $q$. Hereafter we assume all the symbols stored or transmitted through the network are elements of $\mathbb{F}_q$. Besides, we consider a homogeneous group of $n$ storage nodes, each capable of storing $\alpha$ symbols.

**Definition 1** (Bandwidth Adaptive Regenerating Code)**.** *Consider parameters $\alpha$, $n$, $k$, $\delta$, a set $D = \{d_1, \cdots, d_\delta\}$, with $d_1 < \cdots < d_\delta$, and a* total repair bandwidth function $\gamma : D \to [\alpha, \infty)$*. A bandwidth adaptive regenerating code $\mathcal{C}(n, k, D, \alpha, \gamma)$ is a regenerating code with sub-packetization level $\alpha$, such that following mechanisms are guaranteed.*

- Repair: *In each repair process the number of helpers, $d$, can be chosen arbitrarily from the set $D$. The choice of helper nodes is also arbitrary, and each of the chosen helpers then provides $\beta(d) = \gamma(d)/d$ repair symbols.*
- Data Reconstruction: *The data collector recovers the whole source data by accessing any arbitrary set of $k$ nodes.*

Note that the flexibility of the repair procedure depends on the parameter $\delta$, such that for a larger $\delta$, there are more options to select the number of helpers. In general, it is appealing to have small choices such as $d_1$, to guarantee the capability of code to perform repair when the number of available helpers is small, and also large choices such as $d_\delta$, to provide the capability of reducing the total as well as per-node repair bandwidth, and hence the transmission delay, whenever a larger number of helpers are available. The coding scheme we present in this work allows to design such a range for the elements in $D$.

**Definition 2** (Total Storage Capacity)**.** *For a set of parameters $\alpha$, $n$, $k$, $\delta$, a set $D = \{d_1, \cdots, d_\delta\}$, and a given function $\gamma : D \to [\alpha, \infty)$, the* total storage capacity *of a bandwidth adaptive regenerating code, $\mathcal{C}(n, k, D, \alpha, \gamma)$, is the maximum size of a file that could be stored in a network of $n$ storage nodes with sub-packetization level $\alpha$, using a bandwidth adaptive regenerating code $\mathcal{C}(n, k, D, \alpha, \gamma)$. We will denote the storage capacity of such a system by $F(n, k, D, \alpha, \gamma)$, or simply $F$, when the parameters could be inferred from the context.*

**Definition 3** (Bandwidth Adaptive MSR Codes, and Flexibility Degree)**.** *For any choice of parameters $\alpha$, $n$, $k$, $\delta$, and set $D = \{d_1, \cdots, d_\delta\}$, the bandwidth adaptive regenerating codes that realize both the MDS property defined by*

$$F(n, k, D, \alpha, \gamma) = k\alpha, \tag{2}$$

*as well as the the MSR characteristic equation simultaneously for all $d \in D$, given as*

$$\alpha = (d - k + 1)\beta(d), \quad \forall d \in D, \tag{3}$$

*are referred to as* bandwidth adaptive MSR *codes. Moreover, the number of elements in the set $D$ is referred to as* flexibility degree *of the code, and is denoted by $\delta$.*

### B. Main Results

The main contribution of this work is to provide a bandwidth adaptive MSR coding scheme with small sub-packetization level, and field size requirement. This coding scheme also guarantees exact repair of any failed node with many different choices of the number of helpers. This result is formally stated in the following theorem. In this paper $\mathrm{lcm}()$ denotes the least common multiple.

**Theorem 1.** *For arbitrary positive integers $n$, $k$, and $\delta$, there exists an adaptive bandwidth MSR code over a Galois field $\mathbb{F}_q$ of size $q \geq n$, with sub-packetization level $\alpha$ and total storage capacity $F$, satisfying*

$$\alpha = (k-1)\mathrm{lcm}\left(1, 2, \cdots, \delta\right), \qquad F = k\alpha, \tag{4}$$

*which is capable of performing exact-repair using any arbitrary $d_i$ helpers, for*

$$d_i = (i+1)(k-1), \ i \in \{1, \cdots, \delta\},$$

*and simultaneously satisfies the MSR characteristic equation (3) for any $d_i$.* i.e.,

$$\beta(d_i) = \frac{\alpha}{(d_i - k + 1)}, \ i \in \{1, \cdots, \delta\}.$$

We provide a constructive proof for this theorem based on the bandwidth adaptive MSR coding scheme that could be designed to achieve any arbitrary finite flexibility degree, introduced in Section IV.

## III. Related Works

The property of bandwidth adaptivity has been of interest in the literature of regenerating codes for the last few years. There has been a number of researchers who have addressed this problem under different settings. Some, such as [24], [25], [28], have considered functional-repair, in which code design problem reduces to linear network coding, and mainly focused on the theoretical limitations and properties of a bandwidth adaptive regenerating codes. There are, however, a few other works which have considered exact-repair as well [2], [7], [27]. In this section we briefly review these works and their relevance to the problem setup introduced in the previous section.

In the MBR case, [27], [29] addressed a similar setup. For given integers $d_{\min},\ d_{\max}$, authors presented a bandwidth adaptive exact-repair MBR regenerating code with $D = \{d_{\min}, d_{\min}+1, \cdots, d_{\max}\}$, which is simultaneously error resilient against up to a certain number of erroneous nodes in the network. Moreover, [29] also addresses the non-symmetric bandwidth adaptive exact-repair, in which different helpers may participate non-equally in the repair procedure, according to their available resources.

For the case of MSR, in [28] Wang *et. al.* considered a functional-repair coding scheme which supports bandwidth adaptivity. Later [24] also considered a similar setup, while in both these works the main focus is on derivation of the optimal trade-off between the storage overhead and repair bandwidth for the functional-repair in a *coordinated* setting, where more than one node failure is considered to be repaired together. Note, however, that none of these works address the exact-repair with bandwidth adaptivity in regenerating codes.

Aggrawal *et. al.* [25] also considered a functional-repair setup to achieve bandwidth adaptivity in regenerating codes. They analysed the mean-time-to-failure (MTTF) in the regenerating codes with and without bandwidth adaptivity. Their analysis is based on a birth-death process model in which the population of available storage node randomly changes with appropriately chosen rates. They showed that bandwidth adaptivity provides a significant gain in terms of MTTF.

When considering exact-repair regenerating codes, Cadambe *et. al.* were the first to address the bandwidth adaptivity [7]. They present an interference alignment based regenerating coding scheme which is capable of performing exact-repair with bandwidth adaptivity in the repair procedure. The code presented by Cadambe *et. al.* is the first exact-repair regenerating code with bandwidth adaptivity, however, their coding scheme only asymptotically achieves the MSR optimality, when $\alpha$ and $\beta$ tend to infinity with proper ratio. The importance of this result, however, is to show that bandwidth adaptivity could be implemented without extra cost in the optimal trade-off between the storage overhear and repair bandwidth, at least for the MSR codes, even when the exact-repair is required.

In [26], a similar setup, referred to as *progressive engagement*, is considered for regenerating codes with bandwidth adaptive exact-repair, and two coding schemes are provided. While the two coding schemes both preserve the MDS property, none of them satisfy the MSR characteristic equation simultaneously for different choices of $d$. Another main difference between the progressive engagement setup and the one considered in this work is that the authors in [26] relax the property that *any* subset of surviving nodes could be considered as helpers by assuming that all the remaining systematic nodes are always participating as helpers. Moreover, they require the set of available choices for the number of helpers, namely $D$, to be $D = \{k, k+1, \cdots, n-1\}$, while in our formulation, $D$ does not need to contain all integers between $k$ and $n-1$. Indeed, we will provide a different view point of our result in Section IV-D, which shows our formulation could indeed consider $D$ to be arbitrary.

The first two explicit constructions for exact-repair MSR codes with bandwidth adaptivity with finite sub-packetization level were introduced in [2]. Both of these constructions work for any arbitrary values of $k$ and $n$, and the set of choices for the number of helpers, $D$, could be designed to contain any value $d_i$ such that $k < d_i < n$. However, the required value for the sub-packetization level, $\alpha$, in both constructions is still considerably huge. In particular, for a DSS with $n$ storage nodes, and the set $D = \{d_1, \cdots, d_\delta\}$, constructions in [2] require a field size of $(n-k)n$ and

$$\alpha = \left[\mathrm{lcm}\left(d_1 - k + 1, \cdots, d_\delta - k + 1\right)\right]^n. \tag{5}$$

Therefore, these constructions only achieve optimality for storage of contents which are exponentially large in terms of the number of storage nodes in the system, which is still impractical for common parameters in practice. For instance, the coding schemes suggested for Facebook and Windows Azure have $n = 14$ [30], and $n = 10$ [31] respectively, while other industrial solutions even report hundreds of nodes in their DSS [21]. For such schemes the smallest realization of constructions in [2] with bandwidth adaptivity require sub-packetization levels of order $10^{20}$ or higher[1], hence, leaving the problem of exact-repair bandwidth adaptive regenerating code design yet open for practical parameters.

In this work, as presented in Theorem 1, we address the exact-repair bandwidth adaptive MSR code design problem with small subpacketization level, following the *product matrix* (PM) framework introduced in [6]. Comparing (5) with (4), one could see that the presented scheme reduces the required $\alpha$ (and $\beta$) values exponentially. However, this scheme works only for $2k - 2 \leq d_i,\ \forall d_i \in D$. As a result, the design of high-rate bandwidth adaptive MSR codes with small $\alpha$ and $\beta$ still remains an open problem.

[1]Considering the required field size for constructions in [2] this is equivalent to per-node storage capacity of order at least tens of peta-bytes.

