Progressive Data Retrieval for Distributed Networked Storage

Yunghsiang S. Han, *Fellow*, *IEEE*, Soji Omiwade, *Member*, *IEEE*, and Rong Zheng, *Senior Member*, *IEEE*

Abstract—We propose a decentralized progressive data retrieval (PDR) mechanism for data reconstruction in a network of Byzantine and crash-stop nodes. The scheme progressively retrieves stored data, such that it achieves the minimum communication cost possible. In particular, PDR gracefully adapts the cost of successful data retrieval to the number of Byzantine and crash-stop storage nodes. At the core of PDR is an incremental Reed-Solomon decoding (IRD) procedure that is highly computation efficient for data reconstruction. IRD's computation efficiency arises from its ability to utilize intermediate computation results. In addition, we provide an in-depth analysis of PDR and compare it to decentralized erasure coding and decentralized fountain coding algorithms for distributed storage systems. Moreover, our implementation results show that PDR has up to 35 times lower computation time over the state-of-the-art error-erasure decoding scheme for distributed storage systems. In our analysis, we also show that the code structure of PDR and the number of available storage nodes are independent of each other, and they can be used to control both the data dissemination and retrieval complexity.

Index Terms—Fault tolerance, error control codes, Reed-Solomon codes, byzantine failure

1 Introduction

The cost of storage has decreased to the point that most storage-space providers offer terabytes of affordable storage to their respective consumers. Moreover, low-power storage media have become widely used in embedded devices and mobile computers. In order to harness the ever growing capacity and decreasing cost of storage for distributed networked storage systems, the following challenges must be addressed:

- Storage volatility due to network disconnections, varying administrative restrictions and nodal-mobility.
- Data erasures; the prevalent flash media has limited program-erase cycles because repeated erasures of any particular block may result in modifications of neighboring blocks.
- Software bugs or malicious attacks, where an adversary manages to compromise the system and modify its data.

At the core of a *robust* networked storage system is a coding scheme that maps information bits to coded bits that are stored distributedly using a set of storage nodes. The information bits can be retrieved from the coded bits, and we refer to the units of this bidirectional mapping as *symbols*. In this paper, coded symbols and storage symbols are used interchangeably. An (n, k) erasure code can be defined by the following two primitives:

- **encode** $\mathbf{c} = (\mathbf{u}, n, k)$ takes as input, k information symbols $\mathbf{u} = [u_0, u_1, \ldots, u_{k-1}]$, and returns a coded vector $\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}]$. Similarly, the coded symbols are stored on n storage nodes, one per node.
- **decode** $\mathbf{u} = (\mathbf{r}, n, k)$ accesses a subset of storage nodes and returns the original k information symbols from possibly corrupted symbols.

For the decode primitive, if any k of the n coded symbols suffice to reconstruct the original data, the code has optimal reception efficiency and is referred to as an (n, k)-Maximum Distance Separable (MDS) erasure code.

A crash-stop or erasure node in a networked storage system is one that becomes unavailable because it can neither transmit nor receive data. Byzantine nodes are those nodes that fail in an arbitrary manner and cannot be trusted. They are more pertinent than erasure nodes given the prevalence of cheap storage devices, software bugs, and malicious attacks [1]. Crash-stop nodes can be identified, via keep-alive messages for instance. However, the identity of Byzantine nodes can be known, to a data collector requiring data reconstruction, only via an error-erasure decoding algorithm.

Efficient encode and decode primitives that can detect data corruption and handle Byzantine failures serve as a fundamental building block to support higher level abstractions such as multireader multiwriter atomic register and digital fingerprints in dependable distributed systems [2], [3]. Given the primitives' fixed error correction capability, their efficiency can be evaluated using the following: the storage overhead, measured as the ratio between the number of storage symbols and total information symbols; the encoding and decoding computation cost; and the communication

- *Y.S. Han is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, #43, Sec. 4, Keelung Rd., Taipei, Taiwan 106, R.O.C. E-mail: yshan@mail.ntust.edu.tw.*
- *S. Omiwade and R. Zheng are with the Department of Computer Science, University of Houston, Philip G. Hoffman Hall, Houston, TX 77204-3010. E-mail: {ooomiwade, rzheng}@uh.edu.*

Manuscript received 8 Oct. 2011; revised 15 Jan. 2012; accepted 31 Jan. 2012; published online 8 Feb. 2012.
Recommended for acceptance by X.-H. Sun.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-10-0756.
Digital Object Identifier no. 10.1109/TPDS.2012.67.

overhead, measured in terms of the total bits transferred in the network for encoding and decoding. The communication overhead is most significant in wide-area networks, sensor networks, and low-bandwidth storage systems [4], [5]. In this work, we propose efficient methods to achieve minimum communication costs.

