Regenerating Codes over a Binary Cyclic Code

Kenneth W. Shum[‡], Hanxu Hou[§†], Minghua Chen[§], Huanle Xu[§], and Hui Li[†*]
[‡]Institute of Network Coding, the Chinese University of Hong Kong
[§]Department of Information Engineering, the Chinese University of Hong Kong
[†]Shenzhen Eng. Lab of Converged Networks Tech., Shenzhen Key Lab of Cloud Computing Tech. and App., Peking University Shenzhen Graduate School

***Abstract*— We present a design framework of regenerating codes for distributed storage systems which employ binary additions and bit-wise cyclic shifts as the basic operations. The proposed coding method can be regarded as a concatenation coding scheme with the outer code being a binary cyclic code, and the inner code a regenerating code utilizing the binary cyclic code as the alphabet set. The advantage of this approach is that encoding and repair of failed node can be done with low computational complexity. It is proved that the proposed coding method can achieve the fundamental tradeoff curve between the storage and repair bandwidth asymptotically when the size of the data file is large.**

I. Introduction

Regenerating codes is a class of erasure-correcting codes introduced by Dimakis *et al.* in [1], with the aim of efficient repair of storage nodes. A data file is encoded and distributed to n storage nodes, such that the file can be decoded from any k of them. Furthermore, upon the failure of a storage node, we want to repair the failed node by downloading some data from any d surviving nodes, with the amount of data sent to the new node as little as possible. The number of data packets sent to the new node during the repair process is an important metric in measuring efficiency of node repair, and is coined the *repair bandwidth* in [1].

We differentiate two modes of repair. The first one is called exact repair and the second one functional repair. In exact repair, the content of the new node is required to be the same as in the failed node. In functional repair, the content of the new node need not be the same as in the failed one, but the property that any k nodes are sufficient in decoding the original file should be maintained. It is shown in [1] that, the minimization of repair bandwidth for functional repair is closely related to the single-source multi-cast problem in network coding theory. After formulating the problem using an information flow graph, a fundamental tradeoff between the amount of storage per node and the repair bandwidth is established. For exact repair, some recent result on the fundamental limit on repair bandwidth can be found in [2]. In the remaining of this paper, we focus on functional repair.

This work was partially supported by the National Basic Research Program of China (No.2012CB315904), NSFC61179028, by a grant from University Grants Committee of Hong Kong Special Administrative Region, China (Project No. AoE/E-02/08), and by the Shenzhen Key Laboratory of Network Coding Key Technology and Application, Shenzhen, China (ZSDY20120619151314964).
* Corresponding author.

In [3], existence of linear network codes achieving all points on the fundamental tradeoff curve for functional-repair regenerating codes is shown. The construction relies on arithmetic of finite field, and as in application of linear network code to single-source multi-cast problem in general, the underlying finite field must be sufficiently large. However, multiplication and division in finite field are costly to implement in software or hardware. In the literature of coding for disk arrays, the computational complexity is reduced by replacing arithmetic finite field by simple bit-wise operations. For example, in [4], maximal-distance separable (MDS) code with a convolutional code as alphabet set is introduced by Piret and Krol. In [5], Blaum and Roth proposed a construction of array codes based on the ring of polynomials with binary coefficients modulo $1 + x + \cdots + x^{p-1}$ for some prime number p. Similar approach was considered by Xiao *et al.* in [6]. Motivated by these constructions of low-complexity array codes, a class of regenerating codes utilizing the XOR operations and bit-wise shifts are proposed recently in [7]. The objective of this paper is to introduce another class of regenerating codes which enables repair by XOR and bit-wise *cyclic* shifts.

After reviewing some preliminaries on binary cyclic codes in Section III, we we show that we can operate arbitrarily close to the fundamental tradeoff curve between storage and repair bandwidth by this family of regenerating codes in Section IV. In Section V, we compare the computational complexity with functional-repair regenerating codes over finite field.