## IV. Coding Scheme

In this section we introduce a bandwidth adaptive exact-repair MSR coding scheme, which could be designed to provide any required flexibility degree with elements of the set $D$ evenly located between the smallest and the largest element, namely $d_1$ and $d_\delta$. We describe the encoding and decoding schemes for storage, repair, and data reconstruction procedures for the adaptive bandwidth MSR code in the following subsections. This coding scheme is closely related to the product matrix MSR code introduced in [6], and could be considered as an extension of the product matrix code that achieves bandwidth adaptivity. As mentioned in the previous section, we will assume all the source and encoded symbols, are elements of a Galois field of an appropriately large field size $q \geq n$, denoted by $\mathbb{F}_q$. Moreover, all the operations hereafter are considered to be filed operations of $\mathbb{F}_q$. We will refer to $\mathbb{F}_q$ as the *code alphabet*.

In the design of the proposed coding scheme, we chose a design parameter $\mu$, and the required flexibility degree $\delta$. All the other parameters of the code, including $\alpha$, $F$, $k$, $D = \{d_1, \cdots, d_\delta\}$, and $\beta(d_i)$ will be then determined based on $\mu$, and $\delta$ as follows. The sub-packetization level is

$$\alpha = \mu \cdot \mathrm{lcm}\,(1, 2, \cdots, \delta)\,. \tag{6}$$

Moreover, we have $k = \mu + 1$, and $F = k\alpha = (\mu+1)\alpha$, which satisfies the MDS property. Finally, for $D$ we have

$$D = \{d_1, \cdots, d_\delta\}, \quad d_i = (i+1)\mu, \ i \in \{1, \cdots, \delta\}. \tag{7}$$

and for any $d_i \in D$, the associated per-node and total repair bandwidths denoted by $\beta(d_i)$, and $\gamma(d_i)$ respectively are

$$\beta(d_i) = \frac{\alpha}{i\mu}, \qquad \gamma(d_i) = d_i\beta(d_i) = \frac{(i+1)\alpha}{i}. \tag{8}$$

### A. Coding for Storage

We begin the introduction of the coding scheme by describing the process of encoding the source symbols and deriving the encoded symbols to be stored in the storage nodes. Similar to the product matrix codes, the first step in encoding for storage in this scheme is to arrange the information symbols in a matrix, denoted by $M$, which we refer to hereafter as the *data matrix*. Let

$$z_\delta = \mathrm{lcm}\,(1, 2, \cdots, \delta)\,. \tag{9}$$

The data matrix in our coding scheme is structured as follows,

$$M = \begin{bmatrix} S_1 & S_2 & O & O & O & O & \cdots & O \\ S_2 & S_3 & S_4 & O & O & O & \cdots & O \\ O & S_4 & S_5 & S_6 & O & O & \cdots & O \\ O & O & S_6 & S_7 & S_8 & O & \cdots & O \\ \vdots & & & & & \ddots & & \vdots \\ O & \cdots & & & O & S_{2z_\delta-4} & S_{2z_\delta-3} & S_{2z_\delta-2} \\ O & \cdots & & & O & O & S_{2z_\delta-2} & S_{2z_\delta-1} \\ O & \cdots & & & O & O & O & S_{2z_\delta} \end{bmatrix}, \tag{10}$$

where, each $S_i,\ i \in \{1, \cdots, 2z_\delta\}$ is a symmetric $\mu \times \mu$ matrix filled with $\mu(\mu+1)/2$ source symbols, and $O$ is a $\mu \times \mu$ zero matrix. Therefore, $M$ has $(z_\delta + 1)\mu$ rows and $z_\delta\mu$ columns. Note that the total number of distinct source symbols in the data matrix $M$ is

$$F = \mu(\mu+1)z_\delta = k\alpha. \tag{11}$$

**Example 1.** Consider the design parameters $\mu = 2$, and $\delta = 2$. Therefore, from (9) we have $z_\delta = 2$, and using (11), the maximum number of source symbols that we can arrange in the data matrix $M$ is 12. Denoting the source symbols by $s_1, \cdots, s_{12}$, then we have

$$M = \begin{bmatrix} S_1 \\ S_2 \\ O \end{bmatrix} \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} s_1 & s_2 \\ s_2 & s_3 \end{bmatrix} \begin{bmatrix} s_4 & s_5 \\ s_5 & s_6 \end{bmatrix} \\ \begin{bmatrix} s_4 & s_5 \\ s_5 & s_6 \end{bmatrix} \begin{bmatrix} s_7 & s_8 \\ s_8 & s_9 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} s_{10} & s_{11} \\ s_{11} & s_{12} \end{bmatrix} \end{bmatrix}. \tag{12}$$

△

Once the data matrix is ready, the source encoder creates the vector of coded symbols for each of the $n$ storage nodes, by calculating the product of a node-specific coefficient vector and the data matrix. To describe this process, we first need the following definition.

**Definition 4.** *[Generalized Vandermonde Matrix] For distinct and non-zero elements $e_1, \cdots, e_m$ of $\mathbb{F}_q$, and some integer $c \geq 0$, a matrix $A_{m\times \ell}$ with entries*

$$A_{i,j} = e_i^{c+j-1}, \quad \textit{for } i \in \{1, \cdots, m\},\ j \in \{1, \cdots, \ell\},$$

*is referred to as a* generalized Vandermonde *matrix.*

The following lemma about these matrices will be used in the proofs of the following theorems,

**Lemma 1.** *Consider distinct and non-zero elements $e_1, \cdots, e_\ell$ in $\mathbb{F}_q$, and an integer $c \geq 0$. Then a square $\ell \times \ell$ generalized Vandermonde matrix, $A$, as defined in Definition 4, is invertible in $\mathbb{F}_q$.*

The proof of the above lemma, simply follows from the fact that,

$$A = \begin{bmatrix} e_1^c & 0 & \cdots & 0 \\ 0 & e_2^c & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & e_\ell^c \end{bmatrix} \begin{bmatrix} 1 & e_1 & e_1^2 & \cdots & e_1^{\ell-1} \\ 1 & e_2 & e_2^2 & \cdots & e_2^{\ell-1} \\ & & \vdots & & \\ 1 & e_\ell & e_\ell^2 & \cdots & e_\ell^{\ell-1} \end{bmatrix},$$

and the two matrices on the right hand side are both full-rank, as one of them is diagonal with non-zero diagonal elements and the other one is a square Vandermonde matrix.

Back to the description of our coding scheme, for distinct and non-zero elements $e_i$'s in $\mathbb{F}_q$, with $i \in \{1, \cdots, n\}$ we set $c = 0$, and define a generalized Vandermonde matrix of size $n \times (z_\delta + 1)\mu$ as

$$\Psi = \begin{bmatrix} 1 & e_1 & e_1^2 & \cdots & e_1^{(z_\delta+1)\mu-1} \\ 1 & e_2 & e_2^2 & \cdots & e_2^{(z_\delta+1)\mu-1} \\ & & \vdots & & \\ 1 & e_n & e_n^2 & \cdots & e_n^{(z_\delta+1)\mu-1} \end{bmatrix}.$$

Note that all of the submatrices of $\Psi$ are also generalized Vandermonde matrices. We refer to $\Psi$ as the *coefficient matrix* and denote the $j^{\text{th}}$ row of $\Psi$ by $\underline{\psi}_j$. The vector of encoded symbols to be stored on node $j,\ j \in \{1, \cdots, n\}$, denoted by $\underline{x}_j$, is calculated as

$$\underline{x}_j = \underline{\psi}_j M. \tag{13}$$

The vector $\underline{\psi}_j$ is the node-specific coefficient vector for storage node $j$. Note that the per-node storage capacity requirement for this coding scheme is then $z_\delta \mu$ as given by (6).

**Example 2.** Let's consider the setting in Example 1 again, and assume we have $n = 7$ nodes in the network. Assume that the code alphabet is the Galois field $\mathbb{F}_{11}$. The coefficient matrix could be formed based on $(e_1, e_2, \cdots, e_7) = (1, 2, \cdots, 7)$ as follows,

$$\Psi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 5 & 10 \\ 1 & 3 & 9 & 5 & 4 & 1 \\ 1 & 4 & 5 & 9 & 3 & 1 \\ 1 & 5 & 3 & 4 & 9 & 1 \\ 1 & 6 & 3 & 7 & 9 & 10 \\ 1 & 7 & 5 & 2 & 3 & 10 \end{bmatrix}. \tag{14}$$

The encoded content of nodes 1 to 7 can be calculated using (12) and (13). For instance, for node 1 we have

$$\underline{x}_1 = [x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}] = [1, 1, 1, 1, 1, 1]M,$$

which gives

$$\begin{aligned} x_{1,1} &= s_1 + s_2 + s_4 + s_5, \\ x_{1,2} &= s_2 + s_3 + s_5 + s_6, \\ x_{1,3} &= s_4 + s_5 + s_7 + s_8 + s_{10} + s_{11}, \\ x_{1,4} &= s_5 + s_6 + s_8 + s_9 + s_{11} + s_{12}, \end{aligned}$$

and similarly for node 7 we have

$$\underline{x}_7 = [x_{7,1}, x_{7,2}, x_{7,3}, x_{7,4}] = [1, 7, 5, 2, 3, 10]M,$$

which gives

$$\begin{aligned}
x_{7,1} &= s_1 + 7s_2 + 5s_4 + 2s_5,\\
x_{7,2} &= s_2 + 7s_3 + 5s_5 + 2s_6,\\
x_{7,3} &= s_4 + 7s_5 + 5s_7 + 2s_8 + 3s_{10} + 10s_{11},\\
x_{7,4} &= s_5 + 7s_6 + 5s_8 + 2s_9 + 3s_{11} + 10s_{12}.
\end{aligned}$$

△

### *B. Data Reconstruction*

In order to reconstruct all the information stored in the system, the data collector accesses $k$ arbitrary nodes in the network and downloads all their contents. To describe the details of the decoding we use the following lemma.