Reed-Solomon (RS) codes have been used for storage in stand-alone storage systems [6]. However, our work concerns RS codes for networked storage. Even though RS codes can be applied to achieve minimum communication overhead, their centralized nature and high coordination overhead make them unsuitable [7], [8]. To address these classical RS coding deficiencies and the need for error correction capabilities in distributed storage systems, we propose a storage-optimal and decentralized coding scheme that disperses contents of a data file, and retrieves it efficiently in networked storage systems. Our RS coding scheme provides better performance than the decentralized erasure and fountain coding schemes.

The key novelty of our solution lies in a progressive data retrieval procedure, which retrieves just enough data from live storage nodes and performs decoding incrementally. As a result, both communication and computation costs are minimized, and adapt to the degree of errors, due to Byzantine nodes, in the system. We provide an analytical characterization of the communication cost for data retrieval using the proposed scheme in the presence of arbitrary Byzantine nodes. Our implementation studies demonstrate up to 35 times lower computation time in decoding, of the progressive data retrieval scheme relative to the classical RS decoding scheme. Moreover, the proposed scheme is comparable to that of a genie-aid decoding process, which assumes knowledge of failure modes. In this paper, we make the following contributions:

- Design and implementation of a novel progressive data retrieval algorithm that is storage and communication optimal, and computationally efficient.
- Provision of an analytical model to evaluate the communication cost of the proposed data retrieval algorithm.
- Security analysis of the proposed decoding scheme.

The rest of the paper is organized as follows. In Section 2, we provide some background information on RS codes. Related work is provided in Section 3, and the progressive data retrieval scheme is presented in Section 4, with the details of the incremental RS decoding algorithm in Section 5. We derive the communication, computation, and security-strength complexities of our proposed scheme in Section 6 and compare it to existing storage schemes. Implementation and evaluation results are presented in Section 10 and we conclude the paper in Section 13.

2 Background

In distributed storage systems, ensuring data persistence given the existence of crash-stop and Byzantine nodes in the network requires the introduction of redundancy for data storage. The simplest form of redundancy is replication; as a generalization of replication, erasure coding offers higher storage efficiency [9]. To store an arbitrary file using (n, k) erasure codes, we divide the file into groups of k symbols. For each group, the k symbols are encoded into n coded symbols and stored at n respective storage nodes.

2.1 RS Codes

An (n, k)-RS code is an (n, k)-MDS code with error correction capabilities. They can recover the original codeword when v symbols are erroneous, if $v \leq \lfloor \frac{n-k-s}{2} \rfloor$, where s is the number of erased symbols. RS codes are the most well-known class of MDS codes. They operate on symbols of m bits, and are linear codes where each symbol is in $GF(2^m)$. n and k satisfy $n = 2^m - 1$ and $n - k = 2t$, where t is the error correction capability of the code.

2.2 Encoding

Let the vector of k information symbols in $GF(2^m)$ be $\mathrm{u} = [u_0, u_1, \ldots, u_{k-1}]$ and $u(x)$ be the information polynomial of u represented as (1). The codeword polynomial, $c(x)$, corresponding to $u(x)$ can be encoded as (2), where $g(x)$ is a generator polynomial of the RS code. The generator polynomial can be computed as (3); α is a primitive element in $GF(2^m)$ and can be determined in advance. b is an arbitrary integer, and g_i is in $GF(2^m)$. It follows from (2) and (3) that $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+2t-1}$ are the roots of $g(x)$ and $c(x)$

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \tag{1}$$

$$c(x) = u(x) g(x) \tag{2}$$

$$\begin{aligned} g(x) &= (x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+2t-1}) \\ &= g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t} x^{2t}. \end{aligned} \tag{3}$$

An alternative encoding mechanism for RS codes is the matrix-vector product shown in (4) [9]. $\boldsymbol{G}$ in (5) is the Vandermonde matrix, and any k columns are linearly independent. The constructions in (2) and (4) result in the same Reed-Solomon code, when b equals 1

$$\boldsymbol{c} = \boldsymbol{u}\boldsymbol{G} \tag{4}$$

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^n \\ \alpha^2 & (\alpha^2)^2 & (\alpha^3)^2 & \cdots & (\alpha^n)^2 \\ & & & \vdots & \\ \alpha^{k-1} & (\alpha^2)^{k-1} & (\alpha^3)^{k-1} & \cdots & (\alpha^n)^{k-1} \end{bmatrix}. \tag{5}$$

2.3 Decoding

Denote the received polynomial as $r(x)$; if there are neither erasures nor errors in the received codeword, then $r(x)$ matches the original codeword polynomial, $c(x)$. When errors and erasures exist in $r(x)$, the decoding is a multistep procedure: Auxiliary polynomials, also known as syndromes, are first computed from the original codeword $c(x)$. These syndromes are then used to compute an *error-locator* polynomial that is used to identify erroneous symbols in the received codeword. After all errors have been identified and corrected in $r(x)$, any k symbols are selected and the original set of information symbols $u(x)$ can be derived by inverting the k columns of the Vandermonde matrix that correspond to the k selected symbols.