II. A Motivating Example

The following example of storage code illustrates the basic ideas. Suppose that we want to store some information bits to four storage nodes, such that we can recover the information bits from any two nodes. Nodes 1 and 2 store the information bits in uncoded format, and nodes 3 and 4 store some parity-check bits. The information bits are divided into groups of $2(m-1)$ bits, for some positive and *odd* integer m. Each group of $2(m-1)$ bits is called a data chunk. As the data chunks are processed in the same manner, we focus on one data chunk. We divide the $2(m-1)$ information bits into two equal parts, each consisting of $m-1$ bits. Let the bits in the first part be $b(1,0), b(1,1), \ldots, b(1,m-2)$, and the bits in the second part be $b(2,0), b(2,1), \ldots, b(2,m-2)$. For $i = 1,2$, let

$$b(i,m-1) := \sum_{j=0}^{m-2} b(i,j)$$

TABLE I
An Example of Storage Code for Four Nodes.

Node 1	Node 2	Node 3	Node 4
$b(1,0)$	$b(2,0)$	$b(1,0)+b(2,0)$	$b(1,0)+b(2,1)$
$b(1,1)$	$b(2,1)$	$b(1,1)+b(2,1)$	$b(1,1)+b(2,2)$
$b(1,2)$	$b(2,2)$	$b(1,2)+b(2,2)$	$b(1,2)+b(2,3)$
$b(1,3)$	$b(2,3)$	$b(1,3)+b(2,3)$	$b(1,3)+b(2,4)$
$b(1,4)$	$b(2,4)$	$b(1,4)+b(2,4)$	$b(1,4)+b(2,0)$

be the parity-check bit of $b(i,0),\dots,b(i,m-2)$.

In this storage code, we stores m bits in each node. For $i=1,2$, node i stores bits $b(i,0)$, $b(i,1),\dots,b(i,m-2)$ and the parity-check bit $b(i,m-1)$. Node 3 stores

$$b(3,j) := b(1,j)+b(2,j),$$

for $j=0,1,\dots,m-1$, and node 4 stores

$$b(4,j) := b(1,j)+b(2,j\oplus_m 1),$$

where $\oplus_m$ stands for addition modulo m. The rate of this storage code is $(m-1)/(2m)$, which is slightly less than $1/2$ for large m. An example for $m=4$ is illustrated in Table I.

We claim that we can recover the information bits from any two nodes. From node 1 and node 2, we can obtain the information bits directly. Hence the claim is obviously true for node 1 and node 2. If we want to decode the information bits from node 1 and node 3, we can subtract $b(1,j)$ from $b(3,j)=b(1,j)+b(2,j)$ to get the value of $b(2,j)$, for $j=0,1,2,\dots,m-2$. Likewise, We can verify that the information bits can be obtained from any one of the information disks and any one of the parity-check disks. Finally, suppose that node 1 and node 2 fail and we want to decode the information bits from node 3 and node 4. We can first compute

$$a(j) := b(3,j)+b(4,j) = b(2,j)+b(2,j\oplus_m 1),$$

for $j=1,3,5,\dots,m-2$. We can then recover $b(2,0)$ by computing

$$\sum_{\ell=1}^{(m-1)/2} a(2\ell-1) = b(2,1)+b(2,2)+\cdots+b(2,m-1) = b(2,0).$$

Once the value of $b(2,0)$ is known, we can get $b(1,0)$ by $b(3,0)+b(2,0)=b(1,0)$, and get $b(2,1)$ by $b(4,0)+b(1,0)=b(2,1)$, and so on. This proves the claim.

In this encoding method, the assumption that the parameter m to be an odd integer is essential. If m is an even integer and if we flip all the information bits $b(i,j)$ from 1 to 0 and from 0 to 1, for $i=1,2$ and $j=0,1,\dots,m-2$, then the content of node 3 and node 4 will not change. The mapping from the information bits in nodes 1 and 2 to the redundancy bits in nodes 3 and 4 is a two-to-one map in this case. So there is no way to recover the information bits from nodes 3 and 4.