**Lemma 2.** *Let $X$ and $\Phi$ be two known matrices of size $(\mu+1)\times\mu$, such that $\Phi$ is a generalized Vandermonde matrix, and assume $\Delta$ is a known diagonal matrix of size $(\mu+1)\times(\mu+1)$, with distinct and non-zero diagonal elements. Then the equation*

$$X = \Phi A + \Delta\Phi B, \tag{15}$$

*is uniquely solvable for unknown $\mu\times\mu$ symmetric matrices $A$ and $B$.*

The proof of this lemma is based on the data reconstruction scheme of the product matrix MSR codes, introduced in [6], and is provided in Appendix A to help keeping this paper self-contained. The following theorem explains the data reconstruction procedure in this coding scheme.

**Theorem 2.** *For the coding scheme presented in subsection IV-A, there exists a decoding scheme to reconstruct all the source symbols arranged in the data matrix $M$ from the encoded content of any arbitrary group of $k=\mu+1$ storage nodes.*

*Proof.* Let's assume the set of accessed nodes is $\{\ell_1,\cdots,\ell_k\}$. Moreover, let's denote the $k\times(z_\delta+1)\mu$ submatrix of $\Psi$ associated with the nodes $\ell_1,\cdots,\ell_k$, by $\Psi_{\text{DC}}$. We will further denote the submatrix of $\Psi_{\text{DC}}$ consisting of columns $(i-1)\mu+1$ through $i\mu$, by $\Psi_{\text{DC}}(i)$. In other words, we have a partitioning of $\Psi_{\text{DC}}$'s columns as

$$\Psi_{\text{DC}} = \begin{bmatrix} \underline{\psi}_{\ell_1} \\ \vdots \\ \underline{\psi}_{\ell_k} \end{bmatrix} = [\Psi_{\text{DC}}(1),\cdots,\Psi_{\text{DC}}(z_\delta+1)]. \tag{16}$$

As a result, defining the diagonal matrix

$$\Lambda_{\text{DC}} = \begin{bmatrix} e_{\ell_1}^{\mu} & 0 & 0 & \cdots & 0 \\ 0 & e_{\ell_2}^{\mu} & 0 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & e_{\ell_k}^{\mu} \end{bmatrix},$$

for each $k\times\mu$ submatrix $\Psi_{\text{DC}}(i)$ we have

$$\Psi_{\text{DC}}(i+1) = \Lambda_{\text{DC}}\Psi_{\text{DC}}(i). \tag{17}$$

Similarly, let's denote the matrix consisting of the collected encoded vectors by $X_{\text{DC}}$, and its partitioning to $k\times\mu$ submatrices $X_{\text{DC}}(i)$, $i\in\{1,\cdots,z_\delta\}$ as follows

$$X_{\text{DC}} = \begin{bmatrix} \underline{x}_{\ell_1} \\ \vdots \\ \underline{x}_{\ell_k} \end{bmatrix} = [X_{\text{DC}}(1),\cdots,X_{\text{DC}}(z_\delta)]. \tag{18}$$

**Example 3.** Following the setting described in Example 1 and Example 2, we have $k=3$, and the coefficient matrix $\Psi$ is given in (14). Let's assume the data collector accesses the storage nodes 1, 2, and 4. Then, with $z_\delta=2$, we have,

$$\Psi_{\text{DC}} = [\Psi_{\text{DC}}(1), \Psi_{\text{DC}}(2), \Psi_{\text{DC}}(3)] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 5 & 10 \\ 1 & 4 & 5 & 9 & 3 & 1 \end{bmatrix},$$

and,

$$\Psi_{\text{DC}}(1)=\begin{bmatrix}1&1\\1&2\\1&4\end{bmatrix},\Psi_{\text{DC}}(2)=\begin{bmatrix}1&1\\4&8\\5&9\end{bmatrix},\Psi_{\text{DC}}(3)=\begin{bmatrix}1&1\\5&10\\3&1\end{bmatrix}.$$

Moreover, with $\mu=2$ we have

$$\Lambda_{\text{DC}}=\begin{bmatrix}1^2&0&0\\0&2^2&0\\0&0&4^2\end{bmatrix}=\begin{bmatrix}1&0&0\\0&4&0\\0&0&5\end{bmatrix}, \tag{19}$$

and

$$\begin{aligned}X_{\text{DC}}&=[X_{\text{DC}}(1),X_{\text{DC}}(2)]\\&=\begin{bmatrix}\underline{x}_1\\\underline{x}_2\\\underline{x}_4\end{bmatrix}=\begin{bmatrix}x_{1,1}&x_{1,2}&x_{1,3}&x_{1,4}\\x_{2,1}&x_{2,2}&x_{2,3}&x_{2,4}\\x_{4,1}&x_{4,2}&x_{4,3}&x_{4,4}\end{bmatrix},\end{aligned}$$

and finally,

$$X_{\text{DC}}(1)=\begin{bmatrix}x_{1,1}&x_{1,2}\\x_{2,1}&x_{2,2}\\x_{4,1}&x_{4,2}\end{bmatrix},X_{\text{DC}}(2)=\begin{bmatrix}x_{1,3}&x_{1,4}\\x_{2,3}&x_{2,4}\\x_{4,3}&x_{4,4}\end{bmatrix}. \tag{20}$$

△

The decoding procedure for data reconstruction consists of $z_\delta$ consecutive steps. The first step uses only the submatrix $X_{\text{DC}}(1)$. Using (17) we have,

$$\begin{aligned}X_{\text{DC}}(1)&=[\Psi_{\text{DC}}(1),\Psi_{\text{DC}}(2)]\begin{bmatrix}S_1\\S_2\end{bmatrix}\\&=\Psi_{\text{DC}}(1)S_1+\Lambda_{\text{DC}}\Psi_{\text{DC}}(1)S_2.\end{aligned}$$

Using Lemma 2, the decoder recovers both $S_1$, and $S_2$, using $X_{\text{DC}}(1)$, in step one. Then, for $i\in\{2,\cdots,z_\delta\}$, the decoder performs step $i$ by using submatrix $X_{\text{DC}}(i)$, and decodes submatrices $S_{2i-1}$, and $S_{2i}$, as follows.

In step $i,\ i\in\{2,\cdots,z_\delta\}$, of the data reconstruction decoding, the decoder uses the submatrix $X_{\text{DC}}(i)$. Note that

$$\begin{aligned}X_{\text{DC}}(i)&=[\Psi_{\text{DC}}(i-1),\Psi_{\text{DC}}(i),\Psi_{\text{DC}}(i+1)]\begin{bmatrix}S_{2(i-1)}\\S_{2i-1}\\S_{2i}\end{bmatrix}\\&=\Psi_{\text{DC}}(i-1)S_{2(i-1)}+[\Psi_{\text{DC}}(i),\Psi_{\text{DC}}(i+1)]\begin{bmatrix}S_{2i-1}\\S_{2i}\end{bmatrix}.\end{aligned}$$

Having the submatrix $S_{2(i-1)}$ already recovered from step $i-1$, the decoder first calculates

$$\begin{aligned}\hat{X}_{\text{DC}}(i)&=X_{\text{DC}}(i)-\Psi_{\text{DC}}(i-1)S_{2(i-1)}.\\&=[\Psi_{\text{DC}}(i),\Psi_{\text{DC}}(i+1)]\begin{bmatrix}S_{2i-1}\\S_{2i}\end{bmatrix}.\end{aligned} \tag{21}$$

Then from (17), and (21), we have

$$\hat{X}_{\text{DC}}(i)=\Psi_{\text{DC}}(i)S_{2i-1}+\Lambda_{\text{DC}}\Psi_{\text{DC}}(i)S_{2i}.$$

Again using Lemma 2, decoder recovers $S_{2i-1}$, and $S_{2i}$ at the end of the step $i$ of the decoding. Therefore, by finishing step $z_\delta$, the decoder reconstructs all the submatrices in the data matrix $M$, and recovers the whole data stored in the network. □

**Example 4.** Following the setting of Examples 1 to 3, consider $\mu=2$, $k=3$, and the data matrix as provided in (12). The first step of decoding in the data reconstruction process based on the encoded data stored in nodes 1, 2 and 4, starts by using only $X_{\text{DC}}(1)$, as given in (20). Using Lemma 2, The decoder then recovers $S_1$, and $S_2$ submatrices of the data matrix $M$.

In the second step then the decoder first calculates

$$\hat{X}_{\text{DC}}(2)=X_{\text{DC}}(2)-\Psi_{\text{DC}}(1)S_2,$$

which is equal to

$$\hat{X}_{\text{DC}}(2)=\Psi_{\text{DC}}(2)S_3+\Lambda_{\text{DC}}\Psi_{\text{DC}}(2)S_4,$$

and hence is of the desired form of (15). Therefore, again using lemma 2, the decoder recovers $S_3$, and $S_4$, which completes the data reconstruction. △

Algorithm IV-B summarizes the data reconstruction mechanism in this coding scheme.