We discuss each step in detail, beginning with decoding when there are only Byzantine nodes. Later, we generalize to codewords having erasures. The received polynomial can be written as a sum of the original codeword and the errors introduced, as shown in (6). $e(x)$ is the polynomial indicating the locations and values of each error in $r(x)$. Syndrome S_i is given by the expression in (7), for each i in $\{b, b+1, \dots, b+2t-1\}$. Since $g(x)$ and $c(x)$ have $\alpha^b, \alpha^{b+1}, \dots, \alpha^{b+2t-1}$ as roots, then if the ith symbol in the received codeword $r(x)$ is not erroneous, S_i evaluates to zero

$$r(x) = c(x) + e(x) \text{ where } e(x) = \sum_{j=0}^{n-1} e_j x^j \tag{6}$$

$$S_i = r(\alpha^i) = e(\alpha^i) = \sum_{j=0}^{n-1} e_j \alpha^{ij}. \tag{7}$$

Assume that $v \leq t$ errors occur in unknown locations $j_1, j_2, \dots, j_v$ of the received polynomial. Then, $e(x)$ in (6) can be expressed as (8), and e_{j_ℓ} is the value of the ℓth error, for each ℓ in $\{1, 2, \dots, v\}$. A requirement for successful decoding is to determine j_ℓ and e_{j_ℓ} for all ℓ. Instead of solving the $2t$ syndrome equations in (7), an *error-locator* polynomial is introduced in (9). For all ℓ in $\{1, 2, \dots, v\}$, Λ_ℓ are the coefficient of $\Lambda(x)$, when the product is simplified, where Λ_0 is unit. By Newton's identity, we then have (10), where $i \in \{b+v, b+v+1, \dots, b+2t-1\}$ [10]

$$e(x) = \sum_{\ell=1}^{v} e_{j_\ell} x^{j_\ell} \tag{8}$$

$$\Lambda(x) = \prod_{\ell=1}^{v} (1 - x\alpha^{j_\ell}) = \sum_{\ell=0}^{v} \Lambda_\ell x^\ell \tag{9}$$

$$S_i = -\sum_{j=1}^{v} \Lambda_j S_{i-j}. \tag{10}$$

The coefficients of the error-locator polynomial can be determined by applying the classical Berlekamp-Massey algorithm (BMA), Welch-Berlekamp (WB) or Euclid's algorithm to solve (10). Once all coefficients of the error-locator polynomial are found, the values α^{j_ℓ} can be determined by successive substitution through a procedure known as the Chien search. We then solve for each e_{j_ℓ} via the Forney formula [10]. Fig. 1 summarizes the RS decoding process.

When both errors and erasures exist, an error-erasure decoding algorithm must be implemented in order to decode the received vector efficiently [10], [11], [12]. Let $r(x)$ be the received polynomial and $r(x) = c(x) + e(x) + \gamma(x) = c(x) + \lambda(x)$, where $e(x) = \sum_{j=0}^{n-1} e_j x^j$ is the error polynomial, $\gamma(x) = \sum_{j=0}^{n-1} \gamma_j x^j$ the erasure polynomial, and $\lambda(x) = \sum_{j=0}^{n-1} \lambda_j x^j = e(x) + \gamma(x)$ the errata polynomial. For error-erasure decoding, two extra decoding steps are needed compared with the error-only decoding: calculation of erasure-locator polynomial and computation of the modified syndromes. The modified syndromes are then used to replace the normal syndromes in the decoding procedure. If the location of the erasures are given, the erasure-locator polynomial can be calculated directly as follows. Let $i_1, i_2, \dots, i_s$ be the s erasure positions. Using the definition of $\Gamma(x)$ shown in (11), the modified syndromes are obtained by (12), where j is in $\{b+s, b+s+1, \dots, b+2t-1\}$

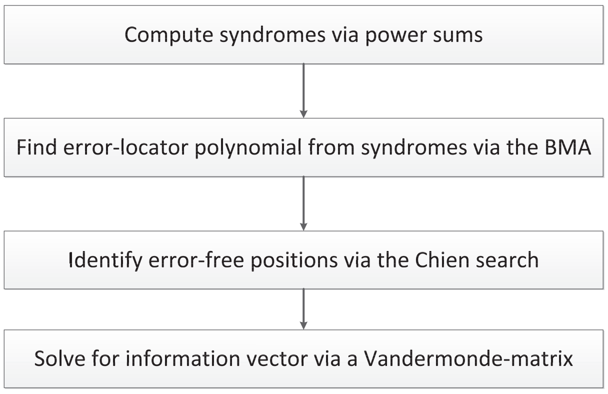

Compute syndromes via power sums
Find error-locator polynomial from syndromes via the BMA
Identify error-free positions via the Chien search
Solve for information vector via a Vandermonde-matrix

Fig. 1. Reed-Solomon decoding for an arbitrary codeword.

$$\Gamma(x) = \prod_{\ell=1}^{s} (x - \alpha^{i_\ell}) = \sum_{\ell=0}^{s} \Gamma_\ell x^\ell \tag{11}$$

$$Q_j = \sum_{i=0}^{s} \Gamma_{s-j} S_{j-i}. \tag{12}$$

3 Related Work

XOR-based erasure-resilient coding schemes have been applied to handle crash-stop failures. In practice, these codes encode and decode data faster than finite field-based RS codes [6]. However, they cannot handle Byzantine failures. Our progressive data retrieval coding scheme can, however, tolerate both crash-stop and Byzantine failures.