We remark that in practical implementation we need not store the the last bit $b(i,m-1)$. As the last bit in each disk can be obtained by summing the first $m-1$ bits, we can compute the last bit $b(i,m-1)$ if necessary.

III. Codes over a Finite Principal Ideal Ring

The example in the previous section can be more compactly described if we represent the bits as the coefficients of a polynomial. For $i=1,2,3,4$, let

$$s_i(z) := \sum_{j=0}^{m-1} b(i,j)z^j,$$

where z is an indeterminate. Node i stores the coefficients of $s_i(z)$. Let

$$\mathcal{R} := \mathbb{F}_2[z]/(z^m+1).$$

and $\mathcal{C}$ be the subring of $\mathcal{R}$ consisting of polynomials with coefficients summing to zero. The polynomials $s_i(z)$ are regarded as elements in $\mathcal{C}$. The bits stored in node 3 and 4 are, respectively, the coefficients of $s_1(z)+s_2(z)$ and $zs_1(z)+s_2(z)$. The code shown in Table I can thus be regarded as a code of length 4 over the ring $\mathcal{C}$ with generator matrix

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

The ring $\mathcal{C}$ is a binary cyclic code, namely, the simple parity-check code. The regenerating codes constructed in this paper are indeed codes over a finite principal ideal ring [8]. Nevertheless, we only need the case when the finite principal ideal ring is a binary cyclic code. In the following, we review some basic theory of binary cyclic codes, in particular, the transform-domain analysis or the Mattson-Solomon polynomial. We refer the readers to standard texts on coding theory, such as [9], for more details.

Let m be a positive odd integer, and $f_0(z)f_1(z)\cdots f_L(z)$ be the prime factorization of polynomial x^m+1 over $\mathbb{F}_2$. Since $x+1$ is a factor of x^m+1, after some re-labeling, we assume without loss of generality that $f_0(z)=z+1$. A *binary cyclic code* $\mathcal{C}$ of length m is a subset of $\mathcal{R}$ which is closed under addition and multiplication by z. The polynomials in $\mathcal{C}$ are called the *codewords*. It is well known that a cyclic code is characterized by a generator polynomial, i.e., we can find a polynomial $g(z)\in\mathcal{R}$ such that $\mathcal{C}=\{a(z)g(z) : a(z)\in\mathcal{R}\}$. As a vector space, the dimension of $\mathcal{C}$ over $\mathbb{F}_2$ is equal to $m-\deg g$.

There is an alternate description of cyclic codes in terms of extension field. Let q be a power of 2 such that the finite field $\mathbb{F}_q$ contains a primitive m-th root of unity. Let γ be a fixed m-th root of unity in $\mathbb{F}_q$. For $\ell=0,1,2,\dots,L$, we define j_ℓ to be an integer between 1 and $m-1$ such that $f_\ell(\gamma^{j_\ell})=0$, and F_ℓ be the the smallest subfield in $\mathbb{F}_q$ which contains γ^{j_ℓ}. The field F_ℓ is the subfield of $\mathbb{F}_q$ with degree $\deg f_\ell(z)$ over $\mathbb{F}_2$. In particular, we have $j_0=0$ and $F_0=\mathbb{F}_2$.