**Algorithm 1** Data Reconstruction

Input: $\mu$, $z_\delta$, $\underline{x}_{\ell_i}$, $\underline{\psi}_{\ell_i}$, and $e_{\ell_i}$ for $i \in \{1, \cdots, k\}$.
Output: Submatrices $S_1, \cdots, S_{2z_\delta}$.
Form matrices $\Psi_{\text{DC}}(i)$, for $i \in \{1, \cdots, z_\delta + 1\}$, using (16).
Form matrices $X_{\text{DC}}(i)$, for $i \in \{1, \cdots, z_\delta\}$, using (18).
Form matrix $\Lambda_{\text{DC}}$, using (19).
Recover submatrices $S_1$, $S_2$ from $X_{\text{DC}}(1)$, $\Psi_{\text{DC}}(1)$, and $\Lambda_{\text{DC}}$, using Lemma 2.
**for** $i = 2$ to $z_\delta$ **do**
    Calculate $\hat{X}_{\text{DC}}(i) = X_{\text{DC}}(i) - \Psi_{\text{DC}}(i-1) S_{2(i-1)}$.
    Recover submatrices $S_{2i-1}$, $S_{2i}$ from $\hat{X}_{\text{DC}}(i)$, $\Psi_{\text{DC}}(i)$, and $\Lambda_{\text{DC}}$, using Lemma 2.
**end for**

### C. Bandwidth Adaptive Exact-Repair

We now describe the bandwidth adaptive repair procedure, by assuming that node $f$ is failed and the set of helpers selected for the repair are $\mathcal{H} = \{h_1, \cdots, h_d\}$, for some arbitrary $d \in D$. The following theorem describes the repair procedure in this bandwidth adaptive MSR code.

**Theorem 3.** *Consider the coding scheme presented in subsection IV-A, with design parameters $\mu$, and $\delta$, and $D$ as defined in (7). For any arbitrary failed node $f$, and any arbitrary set of helpers $\mathcal{H} = \{h_1, \cdots, h_d\}$, for some $d \in D$, there exists a repair scheme for recovering the content of node $f$ with per-node repair bandwidth,*

$$\beta(d) = \frac{\alpha}{d - \mu}. \tag{22}$$

**Remark 1.** *Note that (7) and (22) are consistent with (8), which satisfies the MSR characteristic equation (3) for any $d \in D$.*

*Proof.* Without loss of generality let $d = (m+1)\mu$, for some $m \in \{1, \cdots, \delta\}$. Note that (6), and (7) guarantee that for any $d \in D$, $\alpha$ is an integer multiple of $d - \mu$, hence $\beta(d)$ is an integer. Each helper node $h \in \mathcal{H}$, creates $\beta(d) = \alpha/(d-\mu)$ repair symbols to repair node $f$ as follows. First helper node $h$ partitions its encoded content into $\beta(d)$ equal segments, such that for $i \in \{1, \cdots, \beta(d)\}$, the segment $\underline{x}_h(i)$ is of size $\alpha/\beta(d) = d - \mu = m\mu$, and contains elements $x_{h,(i-1)m\mu+1}$ through $x_{h,im\mu}$. Then we have

$$\underline{x}_h = [\underline{x}_h(1), \cdots, \underline{x}_h(\beta(d))]. \tag{23}$$

Similarly, for any node $\ell$, we split the first $\alpha$ entries of a coefficient vector assigned to node $\ell$, namely $\underline{\psi}_\ell$, into $\beta(d)$ equal segments as

$$\underline{\psi}_\ell(1:\alpha) = \left[\underline{\psi}_\ell(1), \cdots, \underline{\psi}_\ell(\beta(d))\right], \tag{24}$$

where each segment $\underline{\psi}_\ell(i)$ is of size $d - \mu = m\mu$.

Now each helper node $h \in \mathcal{H}$, creates its $\beta(d)$ repair symbols as

$$\begin{aligned} \underline{r}(h, f) &= \left[r_1(h,f), \cdots, r_{\beta(d)}(h,f)\right] \\ &= \left[\underline{x}_h(1) \left(\underline{\psi}_f(1)\right)^{\mathsf{T}}, \cdots, \underline{x}_h(\beta(d)) \left(\underline{\psi}_f(\beta(d))\right)^{\mathsf{T}}\right]. \end{aligned} \tag{25}$$

The repair decoder then stacks $d$ repair vectors $\underline{r}(h, f)$, for $h \in \mathcal{H}$, into a $d \times \beta(d)$ matrix

$$\Upsilon_{\mathcal{H}} = \begin{bmatrix} \underline{r}(\ell_1, f) \\ \vdots \\ \underline{r}(\ell_d, f) \end{bmatrix}. \tag{26}$$

We then introduce the following partitioning of the matrix $\Upsilon_{\mathcal{H}}$, into $\beta(d)$ submatrices, as follows

$$\Upsilon_{\mathcal{H}} = [\Upsilon_{\mathcal{H}}(1), \cdots, \Upsilon_{\mathcal{H}}(\beta(d))], \tag{27}$$

where $\Upsilon_{\mathcal{H}}(i)$, $i \in \{1, \cdots, \beta\}$ is the $i^{\text{th}}$ column in $\Upsilon_{\mathcal{H}}$, of size $d \times 1$.

Before starting to describe the repair decoding procedure, we need to introduce some notations associated to a given repair scenario. Consider a repair procedure with $d = (m+1)\mu$. For the corresponding $\beta(d) = \alpha/(d-\mu)$, we will partition matrix $M$ as depicted in Fig. 1, and equation (28). Note that this results in $\beta(d)$ non-overlapping diagonal submatices $M_i$, $i \in \{1, \cdots, \beta(d)\}$, each of size $m\mu \times m\mu$, along with $\mu \times \mu$ symmetric submatrices $S_{2m}, S_{4m}, \cdots, S_{2\beta(d)m} = S_{2z_\delta}$ as shown in the figure. It is worth mentioning that the general pattern of the partitioning shown in figure 1 is preserved the same for all $d \in D$, and only the size, and number of the $M_i$ diagonal blocks changes for different choices of the parameter $d$. From

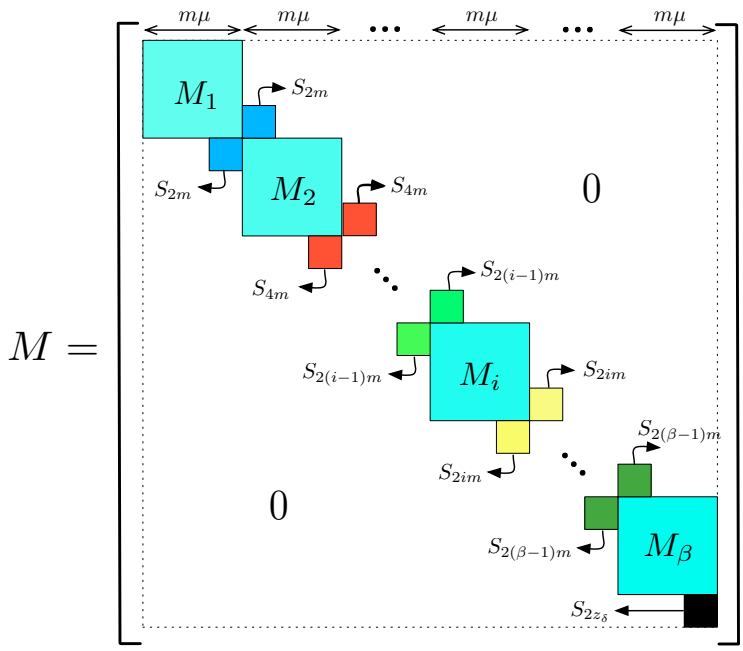


Fig. 1. In the above figure $\beta$ represents $\beta(d)$, for some chosen $d \in D$, such that, $d = (m+1)\mu$. Moreover, the white area is filled by zeros, and each coloured squares represent a non-zero symmetric submatrix of $M$.

$$M_i = \begin{bmatrix} S_{2(i-1)m+1} & S_{2(i-1)m+2} & O & O & O & \cdots & O \\ S_{2(i-1)m+2} & S_{2(i-1)m+3} & S_{2(i-1)m+4} & O & O & \cdots & O \\ O & S_{2(i-1)m+4} & S_{2(i-1)m+5} & S_{2(i-1)m+6} & O & \cdots & O \\ \vdots & & & & \ddots & & \vdots \\ O & \cdots & & O & S_{2im-4} & S_{2im-3} & S_{2im-2} \\ O & \cdots & & O & O & S_{2im-2} & S_{2im-1} \end{bmatrix}. \tag{28}$$

the construction of the data matrix, introduced in (10), each $M_i$ submatrix will be symmetric. As a result, the data matrix $M$ could be interpreted in terms of the submatrices $M_i$, and $S_{2i}$ for $i \in \{1, \cdots, \beta(d)\}$, associated to a repair procedure with $d = (m+1)\mu$, $d \in D$.

Finally the last notations we use to describe the adaptive repair decoding scheme, using a given set of helpers $\mathcal{H} = \{h_1, \cdots, h_d\}$, is,

$$\Omega_{\mathcal{H}}(i) = \begin{bmatrix} e_{h_1}^{(i-1)m\mu} & e_{h_1}^{(i-1)m\mu+1} & \cdots & e_{h_1}^{(im+1)\mu-1} \\ & & \vdots & \\ e_{h_d}^{(i-1)m\mu} & e_{h_d}^{(i-1)m\mu+1} & \cdots & e_{h_d}^{(im+1)\mu-1} \end{bmatrix}, \quad i \in \{1, \cdots, \beta(d)\}. \tag{29}$$

Note that, $\Omega_{\mathcal{H}}(i),\ i \in \{1, \cdots, \beta(d)\}$, is a $d \times d$ generalized Vandermonde matrix and hence is invertible as shown in Lemma 1. We denote the upper $(d-\mu) \times d$ submatrix of $(\Omega_{\mathcal{H}}(i))^{-1}$ by $\Theta_{\mathcal{H}}(i)$, and the lower $\mu \times d$ submatrix by $\Xi_{\mathcal{H}}(i)$. Then we have,

$$(\Omega_{\mathcal{H}}(i))^{-1} = \begin{bmatrix} \Theta_{\mathcal{H}}(i) \\ \Xi_{\mathcal{H}}(i) \end{bmatrix}. \tag{30}$$

The decoding procedure for the repair of node $f$ is performed in $\beta(d)$ sequential steps as will be described in the following. Let's begin with the first step.