In the context of network storage for wireless sensor networks, randomized linear codes and fountain codes have been applied with the objective of a data collector retrieving data from each of k data sources by accessing any k out of n storage nodes [7], [8]. Although, up to $n-k$ crash-stop node failures can be tolerated, none of these methods are suitable for data reconstruction when Byzantine nodes exist.

Goodson et al. propose a coding scheme that handles Byzantine and crash-stop failures [2]. In particular, they design a consistency protocol for asynchronous environments via versions. Their protocol bares similarities with our progressive scheme in that it uses RS codes and accounts for data integrity. However, it does not address the communication and computation aspects of progressive data retrieval when there are failures in the distributed storage system. Instead, coded fragments are retrieved one time, and deemed one of three states: complete, incomplete, or repairable, depending on the degree of errors in the system.

4 Progressive Data Retrieval

Before presenting our proposed data reconstruction algorithm for distributed networked storage, we explain how a data file is stored in the network. We use the abstraction of a data node that is a data source, where networked storage nodes are used to achieve data persistence and any storage node may become crash-stop or Byzantine. Any RS decoding

```
Require: $(n, \hat{k})$-MDS encoding of original data, file
Ensure: Reconstructed data, file$'$, equals file or $\mathbf{0}$
 1: create an empty data-file, file$'$
 2: for each encoded group $[r_0, r_1, \ldots, r_{n-1}]$ of file do
 3:   $i \leftarrow \hat{k}$
 4:   $\boldsymbol{r}_i \leftarrow$ retrieve $[r_{j_0}, r_{j_1}, \ldots, r_{j_{\hat{k}-1}}]$ from any $\hat{k}$ nodes
 5:   repeat
 6:     $\boldsymbol{u} \leftarrow \boldsymbol{r}_i \hat{\boldsymbol{G}}^{-1}$
 7:     if CRC($\boldsymbol{u}$) is correct then
 8:       remove CRC header from $\boldsymbol{u}$ to obtain $\boldsymbol{u}_0$
 9:       append contents of $\boldsymbol{u}_0$ to file$'$
10:     end if
11:     repeat
12:       $i \leftarrow i + 2$
13:       retrieve $r_{i_1}, r_{i_2}$ from any two remaining nodes
14:       update $\boldsymbol{r}_i$ with $r_{i_1}$ and $r_{i_2}$
15:     until IRD$(\hat{k}, n, \frac{i-\hat{k}}{2}, \boldsymbol{r}_i) \neq \mathbf{0}$ or $i \geq n-1$
16:   until $i \geq n-1$
17:   file$' \leftarrow \mathbf{0}$
18:   exit
19: end for
```

Fig. 2. PDR.

algorithm may fail to identify a data-integrity violation. Since an RS decoding mechanism is at the core of our proposed algorithm, each data node adds a message authentication code to each data block it generates for data integrity. One-way hash functions such as the MD5 or SHA-1 can be applied [13], and we adopt CRC codes because they suffice to identify errors [9], [14]. Let the size of the original file before adding a CRC header be T_0 bits. If the size of the CRC overhead is r bits, then the probability of not detecting one error by the CRC code is $1/2^r$. The overhead is dependent only on T_0 and r and can be amortized by using large data blocks.

After adding the CRC, each data block is then partitioned into information symbols of length m bits and encoded using either the polynomial or matrix product provided in Section 2.2. The data node divides its data into $\lceil T/m \rceil$ symbols, where $T = T_0 + r$, and each symbol is an element in $GF(2^m)$. Unlike our previous decentralized scheme for distributed networked storage [15], the number of information symbols, $\hat{k}$, per encoding group of the proposed scheme is decoupled from the number of data nodes, k. Hence, we will employ an $(n, \hat{k})$-MDS code. Next, the set of $\lceil T/m \rceil$ symbols are partitioned into d information groups each of $\hat{k}$ symbols, where d is defined as shown in (13). CRC codes are added to every information group, and the data size T includes those bits added by CRC codes. The last information group may have less than $\hat{k}$ symbols. In this case, zeros are appended during the encoding procedure

$$d \triangleq \left\lceil \frac{\lceil T/m \rceil}{\hat{k}} \right\rceil. \tag{13}$$