For a given generator polynomial $g(z)$, let

$$\mathcal{N} := \{\ell\in\{0,1,\dots,L\} : g(\gamma^{j_\ell})\neq 0\}$$

be the set of *non-nulls*. The function

$$\theta(c(z)) := (c(\gamma^{j_\ell}))_{\ell\in\mathcal{N}}$$

is a bijection from $\mathcal{C}$ to the cartesian product $\prod_{\ell\in\mathcal{N}} F_\ell$. Given an $|\mathcal{N}|$-tuple $(\delta_\ell)_{\ell\in\mathcal{N}}$ in $\prod_{\ell\in\mathcal{N}} F_\ell$, the inverse of function θ can be computed by

$$(\delta_\ell)_{\ell\in\mathcal{N}} \mapsto \Big(\sum_{\ell\in\mathcal{N}} tr_\ell(\delta_\ell \gamma^{-ij_\ell})\Big)_{i=0}^{m-1},$$

where tr is the trace function from F_ℓ to $\mathbb{F}_2$,

$$tr_\ell(x) := x + x^2 + x^{2^2} + \cdots + x^{2^{\deg f_\ell - 1}}.$$

We define addition and multiplication on the cartesian product $\prod_{\ell\in\mathcal{N}} F_\ell$ by component-wise addition and component-wise multiplication, respectively. Then, the map θ preserves addition and multiplication, and is thus a ring-isomorphism.

Consider the parity-check code of length $m = 31$ as an example. The polynomial $z^{31}+1$ can be factorized as a product of $z+1$ and six irreducible polynomials of degree 5. The codewords of the parity-check code $\mathcal{C}$ of length 31 are the polynomials in $\mathcal{R}$ which have 1 as a root. Let γ be a primitive 31-st root of unity in $\mathbb{F}_{32}$. We have an isomorphism $\theta : \mathcal{C} \to (\mathbb{F}_{32})^6$ given by

$$\theta(c(z)) := (c(\gamma), c(\gamma^3), c(\gamma^5), c(\gamma^7), c(\gamma^{11}), c(\gamma^{15})).$$

We fix a binary cyclic code $\mathcal{C}$ of length m, and treat it as the alphabet set. A polynomial in $\mathcal{C}$ will be called a *symbol* or a *data packet*. Each symbol can carry $m - \deg g$ bits of information. We define a code over $\mathcal{C}$ of length ν and dimension κ by a $\kappa \times \nu$ matrix $\mathbf{G}$ over $\mathcal{R}$. The encoding is performed by multiplying a row vector $\mathbf{w}$ of length κ containing κ source symbols, and the generator matrix $\mathbf{G}$. An entry in $\mathbf{wG}$ is called a *coded packet* or a *coded symbol*. A coded packet is thus an $\mathcal{R}$-linear combination of the κ source symbols, with elements from $\mathcal{R}$ as the coefficients.

A collection of κ coded packets is called an *information set*, or said to be *decodable* if we can recover the κ source symbols in $\mathbf{w}$. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_\kappa\}$ be the index set of κ coded symbols and $\mathbf{G}[\mathcal{I}]$ be the submatrix of $\mathbf{G}$ obtained by retaining the columns indexed by $\mathcal{I}$. The coded symbols with indices in $\mathcal{I}$ form an information set if and only the submatrix $\mathbf{G}$ is invertiable, i.e., there is a matrix $\tilde{\mathbf{G}}$ over $\mathcal{R}$ such that $\mathbf{G}\tilde{\mathbf{G}}$ is the $\kappa \times \kappa$ identity matrix.

The next theorem gives an equivalence condition for decodability, and is a direct consequence of a result about codes over finite principal ideal ring in general [8, Thm. 6.3]. For each $\ell = 0, 1, 2, \ldots, L$, let $\theta_\ell : \mathcal{R} \to F_\ell$ be defined as

$$\theta_\ell(a(z)) := a(\gamma^{j_\ell}),$$

where $a(z)$ can assume any value in $\mathcal{R}$ (recall that γ^{j_ℓ} is a root of the polynomial $f_\ell(z)$). In terms of these notations, we can write

$$\theta(c(z)) = (\theta_\ell(c(z)))_{\ell\in\mathcal{N}}$$

for all $c(z) \in \mathcal{C}$. For a vector $\mathbf{v}$ whose components are polynomials in $\mathcal{R}$, we define $\theta_\ell(\mathbf{v})$ by applying θ_ℓ component-wise.