For the failed node $f$, let $\underline{\phi}_f$ denote the $1 \times \mu$ vector ,

$$\underline{\phi}_f = \begin{bmatrix} 1 \ , \ e_f \ , \ \cdots \ , \ e_f^{\mu-1} \end{bmatrix}. \tag{31}$$

Using a partitioning similar to (23) for $\underline{x}_f$, then we have,

$$\underline{x}_f(1) = \underline{\psi}_f \begin{bmatrix} \begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix} \\ O \quad \cdots \quad O \quad S_{2m} \end{bmatrix} = \underline{\psi}_f(1) M_1 + \left[O, \cdots, O, e_f^{m\mu} \underline{\phi}_f S_{2m}\right]_{\mu \times m\mu}. \tag{32}$$

In the first step, the decoder recovers the two terms on the right in the above equation to reconstruct $\underline{x}_f(1)$ using only the first repair symbol received from each of the helpers, namely $r_1(h_i, f)$, for $i \in \{1, \cdots, d\}$, as follows.

Using (23) to (27), and the partitioning denoted in Fig. 1, the submatrix $\Upsilon_{\mathcal{H}}(1)$, introduced in (27) can be written as,

$$\Upsilon_{\mathcal{H}}(1) = \begin{bmatrix} \underline{x}_{h_1}(1) \left(\underline{\psi}_f(1)\right)^{\intercal} \\ \vdots \\ \underline{x}_{h_d}(1) \left(\underline{\psi}_f(1)\right)^{\intercal} \end{bmatrix} = \begin{bmatrix} \underline{x}_{h_1}(1) \\ \vdots \\ \underline{x}_{h_d}(1) \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal}$$

$$= \begin{bmatrix} 1 & e_{h_1} & e_{h_1}^2 & \cdots & e_{h_1}^{(m+1)\mu-1} \\ & & \vdots & & \\ 1 & e_{h_d} & e_{h_d}^2 & \cdots & e_{h_d}^{(m+1)\mu-1} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix} \\ O \quad \cdots \quad O \quad S_{2m} \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal}. \tag{33}$$

Then using (29), we have,

$$\Upsilon_{\mathcal{H}}(1) = \Omega_{\mathcal{H}}(1) \begin{bmatrix} \begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix} \\ O \quad \cdots \quad O \quad S_{2m} \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal}. \tag{34}$$

Multiplying the inverse of $\Omega_{\mathcal{H}}(1)$ from right to the both sides of (34), and using (30) the decoder derives

$$\begin{bmatrix} \begin{bmatrix} & & \\ & M_1 & \\ & & \end{bmatrix} \\ O \quad \cdots \quad O \quad S_{2m} \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal} = \begin{bmatrix} \Theta_{\mathcal{H}}(i) \\ \Xi_{\mathcal{H}}(i) \end{bmatrix} \Upsilon_{\mathcal{H}}(1)$$

That gives,

$$M_1 \left(\underline{\psi}_f(1)\right)^{\intercal} = \Theta_{\mathcal{H}}(1) \Upsilon_{\mathcal{H}}(1), \tag{35}$$

and similarly, using (31),

$$S_{2m} \left(e_f^{(m-1)\mu} \underline{\phi}_f\right)^{\intercal} = \Xi_{\mathcal{H}}(1) \Upsilon_{\mathcal{H}}(1). \tag{36}$$

Since both $M_1$, and $S_{2m}$ are symmetric, from (35) we have,

$$\underline{\psi}_f(1) M_1 = \left(\Theta_{\mathcal{H}}(1) \Upsilon_{\mathcal{H}}(1)\right)^{\intercal}, \tag{37}$$

and from (36), by multiplying the scalar $e_f^{\mu}$, we get

$$e_f^{m\mu} \underline{\phi}_f S_{2m} = e_f^{\mu} \left(\Xi_{\mathcal{H}}(1) \Upsilon_{\mathcal{H}}(1)\right)^{\intercal}. \tag{38}$$

From (37), and (38), and using (32) the decoder then recovers $\underline{x}_f(1)$ as,

$$\underline{x}_f(1) = \underline{\psi}_f(1) M_1 + \left[O, \cdots, O, e_f^{m\mu} \underline{\phi}_f S_{2m}\right]_{\mu \times m\mu}, \tag{39}$$

where, the rightmost term in the above expression is derived by padding $m-1$, $\mu \times \mu$ zero matrices, $O$, to the left of the matrix calculated in (38).

In step $i$ for $i=2$ through $\beta(d)$ of the repair decoding, the decoder then recovers $\underline{x}_f(i)$, using $\Upsilon_{\mathcal{H}}(i)$ received from the helpers, along with $e_f^{(i-1)m\mu}\underline{\phi}_f S_{2(i-1)m}$, recovered from the step $i-1$ of decoding. To this end, first note that similar to (33) the repair symbols in $\Upsilon_{\mathcal{H}}(i)$ can be written as,

$$\Upsilon_{\mathcal{H}}(i)=\begin{bmatrix}\underline{x}_{h_1}(i)\left(\underline{\psi}_f(i)\right)^{\intercal}\\ \vdots\\ \underline{x}_{h_d}(i)\left(\underline{\psi}_f(i)\right)^{\intercal}\end{bmatrix}=\begin{bmatrix}\underline{x}_{h_1}(i)\\ \vdots\\ \underline{x}_{h_d}(i)\end{bmatrix}\left(\underline{\psi}_f(i)\right)^{\intercal}$$
$$=\begin{bmatrix}e_{h_1}^{(i-1)m\mu} & e_{h_1}^{(i-1)m\mu+1} & \cdots & e_{h_1}^{(im+1)\mu-1}\\ & & \vdots & \\ e_{h_d}^{(i-1)m\mu} & e_{h_d}^{(i-1)m\mu+1} & \cdots & e_{h_d}^{(im+1)\mu-1}\end{bmatrix}\begin{bmatrix}S_{2(i-1)m} & O & \cdots & O\\ \begin{bmatrix} & & \\ & M_i & \\ & & \end{bmatrix} & & & \\ O & \cdots & O & S_{2m}\end{bmatrix}\left(\underline{\psi}_f(i)\right)^{\intercal}.$$

Using (29) we can rewrite the above equation as,

$$\Upsilon_{\mathcal{H}}(i)=\begin{bmatrix}e_{\ell_1}^{(i-1)m\mu-\mu}\underline{\phi}_{\ell_1}\\ \vdots\\ e_{\ell_d}^{(i-1)m\mu-\mu}\underline{\phi}_{\ell_d}\end{bmatrix}S_{2(i-1)m}\left(e_f^{(i-1)m\mu}\underline{\phi}_f\right)^{\intercal}+\Omega_{\mathcal{H}}(i)\begin{bmatrix}\begin{bmatrix} & & \\ & M_i & \\ & & \end{bmatrix}\\ O \quad \cdots \quad O \quad S_{2im}\end{bmatrix}\left(\underline{\psi}_f(i)\right)^{\intercal}.$$

The decoder first removes the contribution of the $S_{2(i-1)m}$ submatrix in the repair symbols in $\Upsilon_{\mathcal{H}}(i)$ by calculating

$$\hat{\Upsilon}_{\mathcal{H}}(i)=\Upsilon_{\mathcal{H}}(i)-\begin{bmatrix}e_{\ell_1}^{(i-1)m\mu-\mu}\underline{\phi}_{\ell_1}\\ \vdots\\ e_{\ell_d}^{(i-1)m\mu-\mu}\underline{\phi}_{\ell_d}\end{bmatrix}S_{2(i-1)m}\left(e_f^{(i-1)m\mu}\underline{\phi}_f\right)^{\intercal}. \tag{40}$$

In the above expression, $S_{2(i-1)m}\left(e_f^{(i-1)m\mu}\underline{\phi}_f\right)^{\intercal}$ is itself derived by transposing $e_f^{(i-1)m\mu}\underline{\phi}_f S_{2(i-1)m}$. As a result we have,

$$\hat{\Upsilon}_{\mathcal{H}}(i)=\Omega_{\mathcal{H}}(i)\begin{bmatrix}\begin{bmatrix} & & \\ & M_i & \\ & & \end{bmatrix}\\ O \quad \cdots \quad O \quad S_{2im}\end{bmatrix}\left(\underline{\psi}_f(i)\right)^{\intercal}.$$

Therefore, similar to (35) through (38) the decoder derives,

$$\underline{\psi}_f(i)M_i=\left(\Theta_{\mathcal{H}}(i)\Upsilon_{\mathcal{H}}(i)\right)^{\intercal}, \tag{41}$$

and

$$e_f^{im\mu}\underline{\phi}_f S_{2im}=e_f^{\mu}\left(\Xi_{\mathcal{H}}(i)\Upsilon_{\mathcal{H}}(i)\right)^{\intercal}. \tag{42}$$

Finally, using (41) and (42), we have

$$\underline{x}_f(i)=\underline{\psi}_f(i)M_i+\left[O,\cdots,O,e_f^{im\mu}\underline{\phi}_f S_{2im}\right]_{\mu\times m\mu}. \tag{43}$$

□

The following algorithm summarizes the bandwidth adaptive repair procedure.