We write the ith information group for encoding as the information vector, $\boldsymbol{u}_i = [u_{i0}, u_{i1}, \ldots, u_{i(\hat{k}-1)}]$, where $1 \leq i \leq d$. The data node encodes $\boldsymbol{u}_i$ into $\boldsymbol{c}_i = [c_{i0}, c_{i1}, \ldots, c_{i(n-1)}]$ with n symbols as (14), where $\boldsymbol{G}$ is the Vandermonde matrix in (5), after substituting $\hat{k}$ in for k. The data node then packs all $c_{i,j}$, $1 \leq i \leq d$, and sends them with the index, j, to storage node $(j+1)$ via the network

$$\boldsymbol{c}_i = \boldsymbol{u}_i \boldsymbol{G}. \tag{14}$$

For data reconstruction, a data collector must access a sufficient number of storage nodes to ensure data integrity. Among n storage nodes, let the number of crash-stop nodes that have not been accessed and the number of Byzantine nodes be s and v, respectively. The Vandermonde matrix, $\boldsymbol{G}$, in (14) is a generator matrix for an RS code and thus an error-erasure decoding algorithm can recover any codeword if there is no error in at least $\hat{k}$ encoded symbols [9]. Without loss of generality, assume that the data collector retrieves encoded symbols from storage nodes $j_0, j_1, \ldots j_{\hat{k}-1}$. If no error is present, the $\hat{k}$ symbols in the ith group of any data node can be recovered by solving the following system of linear equations:

$$[u_{i0}, u_{i1}, \ldots, u_{i(\hat{k}-1)}]\hat{\boldsymbol{G}} = [r_{ij_0}, r_{ij_1}, \ldots, r_{ij_{\hat{k}-1}}], \tag{15}$$

where

$$\hat{\boldsymbol{G}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{j_0} & \alpha^{j_1} & \cdots & \alpha^{j_{\hat{k}-1}} \\ (\alpha^{j_0})^2 & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_{\hat{k}-1}})^2 \\ & & \vdots & \\ (\alpha^{j_0})^{\hat{k}-1} & (\alpha^{j_1})^{\hat{k}-1} & \cdots & (\alpha^{j_{\hat{k}-1}})^{\hat{k}-1} \end{bmatrix}.$$

$\hat{\boldsymbol{G}}$ can be constructed by the primitive element and the index associated with r_{ij_ℓ}, $0 \leq \ell < \hat{k}$.

From Section 2, we know that RS codes can recover from any v errors, so long as $v \leq \lfloor \frac{n-\hat{k}-s}{2} \rfloor$. Therefore, if the number of compromised nodes is small, more erasures can be tolerated and less nodes can be accessed by assuming these nodes are unavailable.

Our proposed data reconstruction algorithm, PDR, is illustrated in Fig. 2. PDR proceeds in stages, where l errors are assumed at stage l and each stage corresponds to an attempt to reconstruct the original codeword (Lines 12-14). At the core of PDR is an efficient incremental RS decoding algorithm, IRD, illustrated in Fig. 3. IRD allows for data reconstruction even when some nodes are Byzantine or crash-stop, and it can utilize intermediate computation results from previous stages and decode incrementally to avoid unnecessary computations.