Theorem 1. *Let $i_1, i_2, \ldots, i_\kappa$ be κ column indices of a generator matrix $\mathbf{G}$, and let $\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_\kappa}$ be the columns of $\mathbf{G}$ indexed by $i_1, \ldots, i_\kappa$. The coded symbols indexed by $i_1, \ldots, i_\kappa$ are decodable if and only if the vectors $\theta_\ell(\mathbf{v}_{i_1})$, $\theta_\ell(\mathbf{v}_{i_2}), \ldots, \theta_\ell(\mathbf{v}_{i_\kappa})$ are linearly independent over F_ℓ for each $\ell \in \mathcal{N}$.*

IV. Optimal Functional-Repair Regenerating Codes over Binary Parity-Check Code

In the rest of this paper, we consider the case that the binary cycle code be the simple parity-check code of length m and dimension $m-1$, with m equal to a prime number. The results can be readily generalized to general binary cyclic codes.

In the encoding process, the data file is first divided into many pieces, with each piece containing $B(m-1)$ bits, where B is an integer to be determined later. As each piece of data will be encoded and decoded in the same manner, for the ease of presentation, we assume that a data file contains $B(m-1)$ bits. Each group of $m-1$ bits is encoded to a codeword of the binary cyclic code $\mathcal{C}$. We let $s_1(z), s_2(z), \ldots, s_B(z) \in \mathcal{C}$ be the resulting codewords. We call these B packets the *source data packets*.

We store α coded packets in each node. Each coded packets is an $\mathcal{R}$-linear combination of the B source data pckets. The coefficients of the linear combination form the *global encoding vector* of the corresponding coded packet. We want to generate the coded packets such that in each collection of k storage nodes, we can find B decodable packets among the $k\alpha$ coded packets. We call this property the (n, k) *recovery property*. (This is weaker than the maximal-distance separable (MDS) property because the binary cyclic code may have rate strictly less than 1.)

There is a time index which is initialized to 0, and is advanced by 1 after a node repair. Suppose that the current time index is t and node f fails. The new node which replaces the failed one is generated by contacting d surviving nodes. The storage nodes which participate in the repair process are also called the *helpers*. At time t, let H_t denote the index set of the d helper nodes. Each of the helper node transmits β packets to the new node, and each of these packets is obtained by adding some cyclic-shifted version of the α encoded packets in the memory. Suppose that node i, for some $i \in H_t$, is a helper. The global encoding vector of the b-th packet sent to the new node, for $b = 1, 2, \ldots, \beta$, is of the form

$$\sum_{j=1}^{\alpha} z^{\sigma_{ij}^t(b)} \mathbf{v}_{ij}^t,$$

where $\mathbf{v}_{ij}^t$ is the global encoding vector of the j-th packet in the i-th node at time t, and $\sigma_{ij}^t(b)$ is an integer between 0 and $m-1$. Upon receiving the $d\beta$ packets from the helpers, the new node computes and stores α packets. For $a = 1, 2, \ldots, \alpha$, the α-th packet stored in the new node is obtained by first cyclically shift the c-th received packet by $\tau_c^t(a)$ bits and then adding the resulting packets. The local encoding coefficients are powers of z, and the computations required during the repair process are just cyclic shifts and binary additions. The global encoding vectors of the new packets are also computed and stored. We want to show that by choosing the values of $\sigma_{ij}^t(b)$'s and $\tau_c^t(a)$'s appropriately, we can maintain the property that there are B decodable packets in any set of k nodes.

We can prove this by modifying the argument in [3] on the existence of regenerating codes over finite field, and invoking the DeMillo-Lipton-Schwartz-Zippel lemma (see e.g. [10, p. 224]).