**Remark 2.** *Note that calculating the inverse matrices $(\Omega_{\mathcal{H}}(i))^{-1}$, for $i\in\{1,\cdots,\beta\}$, can be carried out recursively since from (29) we have,*

$$\Omega_{\mathcal{H}}(i)=\begin{bmatrix}e_{h_1}^{(i-1)m\mu} & 0 & \cdots & 0\\ 0 & e_{h_2}^{(i-1)m\mu} & \cdots & 0\\ \vdots & & \ddots & \vdots\\ 0 & 0 & \cdots & e_{h_d}^{(i-1)m\mu}\end{bmatrix}\Omega_{\mathcal{H}}(1)$$

**Algorithm 2** Bandwidth Adaptive Repair

Input: $f$, $e_f$, $\beta$, $m$, $\mu$ and $\underline{r}(h,f),\ \underline{\psi}_h,\ e_h$, for $h \in \mathcal{H}$.
Form matrices $\Upsilon_{\mathcal{H}}(i)$, for $i \in \{1,\cdots,\beta\}$, using (26), and (27).
Form vectors $\underline{\phi}_f$ using (31).
Calculate matrices $\Omega_{\mathcal{H}}(i)$, and $(\Omega_{\mathcal{H}}(i))^{-1}$, for $i \in \{1,\cdots,\beta\}$, using (29).
Calculate matrices $\underline{\psi}_f(1)M_1$ using (37).
Calculate matrices $e_f^{ma}\underline{\phi}_f S_{2m}$ using (38).
Recover $\underline{x}_f(1)$ using (39).
**for** $i = 2$ to $\beta$ **do**
    Calculate $\hat{\Upsilon}_{\mathcal{H}}(i)$ using (40).
    Calculate $\underline{\psi}_f(i)M_i$ using (35).
    Calculate $e_f^{im\mu}\underline{\phi}_f S_{2im}$ using (42).
    Recover $\underline{x}_f(i)$ using (43).
**end for**
Form $\underline{x}_f = [\underline{x}_f(1),\cdots,\underline{x}_f(\beta)]$.

**Remark 3.** *In a DSS with $n$ nodes, for $D = \{d_1,\cdots,d_\delta\}$, the bandwidth adaptive MSR codes presented in [2] require*

$$\alpha = \left(\mathrm{lcm}\left(d_1 - k + 1,\cdots,d_\delta - k + 1\right)\right)^n. \tag{44}$$

*Comparing (44) with (6), one could see that the presented scheme reduces the required $\alpha$ (and $\beta$) values exponentially. However, this scheme works only for $2k-2 \le d_i,\ \forall d_i \in D$. Hence, the design of high-rate bandwidth adaptive MSR codes with small $\alpha$ and $\beta$ still remains an open problem.*

The following example provides a detailed illustration of the MSR bandwidth adaptive exact-repair procedure in the same setup as described in Examples 1 to 4.

**Example 5.** As in the previous examples we will consider $\mu = 2$, and $\delta = 2$, which gives $k = 3$, and using (6), $\alpha = 4$. The code alphabet is $\mathbb{F}_{11}$, and the data matrix $M$ as given in (12). As a result, using (7), we have $D = \{d_1, d_2\} = \{4, 6\}$, and from (22) their associated per-node repair bandwidths are $\beta_1 = 2$, and $\beta_2 = 1$. Without loss of generality, let's assume node 7 is failed, *i.e.* $f = 7$. In this setup, the following two repair scenarios are then possible.

*A:* One option is to use $d = d_2 = 6$ helpers, and download only $\beta_2 = 1$ repair symbol from each of them, which means $d = (m+1)\mu$ for $m = 2$. Consider $\mathcal{H} = \{1, 2, \cdots, 6\}$. In this case, using the coefficient matrix $\Psi$ for this setup, which is given in (14), from (24), with $\alpha = 4, \beta = 1$, for any node $\ell$ we have

$$\underline{\psi}_\ell(1) = \underline{\psi}_\ell(1:4),$$

which is a row vector consisting of the first $\alpha = 4$ elements of the coefficient vector assigned to node $\ell$. As a result, using (23), each helper node $h \in \mathcal{H}$ will use all of its encoded content $\underline{x}_h$ to create a single repair symbol as

$$\begin{aligned}\underline{r}(h,f) = [r_1(h,f)] &= \left[\underline{x}_h(1)\cdot\left(\underline{\psi}_f(1)\right)^{\intercal}\right] \\ &= \left[\underline{x}_h \cdot [1,7,5,2]^{\intercal}\right].\end{aligned}$$

The repair decoder will then receive

$$\begin{aligned}\Upsilon_{\mathcal{H}} &= [\Upsilon_{\mathcal{H}}(1)] \\ &= \begin{bmatrix}\underline{r}(1,f)\\ \vdots \\ \underline{r}(6,f)\end{bmatrix} = \begin{bmatrix} x_{1,1}+7x_{1,2}+5x_{1,3}+2x_{1,4}\\ x_{2,1}+7x_{2,2}+5x_{2,3}+2x_{2,4}\\ x_{3,1}+7x_{3,2}+5x_{3,3}+2x_{3,4}\\ x_{4,1}+7x_{4,2}+5x_{4,3}+2x_{4,4}\\ x_{5,1}+7x_{5,2}+5x_{5,3}+2x_{5,4}\\ x_{6,1}+7x_{6,2}+5x_{6,3}+2x_{6,4}\end{bmatrix}.\end{aligned}$$

Moreover, using (31), and with $e_f = 7$ from (14) as also used in all previous examples, for $f = 7$ we get

$$\underline{\phi}_f = [1, 7].$$

Finally notice that in this case the repair decoding will only have one single step. Therefore, for step $i = 1$, with $m = 2$, and $\mu = 2$ we get $e_\ell^{im\mu} = e_\ell^4$, and hence,

$$\Omega_{\mathcal{H}}(1) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 5 & 10 \\ 1 & 3 & 9 & 5 & 4 & 1 \\ 1 & 4 & 5 & 9 & 3 & 1 \\ 1 & 5 & 3 & 4 & 9 & 1 \\ 1 & 6 & 3 & 7 & 9 & 10 \end{bmatrix}. \tag{45}$$

Then, based on the partition represented in Fing. (1), for

$$M_1 = \begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix},$$

the decoder has access to

$$\begin{aligned} \Upsilon_{\mathcal{H}} &= \Omega_{\mathcal{H}}(1) \begin{bmatrix} \begin{bmatrix} & \\ M_1 & \\ & \end{bmatrix} \\ O \quad S_4 \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal} \\ &= \Omega_{\mathcal{H}}(1) \begin{bmatrix} \begin{bmatrix} s_1 & s_2 & s_4 & s_5 \\ s_2 & s_3 & s_5 & s_6 \\ s_4 & s_5 & s_7 & s_8 \\ s_5 & s_6 & s_8 & s_9 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} s_{10} & s_{11} \\ s_{11} & s_{12} \end{bmatrix} \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal}. \end{aligned}$$

Note that the $\Omega_{\mathcal{H}}(1)$, given in (45) is an invertible matrix in the code alphabet $\mathbb{F}_{11}$, and we have

$$(\Omega_{\mathcal{H}}(1))^{-1} = \begin{bmatrix} \Theta_{\mathcal{H}}(1) \\ \Xi_{\mathcal{H}}(1) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 6 & 7 & 9 & 7 & 6 & 10 \\ 10 & 10 & 9 & 0 & 3 & 1 \end{bmatrix} \\ \begin{bmatrix} 3 & 6 & 9 & 1 & 9 & 5 \\ 1 & 8 & 0 & 8 & 2 & 3 \end{bmatrix} \\ \begin{bmatrix} 2 & 7 & 7 & 5 & 8 & 4 \\ 1 & 6 & 10 & 1 & 5 & 10 \end{bmatrix} \end{bmatrix}.$$

The decoder then calculates the lost data $\underline{x}_f$, using (39), with $e_f^\mu = 7^2 = 5$ in $\mathbb{F}_{11}$, as follows

$$\underline{x}_f = (\Theta_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}})^{\intercal} + [0, 0, 5\,(\Xi_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}})^{\intercal}].$$

*B:* The second option is to use $d = d_1 = 4$ helpers. With $\mu = 2$ we have $d = (m+1)\mu$ for $m = 1$. In this case, using (22) we will have $\beta_1 = 4/(4-2) = 2$ repair symbols per helper node. Let's without loss of generality assume node 7 is failed, *i.e.* $f = 7$, and $\mathcal{H} = \{1, 2, 3, 4\}$ is the set of helper nodes chosen to perform the repair. As a result, using the coefficient matrix in the code alphabet $\mathbb{F}_{11}$, given in (14), from equation (24) for the helper nodes we have

$$\begin{aligned} &\underline{\psi}_1(1) = [1, 1], \quad \underline{\psi}_1(2) = [1, 1], \\ &\underline{\psi}_2(1) = [1, 2], \quad \underline{\psi}_2(2) = [4, 8], \\ &\underline{\psi}_3(1) = [1, 3], \quad \underline{\psi}_3(2) = [9, 5], \\ &\underline{\psi}_4(1) = [1, 4], \quad \underline{\psi}_4(2) = [5, 9], \end{aligned}$$

and for the failed node $f = 7$,

$$\underline{\psi}_f(1) = [1, 7], \quad \underline{\psi}_f(2) = [5, 2].$$

Similarly, for the coded content of each of these nodes we consider the following partition

$$\begin{aligned} \underline{x}_\ell &= [\underline{x}_\ell(1), \underline{x}_\ell(2)] \\ &= [[x_{\ell,1}, x_{\ell,2}], [x_{\ell,3}, x_{\ell,4}]], \quad \text{for } \ell \in \{1, 2, 3, 4, 7\}. \end{aligned}$$