If a call to IRD does not successfully decode on Line 15, or the decoded codeword fails the CRC check on Line 7, then there must exist more Byzantine nodes than IRD can handle at this stage. In order to correct *one* more error, PDR retrieves *two* additional symbols given that the number of erasures allowed is reduced by two. Therefore, the total number of symbols retrieved at stage l is $\hat{k} + 2l$. Using a $(1{,}023{,}401)$-MDS code and a 1 percent error Byzantine node rate, 409.2 storage nodes are accessed on average (c.f.: Section 10). Hence, PDR makes $\lceil \frac{409.2-401}{2} \rceil = 5$ decoding requests to IRD. Consider a naive scheme that retrieves coded symbols from all reachable storage nodes and decodes only once. The naive scheme may incur less total computation than PDR, but has a high communication cost, since coded symbols must be retrieved from every node that is not crash-stop. PDR handles such tradeoffs between computation and communication more

```
Input: $(\hat{k}, n, \ell, \boldsymbol{r}_\ell)$
Output: $\hat{k}$ error-free symbols of $\boldsymbol{r}$, or $\mathbf{0}$
 1: if $\ell = 1$ then
 2:     compute $F_j$ via (27) for all $j \notin \mathbf{U}_0$
 3:     $\tilde{\Lambda}^{(0)}(x) \leftarrow 1$; $\tilde{\Omega}^{(0)}(x) \leftarrow 0$; $\Phi^{(0)}(x) \leftarrow 0$; $\Theta^{(0)}(x) \leftarrow 1$
 4: end if
 5: for $i = 1$ to $2$ do
 6:     $x_i^{(\ell)} \leftarrow \alpha^{j_i^{(\ell)}}$; $y_i^{(\ell)} \leftarrow S^{(\ell)}(x_i^{(\ell)})$
 7: end for
 8: for $i = 1$ to $2$ do
 9:     $b_i^{(\ell-1)} \leftarrow \tilde{\Omega}^{(\ell-1)}(x_i^{(\ell)}) - y_i^{(\ell)}\tilde{\Lambda}^{(\ell-1)}(x_i^{(\ell)})$
10:     if $b_i^{(\ell-1)} = 0$ then
11:         $\tilde{\Lambda}^T(x) \leftarrow \tilde{\Lambda}^{(\ell-1)}(x)$; $\Theta^T(x) \leftarrow (x - x_i^{(\ell)})\Theta^{(\ell-1)}(x)$
12:         $\tilde{\Omega}^T(x) \leftarrow \tilde{\Omega}^{(\ell-1)}(x)$; $\Phi^T(x) \leftarrow (x - x_i^{(\ell)})\Phi^{(\ell-1)}(x)$
13:     else
14:         $a_i^{(\ell-1)} \leftarrow \Theta^{(\ell-1)}(x_i^{(\ell)}) - y_i^{(\ell)}\Phi^{(\ell-1)}(x_i^{(\ell)})$
15:         $\Theta^T(x) \leftarrow (x - x_i^{(\ell)})\tilde{\Omega}^{(\ell-1)}(x)$
16:         $\Phi^T(x) \leftarrow (x - x_i^{(\ell)})\tilde{\Lambda}^{(\ell-1)}(x)$
17:         $\tilde{\Omega}^T(x) \leftarrow b_i^{(\ell-1)}\Theta^{(\ell-1)}(x) - a_i^{(\ell-1)}\tilde{\Omega}^{(\ell-1)}(x)$
18:         $\tilde{\Lambda}^T(x) \leftarrow b_i^{(\ell-1)}\Phi^{(\ell-1)}(x) - a_i^{(\ell-1)}\tilde{\Lambda}^{(\ell-1)}(x)$
19:     end if
20:     if rank $[\tilde{\Omega}^T(x), \tilde{\Lambda}^T(x)] >$ rank $[\Theta^T(x), \Phi^T(x)]$ then
21:         swap $[\tilde{\Omega}^T(x), \tilde{\Lambda}^T(x)] \leftrightarrow [\Theta^T(x), \Phi^T(x)]$
22:     end if
23:     if $i = 1$ then
24:         $\tilde{\Omega}^{(\ell-1)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\tilde{\Lambda}^{(\ell-1)}(x) \leftarrow \tilde{\Omega}^T(x)$
25:         $\Theta^{(\ell-1)}(x) \leftarrow \Theta^T(x)$; $\Phi^{(\ell-1)}(x) \leftarrow \Phi^T(x)$
26:     else
27:         $\tilde{\Omega}^{(\ell)}(x) \leftarrow \tilde{\Omega}^T(x)$; $\tilde{\Lambda}^{(\ell)}(x) \leftarrow \tilde{\Omega}^T(x)$
28:         $\Theta^{(\ell)}(x) \leftarrow \Theta^T(x)$; $\Phi^{(\ell)}(x) \leftarrow \Phi^T(x)$
29:     end if
30: end for
31: if $\deg(\tilde{\Lambda}^{(\ell)}(x)) = \ell$ then
32:     $\ell \leftarrow$ Chien-search$(\tilde{\Lambda}^{(\ell)}(x))$
33:     if $\ell \le n - \hat{k}$ and $\ell = \deg(\tilde{\Lambda}^{(\ell)}(x))$ then
34:         return any $\hat{k}$ error-free symbols of $\boldsymbol{r}_\ell$
35:     end if
36: end if
37: return $\mathbf{0}$
```

Fig. 3. IRD.

gracefully. For any coding group, if all symbols have been retrieved and PDR still cannot successfully decode, then Line 17 signals a decoding failure.

PDR achieves minimum communication cost as additional symbols are retrieved only when necessary. Moreover, we may substitute in the state of the art Berlekamp-Massey algorithm or either of the Welch-Berlekamp or euclidean algorithm in for IRD, on Line 15 to reconstruct the original codeword [9]. Our implementation results, however, show that IRD is up to 35 times faster than the BMA and we provide the reasons why in Section 5.

5 IRD

Compared to classical RS decoding, IRD utilizes intermediate computation results and decodes incrementally as more symbols become available. In this section, we characterize the operations in Fig. 3 and highlight the incremental computation process of IRD.