Lemma 2 (DeMillo-Lipton-Schwartz-Zippel)**.** *Let $\mathbb{F}$ be a finite field and $\mathcal{S}$ be a subset of elements in $\mathbb{F}$. Let f be a non-zero multivariate polynomial in $\mathbb{F}[X_1, X_2, \ldots, X_N]$ of degree e. Then the polynomial f has at most $e|\mathcal{S}|^{N-1}$ roots in $\mathcal{S}^N$.*

In [3], the existence of regenerating codes over a finite field is proved by showing that we can choose the local encoding coefficient such that a collection of determinants are all evaluated to be non-zero. In the case of regenerating codes over a binary parity-check code, we want to restrict the local encoding coefficients to be powers of z, and the collection of determinants are evaluated to be non-zero in several finite fields. With these modification, the requirement on the packet length, m, will be stated in the next theorem.

Given the values of system parameters n, k, d, α and β, we let the size of a piece of data, B, be

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

A storage code with these system parameters attains the fundamental tradeoff between the amount of storage per node and the repair bandwidth. For $i = 1, 2, \ldots, k$, let s_i be the i-th term in the above summation,

$$s_i := \min\{(d-i+1)\beta, \alpha\},$$

and for $i = k+1, k+2, \ldots, n$ let $s_i = 0$. Define $\mathcal{H}$ as the set of vectors of length n, whose components are non-negative integers, which are majorized by the vector $\mathbf{s} = (s_1, s_2, \ldots, s_n)$. In other words, if we sort the components of a vector $\mathbf{h} \in \mathbb{Z}_+^n$ in non-increasing order as $h_{[1]} \geq h_{[2]} \geq \cdots \geq h_{[n]}$, then $\mathbf{h}$ is in $\mathcal{H}$ if and only if

$$\sum_{i=1}^{\mu} h_{[i]} \begin{cases} \leq \sum_{i=1}^{\mu} s_i & \text{for } m = 1, 2, \ldots, n-1, \\ = B & \text{for } \mu = n. \end{cases}$$

We refer the readers to [11] for more details on majorization theory.

Theorem 3. *Let n, k, d, α and β be fixed system parameters of a distributed storage system. Let m be an odd prime, and $ord_m(2)$ be the multiplicative order of 2 mod m. If m is larger than*

$$\frac{m-1}{ord_m(2)} \cdot B \cdot \max\Big\{ \binom{n\alpha}{B}, 2|\mathcal{H}| \Big\}, \tag{2}$$

where B is defined in (1), then there exists a functional-repair regenerating code over the binary parity-check code of length m, which achieves the capacity of a distributed storage system with system parameters n, k, d, α and β, using only shift and add operations in the encoding and repairing process.

Sketch of proof: The proof is basically the same as in [3]. The encoding coefficients in $\mathbf{v}_{ij}^0$'s when we initialize the storage system are restricted to be powers of z. The local encoding coefficients in each repair process are also powers of z. The local encoding coefficients in each repair process are chosen such that a collection of sets of k packets are decodable. For each set of k packets in this collection, we need to guarantee that the decodability by invoking Theorem 1 in the previous section. We note that x^m+1 has $1+(m-1)/ord_m(2)$ factors, one of which is $x+1$ and the rests are factors of degree $ord_m(2)$. In the application of Lemma 2, we take the set $\mathcal{S}$ to be the powers of z. ■

We may choose the parameter m to be a prime number larger than $n-3$, such that the multiplicative order of 2 mod m is equal to $m-1$. In this case, the polynomial x^m+1 is factorized as a product of two irreducible polynomials, namely, $x+1$ and $x^{m-1} + x^{m-2} + \cdots + 1$. The value of $ord_m(2)$ in this case is equal to $m-1$. Under the Artin's conjecture on primitive roots, there are infinitely many such prime number m [12].