Each helper node $h \in \mathcal{H}$ then creates two repair symbols

$$\begin{aligned} \underline{r}(h, f) &= [r_1(h, f), r_2(h, f)] \\ &= \left[\underline{x}_\ell(1) \left(\underline{\psi}_f(1)\right)^{\intercal}, \underline{x}_\ell(2) \left(\underline{\psi}_f(2)\right)^{\intercal}\right] \\ &= [x_{h,1} + 7x_{h,2},\ 5x_{h,3} + 2x_{h,4}], \end{aligned}$$

and the repair decoder receives

$$\begin{aligned}\Upsilon_{\mathcal{H}} &= [\Upsilon_{\mathcal{H}}(1), \Upsilon_{\mathcal{H}}(2)] \\ &= \left[\begin{bmatrix} r_1(1,f) \\ \vdots \\ r_1(4,f) \end{bmatrix} \begin{bmatrix} r_2(1,f) \\ \vdots \\ r_2(4,f) \end{bmatrix}\right] \\ &= \left[\begin{bmatrix} x_{1,1}+7x_{1,2} \\ x_{2,1}+7x_{2,2} \\ x_{3,1}+7x_{3,2} \\ x_{4,1}+7x_{4,2} \end{bmatrix} \begin{bmatrix} 5x_{1,3}+2x_{1,4} \\ 5x_{2,3}+2x_{2,4} \\ 5x_{3,3}+2x_{3,4} \\ 5x_{4,3}+2x_{4,4} \end{bmatrix}\right].\end{aligned}$$

Moreover, using (31), and with $e_f = 7$ from (14) as also used in all previous examples, for $f = 7$ we get

$$\underline{\phi}_f = [1, 7].$$

Finally notice that in this case the repair decoding will have two steps. Therefore for step $i = 1$, with $m = 1$, and $\mu = 2$ we get $e_\ell^{im\mu} = e_\ell^2$, and hence, using equation (29), we have

$$\Omega_{\mathcal{H}}(1) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 5 \\ 1 & 4 & 5 & 9 \end{bmatrix}. \tag{46}$$

Then, according to the partition represented in Fig. (1), for

$$M_1 = S_1, \quad M_2 = S_3,$$

the repair decoder has access to

$$\begin{aligned}\Upsilon_{\mathcal{H}}(1) &= \Omega_{\mathcal{H}}(1) \begin{bmatrix} M_1 \\ S_2 \end{bmatrix} \left(\underline{\psi}_f(1)\right)^{\intercal} \\ &= \Omega_{\mathcal{H}}(1) \left[\begin{bmatrix} s_1 & s_2 \\ s_2 & s_3 \\ s_4 & s_5 \\ s_5 & s_6 \end{bmatrix}\right] \begin{bmatrix} 1 \\ 7 \end{bmatrix}.\end{aligned}$$

As expected $\Omega_{\mathcal{H}}(1)$ is an invertible matrix in the code alphabet $\mathbb{F}_{11}$, and we have

$$(\Omega_{\mathcal{H}}(1))^{-1} = \begin{bmatrix} \Theta_{\mathcal{H}}(1) \\ \Xi_{\mathcal{H}}(1) \end{bmatrix} = \left[\begin{bmatrix} 4 & 5 & 4 & 10 \\ 3 & 4 & 4 & 0 \\ 7 & 7 & 9 & 10 \\ 9 & 6 & 5 & 2 \end{bmatrix}\right].$$

The decoder then calculates the lost data $\underline{x}_f(1)$, using (39), with $e_f^{m\mu} = 7^2 = 5$ in $\mathbb{F}_{11}$, as follows

$$\underline{x}_f(1) = \left(\Theta_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}(1)}\right)^{\intercal} + 5\left(\Xi_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}}(1)\right)^{\intercal}.$$

To start the second step, $i = 2$, of the repair decoding, then the repair decoder first uses $e_f^{(i-1)m\mu} = 7^2 = 5$, to calculate

$$S_2 \left(e_f^{(i-1)m\mu} \underline{\phi}_f\right)^{\intercal} = 5\left(\Xi_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}}(1)\right),$$

where $5\left(\Xi_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}}(1)\right)$ is already derived in the step 1. Then the decoder calculates $\hat{\Upsilon}_{\mathcal{H}}(2)$ using equation (40) as,

$$\begin{aligned}\hat{\Upsilon}_{\mathcal{H}}(2) &= \Upsilon_{\mathcal{H}}(2) - \begin{bmatrix} e_1^{((2-1)m-1)\mu} \underline{\phi}_1 \\ \vdots \\ e_4^{((2-1)m-1)\mu} \underline{\phi}_4 \end{bmatrix} S_2 \left(e_f^{(2-1)m\mu} \underline{\phi}_f\right)^{\intercal} \\ &= \begin{bmatrix} r_2(1,f) \\ r_2(2,f) \\ r_2(3,f) \\ r_2(4,f) \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} S_2\left(5\left(\Xi_{\mathcal{H}}(1)\Upsilon_{\mathcal{H}}(1)\right)\right).\end{aligned}$$

From (41), with

$$\Omega_{\mathcal{H}}(2)=\begin{bmatrix}1&1&1&1\\4&8&5&10\\9&5&4&1\\5&9&3&1\end{bmatrix},$$

the repair decoder then has access to

$$\begin{aligned}\hat{\Upsilon}_{\mathcal{H}}(2)&=\Omega_{\mathcal{H}}(2)\begin{bmatrix}M_2\\S_4\end{bmatrix}\left(\underline{\psi}_f(2)\right)^{\intercal}\\&=\Omega_{\mathcal{H}}(2)\begin{bmatrix}\begin{bmatrix}s_7&s_8\\s_8&s_9\end{bmatrix}\\\begin{bmatrix}s_{10}&s_{11}\\s_{11}&s_{12}\end{bmatrix}\end{bmatrix}\begin{bmatrix}5\\2\end{bmatrix}.\end{aligned}$$

As expected $\Omega_{\mathcal{H}}(2)$ is also an invertible matrix in the code alphabet $\mathbb{F}_{11}$, and we have

$$\left(\Omega_{\mathcal{H}}(2)\right)^{-1}=\begin{bmatrix}\Theta_{\mathcal{H}}(2)\\\Xi_{\mathcal{H}}(2)\end{bmatrix}=\begin{bmatrix}\begin{bmatrix}4&4&9&2\\3&1&9&0\end{bmatrix}\\\begin{bmatrix}7&10&1&2\\9&7&3&7\end{bmatrix}\end{bmatrix}.$$

Finally, the decoder calculates the lost data $\underline{x}_f(2)$, using (39), with $e_f^{2m\mu}=7^4=3$ in $\mathbb{F}_{11}$, as follows

$$\underline{x}_f(2)=\left(\Theta_{\mathcal{H}}(2)\Upsilon_{\mathcal{H}(2)}\right)^{\intercal}+3\left(\Xi_{\mathcal{H}}(2)\Upsilon_{\mathcal{H}}(2)\right)^{\intercal}.$$

△

### *D. A Reversed Viewpoint*

In this subsection we provide a different view to the main contribution of this work in the form of the following Corollary. This result is indeed a direct corollary of the Theorem 1.

**Corollary 1.** *Consider a finite set of $\delta$ positive integers $D=\{d_1,\cdots,d_\delta\}$, $1<d_1<\cdots<d_\delta$, and let $\mu$ be a positive common divisor of all the elements of $D$, such that $\mu\notin D$. Then we introduce an adaptive bandwidth MSR code, with $k=\mu+1$, and a small sub-packetization level $\alpha$ and total storage capacity $F$, satisfying*

$$\alpha=\operatorname{lcm}(1,2,\cdots,\Delta)\,\mu,$$

*and*

$$F=k\alpha,$$

*which is capable of performing exact-repair using any arbitrary $d_i$ helpers, for any $d_i\in D$, and simultaneously satisfying the MSR characteristic equation (3) for all $d_i\in D$.*

*Proof.* Simply note that setting the code design parameter, $\mu$, and $\delta$, for the coding scheme presented in the previous subsections guarantees that the sub-packetization level and total storage capacity requirements are satisfied. Moreover, the presented coding scheme will provide the set of possible choices for the number of helpers for exact-repair as given in (7), which contains the required set $D$. □

**Remark 4.** *Note that for the case $d_1=1$, which is excluded in the above theorem, since any node should be repairable by any single helper, the trivial repetition code which stores the same $\alpha$ symbols in all the nodes in the network is the only possible solution, and it is easy to see that it is a bandwidth adaptive exact-repair MSR with $k=1$, as it satisfies the MDS property (2), and the MSR characteristic equation (3) could be satisfied with setting $\beta_i=\alpha/d_i$, for any $i\in\{1,\cdots,\delta\}$, as long as $d_i|\alpha$.*

## V. Discussion and properties

In this section we will briefly review some of the technical requirements and properties of the coding scheme presented in this work. Particularly, we will show that the field size and sub-packetization level requirements of the presented coding scheme are not significantly limiting factors in the practical implementations.

### A. Field Size Requirement

The only factor that influences the choice of the code alphabet $\mathbb{F}_q$ in the presented coding scheme is the existence coefficient matrix $\Psi$, and all the inverses of its square submatrices. To satisfy this requirement for a network with $n$ storage nodes, it is enough to have $q \geq n$ [32], which is the same as the field size requirement of many other coding schemes such as the Product Matrix codes [6] or the commonly used Reed-Solomon codes [4]. Hence, the presented coding scheme in this work is providing the bandwidth adaptivity property at no extra cost in the field size requirements, and any field of size larger than $n$ could be used as the code alphabet. It worth mentioning that the field size requirement of the only other bandwidth adaptive exact-repair MSR constructions, introduced in [2] is lower bounded by $n^2 - kn$ which is significantly larger for large distributed storage networks $n$. Moreover, techniques such as those presented in [33] could easily be applied in the presented coding scheme to reduce the field size to any arbitrary (e.g. binary) small field. However, this discussion is out of the scope of this work.