5.1 The Basic Algorithm

Given the received coded symbols $[r_0, r_1, \ldots, r_{n-1}]$ with erasures set to be zero, the generalized syndrome polynomial $S(x)$ can be calculated as follows:

$$\sum_{j=0}^{n-1} r_j \alpha^{jb} \frac{Q(x) - Q(\alpha^j)}{x - \alpha^j} = \sum_{j=0}^{n-1} \lambda_j \alpha^{jb} \frac{Q(x) - Q(\alpha^j)}{x - \alpha^j}, \tag{16}$$

where $Q(x)$ is an arbitrary polynomial with degree $n - \hat{k}$ [16]. Assume that v errors occur in unknown locations $j_1, j_2, \ldots, j_v$ and s erasures in known locations $m_1, m_2, \ldots, m_s$ of the received polynomial. Then,

$$e(x) = \sum_{\ell=1}^{v} e_{j_\ell} x^{j_\ell} \quad \text{and} \quad \gamma(x) = \sum_{\ell=1}^{s} \gamma_{m_\ell} x^{m_\ell}.$$

e_{j_ℓ} is the value of the ℓth error, $\ell = 1, \ldots, v$, and γ_{m_ℓ} is the value of the ℓth erasure, $\ell = 1, \ldots, s$. Since the received values in the erased positions are zero, $\gamma_{m_\ell} = -c_{m_\ell}$ for $\ell = 1, \ldots, s$. The decoding problem is to determine all j_ℓ, e_{j_ℓ}, and γ_{m_ℓ}. Let $\mathbf{E} = \{j_1, \ldots, j_v\}$, $\mathbf{M} = \{m_1, \ldots, m_s\}$, and $\mathbf{D} = \mathbf{E} \cup \mathbf{M}$. Clearly, $\mathbf{E} \cap \mathbf{M} = \emptyset$. A key equation for decoding RS codes is

$$\Lambda(x)S(x) = \Psi(x)Q(x) + \Omega(x), \tag{17}$$

where

$$\begin{aligned} \Lambda(x) &= \prod_{j \in \mathbf{D}} (x - \alpha^j) = \prod_{j \in \mathbf{E}} (x - \alpha^j) \prod_{j \in \mathbf{M}} (x - \alpha^j) \\ &= \Lambda_{\mathbf{E}}(x)\Lambda_{\mathbf{M}}(x) \end{aligned} \tag{18}$$

$$\Psi(x) = \sum_{j \in \mathbf{D}} \lambda_j \alpha^{jb} \prod_{i \in \mathbf{D}, i \ne j} (x - \alpha^i) \tag{19}$$

$$\Omega(x) = -\sum_{j \in \mathbf{D}} \lambda_j \alpha^{jb} Q(\alpha^j) \prod_{i \in \mathbf{D}, i \ne j} (x - \alpha^i). \tag{20}$$

If $2v + s < n - \hat{k} + 1$, then (17) has a unique solution $\{\Lambda(x), \Psi(x), \Omega(x)\}$. Instead of solving (17) by the euclidean or Berlekamp-Massey algorithm, we introduce a reduced key equation that can be solved by the WB algorithm [9], [16]. Let $\mathbf{Q} = \{j | Q(\alpha^j) = 0\}$. Let a set of coordinates $\mathbf{U} \subset \{0, 1, \ldots, n-1\}$ be defined by $\mathbf{U} = \mathbf{M} \cap \mathbf{Q}$. A polynomial $\Lambda_{\mathbf{U}}(x)$ is then defined by $\Lambda_{\mathbf{U}}(x) = \prod_{j \in \mathbf{U}} (x - \alpha^j)$, which is known for the receiver since $Q(x)$ and $\mathbf{M}$ are both known. Since $\Lambda_U(x)$ divides both $\Lambda(x)$ and $Q(x)$, according to (17), it also divides $\Omega(x)$. Hence, we have the following reduced key equation:

$$\tilde{\Lambda}(x)S(x) = \Psi(x)\tilde{Q}(x) + \tilde{\Omega}(x), \tag{21}$$

where

$$\begin{aligned} \Lambda(x) &= \tilde{\Lambda}(x)\Lambda_{\mathbf{U}}(x) \\ Q(x) &= \tilde{Q}(x)\Lambda_{\mathbf{U}}(x) \\ \Omega(x) &= \tilde{\Omega}(x)\Lambda_{\mathbf{U}}(x). \end{aligned}$$

$\tilde{\Lambda}(x)$ is a multiple of the error-locator polynomial $\Lambda_{\mathbf{E}}(x)$. The reduced key equation has a unique solution if (22) holds where $\deg(\cdot)$ is the degree of a polynomial, and $|\mathbf{U}|$ is the cardinality of $\mathbf{U}$. For all $j \in \mathbf{Q} \backslash \mathbf{U}$, by (21), we have (23), since $\tilde{Q}(\alpha^j) = 0$. Note that α^j is a sampling point and $S(\alpha^j)$ the sampled value for (23). The unique solution $\{\tilde{\Lambda}(x), \tilde{\Omega}(x)\}$ can then be found by the WB algorithm with time complexity $O((n - \hat{k} - |\mathbf{U}|)^2)$ [9]

$$\deg(\tilde{\Omega}(x)) < \deg(\tilde{\Lambda}(x)) < \frac{n - \hat{k} + 1 + s}{2} - |\mathbf{U}| \tag{22}$$