V. Comparison of Computational Complexity

In this section we estimate the computational complexity of encoding, repairing and decoding, in terms of the number of XOR's. Due to the limitation of space, we only consider the case when each piece of data contains $B = k\alpha$ packets, i.e., each set of k nodes contains just enough information to decode the original data file. The equation in (1) holds with $B = k\alpha$ when $(d-i+1)\beta \geq \alpha$ for all $i = 1, 2, \ldots, k$. The minimum value of β is thus $\beta = \alpha/(d-k+1) = B/(k(d-k+1))$. In the remaining of this section, the parameters B, α and β are set to $B = k(d-k+1)$, $\alpha = d-k+1$ and $\beta = 1$.

Without loss of generality we assume that the data file contains $\kappa B(m-1) = \kappa k\alpha(m-1)$ bits, where m is an integer satisfying the condition in Theorem 3, and κ is a positive integer. The computation of the local encoding coefficients is neglected, as it is negligible when κ is very large. For the ease of comparison, we will normalize the complexity by the file size.

Encode. To each piece of data, we first append a parity-check bits after each $m-1$ bits to obtain B codewords in $\mathcal{C}$. The calculation of the B parity-check bits in one piece of data requires $O(Bm)$ XOR operations. A coded packet is obtained by cyclically shifting and adding the B codewords, and the computational complexity is $O(Bm)$ bit operations. Hence the total number of XOR's in encoding is $O(\kappa n\alpha Bm)$. The normalized computational complexity of encoding is $O(\kappa n\alpha Bm/(\kappa B(m-1)) = O(n\alpha) = O(n(d-k+1))$.

Repair. Each of the helping node generates $\kappa\beta$ coded packets by shifting and adding the $\kappa\alpha$ packets in its memory. The total number of XOR in generating one packet to be sent

TABLE II
COMPARISON OF COMPUTATIONAL COMPLEXITY

	Normalized redundancy	Normalized repair bandwidth	Normalized encoding complexity	Normalized repair complexity	Normalized decoding complexity
RC over binary cyclic code	$\frac{m}{m-1} \cdot n/k$	$\frac{m}{m-1} \cdot d/(k(d+1-k))$	$O(n(d-k+1))$	$O(d/k)$	$O(m)$
RC over finite field	n/k	$d/(k(d+1-k))$	$O(n(d-k+1)m_1)$	$O(dm_1/k)$	$O(m_1)$

to the new node is $O(\alpha m)$. The total number of bit operations from the transmitting side is $O(\kappa d\beta\alpha m)$.

The new node generates $\kappa\alpha$ coded packets. Each of them is obtained by combining the $d\beta$ received packets. The required number of XOR's is $O(\kappa\alpha d\beta m)$. The normalized computational complexity of the repair of a failed node is $O(\kappa\alpha d\beta m/(\kappa B(m-1)) = O(d/k)$.

Decode. A data collector recovers the data file by linearly combining $k\alpha$ coded packets. The coefficients in the linear combination are polynomial in $\mathcal{A}$ and are obtained by solving some system of linear equations. We ignore the computational complexity in calculating these coefficients as it is negligible asymptotically when κ is large. The complexity is directly proportional to the number of terms in the coefficients, and in the worst case, there are m terms in each of them. The number of XOR's in recovering one source packet is therefore $O(k\alpha m^2)$. The normalized computational complexity of decoding the data file is $O(\kappa k\alpha m^2/(\kappa k\alpha(m-1))) = O(m)$.