### B. Subpacketization Level

The requirement for the sub-packetization level $\alpha$ for the presented coding scheme is given in by (6). A natural question that arises is how fast does $\alpha$ grow as a function of the code design parameters, $\mu$, and $\delta$. From (6) it is clear that $\alpha$ is proportional to the design parameter $a$. Regarding the dependency of $\alpha$ on $\delta$, from [34] we have,

$$\operatorname{lcm}(1, 2, \cdots, \delta) \leq 4^{\delta},$$

and, recently [35] showed that

$$\operatorname{lcm}(1, 2, \cdots, \delta) \geq 2^{\delta}, \quad \text{for } \delta \geq 7.$$

As a result, for the presented coding scheme, we have

$$\mu 2^{\delta} \leq \alpha \leq \mu 4^{\delta}, \quad \text{for } \delta \geq 7.$$

## VI. Conclusion

We presented an alternative solution for exact-repair MSR codes in which optimal exact-repair is guaranteed simultaneously with a range of choices, $D = \{d_1, \cdots, d_{\delta}\}$, for the number of helpers. The introduced coding scheme is based on the Product Matrix framework, introduced first in [6]. The repair mechanism in this framework is based on specific symmetries in the structure of the *data matrix*. We proposed a novel structure for the data matrix, which preserves the required symmetries in many different submatrices. Corresponding repair mechanisms are also introduced to utilise these symmetries to perform optimal repair in different scenarios, whit varying number of helpers. In addition, the data reconstruction procedure is enhanced based on a novel successive interference cancellation scheme to perform optimally under the new design of data matrix. Comparing to the only other explicit constructions, presented in [2], we showed that when $d_i \geq 2k - 2, \ \forall d_i \in D$, the required values for $\alpha$, and $\beta$ are reduced to from $z_{\delta}^n$ to $kz_{\delta}$ for a DSS with $n$ nodes, and $k$ systematic nodes.

## Appendix A
## Proof of Lemma 2

Multiplying both sides of (15) by $\Phi^{\intercal}$ from right we get

$$X\Phi^{\intercal} = \Phi A \Phi^{\intercal} + \Delta \Phi B \Phi^{\intercal}. \tag{47}$$

Following the notation in [6], we introduce

$$P = \Phi A \Phi^{\intercal}, \quad Q = \Phi B \Phi^{\intercal}.$$

Then using (47) and the above equations we have

$$X\Phi^{\intercal} = P + \Delta Q. \tag{48}$$

Note that both $P$ and $Q$ are symmetric $(\mu + 1) \times (\mu + 1)$ matrices. Recall that $\Delta$ is a diagonal matrix, with non-zero and distinct diagonal elements. Hence, for any $1 \leq i < j \leq (\mu + 1)$, we now have both

$$(X\Phi^{\intercal})_{i,j} = P_{i,j} + \Delta_{i,i} Q_{i,j}, \tag{49}$$

and

$$\begin{aligned} (X\Phi^{\intercal})_{j,i} &= P_{j,i} + \Delta_{j,j} Q_{j,i} \\ &= P_{i,j} + \Delta_{j,j} Q_{i,j}. \end{aligned} \tag{50}$$

Then using (49) and (50), we recover all the non-diagonal elements of $P$ and $Q$. Now, let $\underline{\hat{p}}_i$, and $\underline{\hat{q}}_i$ denote the $i^{\text{th}}$ row of matrices $P$ and $Q$ excluding their diagonal elements. Moreover, let $\hat{\Phi}_i$ denote the submatrix of $\Phi$ derived by removing the $i^{\text{th}}$ row, and finally let's denote the $i^{\text{th}}$ row of $\Phi$ by $\underline{\phi}_i$. Then for all $i \in \{1, \cdots, (\mu+1)\}$ we have

$$\underline{\hat{p}}_i = \underline{\phi}_i A \left(\hat{\Phi}_i\right)^{\intercal},$$
$$\underline{\hat{q}}_i = \underline{\phi}_i B \left(\hat{\Phi}_i\right)^{\intercal},$$

and using Lemma 1, $\left(\hat{\Phi}_i\right)^{\intercal}$ is an invertible matrix, as it is a transposed generalized Vandermonde matrix. Then we calculate the following two matrices using the above equations for $i \in \{1, \cdots, \mu\}$,

$$\begin{bmatrix} \underline{\phi}_1 \\ \vdots \\ \underline{\phi}_\mu \end{bmatrix} A = \hat{\Phi}_{(\mu+1)} A, \quad \begin{bmatrix} \underline{\phi}_1 \\ \vdots \\ \underline{\phi}_\mu \end{bmatrix} B = \hat{\Phi}_{(\mu+1)} B.$$

Now since $\hat{\Phi}_{(\mu+1)}$ is invertible we have both $A$, and $B$.

## REFERENCES


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

[2] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2001–2014, April 2017.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. of IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, Lake Tahoe Incline Villiage, NV, USA, May 2010.

[4] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, vol. 8, no. 2, pp. 300—-304, Feb. 1960.

[5] A. G. Dimakis, P. B. Godfrey, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Anchorage, Alaska, USA, May 2007, pp. 2000–2008.

[6] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, August 2011.

[7] V. Cadambe, S. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic interference alignment for optimal repair of MDS codes in distributed storage," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2974–2987, May 2013.

[8] 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 *Proc. IEEE International Symposium on Information Theory (ISIT)*, Saint Petersburg, Russia, Aug. 2011, pp. 1225–1229.

[9] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Transactions on Information Theory*, vol. 59, no. 3, pp. 1597–1616, March 2013.

[10] B. Sasidharan, G. K. Agarwal, and P. V. Kumar, "A high-rate MSR code with polynomial sub-packetization level," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Hong-Kong, June 2015, pp. 2051–2055.

[11] V. Cadambe, C. Huang, and J. Li, "An alternate construction of an access-optimal regenerating code with optimal sub-packetization level," in *Proc. 21st Nat. Conf. Commun. (NCC)*, Bombay, India, Feb. 2015, pp. 1–6.

[12] K. Kralevska, D. Gligoroski, and H. Øverby, "General sub-packetized access-optimal regenerating codes," *IEEE Communication Letters*, vol. 20, no. 7, pp. 1281–1284, July 2016.

[13] N. Raviv, N. Silberstein, and T. Etzion, "Constructions of high-rate minimum storage regenerating codes over small fields," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2015–2038, April 2017.

[14] Z. Wang, I. Tamo, and J. Bruck, "Explicit minimum storage regenerating codes," *IEEE Transactions on Information Theory*, vol. 62, no. 8, pp. 4466–4480, Aug. 2016.

[15] M. Ye and A. Barg, "Explicit constructions of optimal-access MDS codes with nearly optimal sub-packetization," Available online: https://arxiv.org/abs/1605.08630, 2017.

[16] B. Sasidharan, M. Vajha, and P. V. Kumar, "An explicit, coupled-layer construction of a high-rate msr code with low sub-packetization level, small field size and $d < (n-1)$," Available online: https://arxiv.org/abs/1701.07447, 2017.

[17] S. Jiekak, A.-M. Kermarrec, N. L. Scouarnec, G. Straub, and A. V. Kempen, "Regenerating codes: A system perspective," *Proc. ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 2, pp. 23–32, July 2013.

[18] "Spacemonkey project," http://www.spacemonkey.com.

[19] "Tahoe: The least-authority file system," http://www.tahoe-lafs.org/trac/tahoe-lafs.

[20] F. Dabek, F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, Chateau Lake Louise, Banff, Canada, October 2001, pp. 202–215.

[21] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, NewYork, USA, October 2003.

[22] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The OceanStore prototype," in *Proc. USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, USA, March-April 2003.

[23] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. USENIX Conference on Networked System Design and Implementation (NSDI)*, San Francisco, USA, March 2004.

[24] A.-M. Kermarrec, N. L. Scouarnec, and G. Straub, "Repairing multiple failures with coordinated and adaptive regenerating codes," in *Proc. IEEE International Symposium on Network Coding (NetCod)*, Beijing, China, July 2011, pp. 1–6.

[25] V. Aggarwal, C. Tian, V. A. Vaishampayan, and Y.-F. R. Chen, "Distributed data storage systems with opportunistic repair," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Toronto, Canada, April–May 2014, pp. 1833–1841.

[26] M. Hajiaghayi and H. Jafarkhani, "MDS codes with progressive engagement property for cloud storage systems," Available online: https://arxiv.org/abs/1605.06927, 2016.

[27] K. Mahdaviani, A. Khisti, and S. Mohajer, "Bandwidth adaptive & error resilient regenerating codes with minimum repair bandwidth," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Barcelona, Spain, July 2016, pp. 235–239.

[28] X. Wang, Y. Xu, Y. Hu, and K. Ou, "MFR: Multi-loss flexible recovery in distributed storage systems," in *Proc. IEEE International Conference on Communications (ICC)*, Cape Town, South Africa, May 2010, pp. 1–5.
[29] K. Mahdaviani, A. Khisti, and S. Mohajer, "Bandwidth adaptive & error resilient MBR exact repair regenerating codes," Available online: http://www.comm.utoronto.ca/~akhisti/BAER_full.pdf, 2016.
[30] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster," in *Proc. 5th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, San Jose, USA, June 2013.
[31] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. of the USENIX Annual Technical Conference (ATC)*, Boston, USA, June 2012.
[32] H. Althaus and R. Leake, "Inverse of a finite-field vandermonde matrix," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 173–173, Jan. 1969.
[33] N. Raviv, "Asymptotically optimal regenerating codes over any field," Available online: https://arxiv.org/abs/1609.06420, 2016.
[34] M. Nair, "On chebyshev-type inequalities for primes," *The American Mathematical Monthly*, vol. 88, no. 2, pp. 126–129, Feb. 1982.
[35] V. Diekert, M. Kufleitner, G. Rosenberger, and U. Hertrampf, *Discrete Algebraic Methods: Arithmetic, Cryptography, Automata and Groups*. Walter de Gruyter GmbH & Co KG., 2016.