We compare with the functional-repair regenerating codes over finite field in [3]. The size of the finite field is 2^{m_1}, where m_1 is an integer larger than

$$\log_2 \Big(B \cdot \max\Big\{\binom{n\alpha}{B}, 2|\mathcal{H}|\Big\}\Big). \tag{3}$$

We implement the finite field of size 2^{m_1} as the quotient ring $\mathbb{F}_2[x]/(g(x))$ for some irreducible polynomial $g(x)$ of degree m_1, and use a polynomial basis to represent a finite field element. Addition is bit-wise XOR and multiplication takes $O(m_1^2)$ bit operations. (See e.g. [13, Chp. 11].) A coded packet when we start the storage system is obtained by taking an $\mathbb{F}_{2^{m_1}}$-linear combination of the packets. The computation of such a linear combination is dominated by the B multiplications, and the computational complexity is $O(Bm_1^2)$ for each coded packet. The normalized encoding complexity is $O(\kappa n\alpha Bm_1^2/(\kappa Bm_1)) = O(n\alpha m_1) = O(n(d-k+1)m_1)$.

The comparison of computational complexity is summarized in Table II. The first row is the performance metric of the proposed regenerating codes (RC) over binary parity-check codes, and the second row is regenerating codes using a finite field as alphabet. The *normalized redundancy* is defined as the total number of bits in the storage system divided by the number of bits in the data file. As we are comparing at the minimum-storage regenerating point, the storage efficiency is $n\alpha/B = n/k$ for RC over finite field. The coding scheme proposed in this paper has an additional 1 bit per m bits, and this leads to a slight increase of normalized redundancy by a factor of $m/(m-1)$. Similarly, there is a factor of $m/(m-1)$ in the normalized repair bandwidth of RC over parity-check code. The storage efficiency and normalized repair bandwidth of the two coding schemes are approximately the same when m is large.

The regenerating code over binary parity-check code has small complexity in encoding and repair. The encoding and repair computational complexity normalized by the file size does not depend on the packet size m. On the contrary, the encoding and repair complexity of regenerating codes over finite field depend on the field size, and is proportional to the number of bits carried in a finite field element, m_1. The packet size m in regenerating code over parity-check code, however, affects the decoding complexity. The lower bound of m in Theorem 3 is however much larger than the lower bound of m_1 in (3).

VI. CONCLUSION

We propose a framework of designing regenerating codes which employ XOR and bit-wise cyclic shifts. The advantage is that encoding and the repair of a failed node can be carried out by simple cyclic shift operations and binary additions. For functional repair, all operating points on the fundamental tradeoff curve between storage and repair bandwidth can be achieved asymptotically when the file size is large.

REFERENCES

[1] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage system," in *Proc. IEEE INFOCOM*, Anchorage, Alaska, May 2007, pp. 2000–2008.

[2] C. Tian, "Rate region of the (4, 3, 3) exact-repair regenerating codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Istanbul, July 2013, pp. 1426–1430.

[3] Y. Wu, "Existence and construction of capacity-achieving network codes for distributed storage," *IEEE J. Selected Areas in Communications*, vol. 28, no. 2, pp. 277–288, February 2010.

[4] P. Piret and T. Krol, "MDS convolutional codes," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 224–232, March 1983.

[5] M. Blaum and R. M. Roth, "New array codes for multiple phased burst correction," *IEEE Trans. Information Theory*, vol. 39, no. 1, pp. 66–77, January 1993.

[6] M. Xiao, T. Aulin, and M. Médard, "Systematic binary deterministic rateless codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, July 2008, pp. 2066–2070.

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

[8] S. T. Dougherty, J.-L. Kim, and H. Kulosman, "MDS codes over finite principal ideal rings," *Des. Codes Cryptogr.*, vol. 50, pp. 77–92, 2009.

[9] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier science publishers, 1977.

[10] S. Jukna, *Extremal combinatorics - with applications in computer science*, 2nd ed. Berlin: Springer-Verlag, 2011.

[11] A. W. Marshall and I. Olkin, "Theory of majorization and its applications," *Academic, New York*, 1979.

[12] M. R. Murty, "Artin's conjecture for primitive roots," *Math. Intelligencer*, vol. 10, no. 4, pp. 59–67, 1988.

[13] H. Cohen and G. Frey, Eds., *Handbook of elliptic and hyperelliptic curve cryptography*. Chapman & Hall/CRC, 2006.