$$\tilde{\Lambda}(\alpha^j) S(\alpha^j) = \tilde{\Omega}(\alpha^j). \tag{23}$$

Once all coefficients of the errata polynomial are determined, all symbols having errors can be determined by successive substitution through the Chien search. When the solution of (21) is obtained, the errata values can be calculated. Since recovering the errata values is unnecessary for our problem, its computation is omitted. In summary, there are three steps in the decoding of RS codes that must be implemented. First, the sampled values of $S(\alpha^j)$ for $j \in \mathbf{Q} \backslash \mathbf{U}$ must be calculated. Second, the WB algorithm is performed based on the pairs $(\alpha^j, S(\alpha^j))$ in order to obtain a valid $\tilde{\Lambda}(x)$. If a valid $\tilde{\Lambda}(x)$ is obtained, then error locations are found by Chien search; otherwise, there is a decoding failure.

5.2 Incremental Computation

To reconstruct the original file, the data collector retrieves encoded symbols from any $\hat{k}$ storage nodes. Let $Q(x)$ in (16) be given by the product in (24), where the data symbol in position m_ℓ is erased. Moreover, these positions are erased just before the first iteration on Line 12 of PDR. Finally, the generator polynomial of the RS code encoded by (14) has $\alpha^{n-\hat{k}}, \alpha^{n-\hat{k}-1}, \ldots, \alpha$ as roots

$$Q(x) = (x - \alpha^{m_0})(x - \alpha^{m_1}) \cdots (x - \alpha^{m_{n-\hat{k}-1}}). \tag{24}$$

In the ℓth iteration, ℓ errors and $n - \hat{k} - 2\ell$ erasures are assumed in the codeword. Let $(j_1^{(\ell)} + 1)$ and $(j_2^{(\ell)} + 1)$ be the two storage nodes accessed in the ℓth iteration. From $\mathbf{U}_\ell$ in (25) and the fact that the WB algorithm is an iterative rational interpolation method, IRD determines values for $\tilde{\Lambda}^{(\ell)}(x)$ and $\tilde{\Omega}^{(\ell)}(x)$ that satisfy (26), where $S^{(\ell)}(x)$ is the generalized syndrome with $r_i = 0$ for all $r_i \in \mathbf{U}_\ell$

$$\begin{aligned} \mathbf{U}_0 &= \{m_0, \ldots, m_{n-\hat{k}-1}\} \\ \mathbf{U}_\ell &= \mathbf{U}_{\ell-1} \backslash \{j_1^{(\ell)}, j_2^{(\ell)}\} \end{aligned} \tag{25}$$

$$\tilde{\Lambda}^{(\ell)}(\alpha^j) S^{(\ell)}(\alpha^j) = \tilde{\Omega}^{(\ell)}(\alpha^j) \text{ for all } j \in \mathbf{U}_0 \backslash \mathbf{U}_\ell. \tag{26}$$

It has been shown that $\deg(\tilde{\Lambda}^{(\ell)}(x)) > \deg(\tilde{\Omega}^{(\ell)}(x))$ for any ℓ, a feature of the WB algorithm [9]. Thus, if $\deg(\tilde{\Lambda}^{(\ell)}(x)) < \frac{n-\hat{k}+1+|\mathbf{U}_\ell|}{2} - |\mathbf{U}_\ell| = \ell + 1/2$, then a unique solution will exist via (22). By the definition of the generalized syndrome polynomial in (16), for $i \in \mathbf{U}_0 \backslash \mathbf{U}_\ell$, we have

$$\begin{aligned} S^{(\ell)}(\alpha^i) &= \sum_{j=0}^{n-1} r_j \alpha^j \frac{Q(\alpha^i) - Q(\alpha^j)}{\alpha^i - \alpha^j} \\ &= \sum_{j=0, j \notin \mathbf{U}_0}^{n-1} r_j \alpha^j \frac{Q(\alpha^j)}{\alpha^j - \alpha^i} + r_i \alpha^i Q'(\alpha^i) \\ &= \sum_{j=0, j \notin \mathbf{U}_0}^{n-1} \frac{F_j}{\alpha^j - \alpha^i} + r_i \alpha^i Q'(\alpha^i), \end{aligned} \tag{27}$$

where $Q'(x)$ is the derivative of $Q(x)$ and $F_j = r_j \alpha^j Q(\alpha^j)$. Note that (28) follows, and $S^{(\ell)}(\alpha^i)$ is not related to any r_j, where $j \in \mathbf{U}_0$ and $j \neq i$

$$Q'(\alpha^i) = \prod_{j \in \mathbf{U}_0, m_j \neq i} \alpha^i - \alpha^{m_j}. \tag{28}$$

Hence, $S^{(\ell-1)}(\alpha^i) = S^{(\ell)}(\alpha^i)$ for all $i \in \mathbf{U}_0 \backslash \mathbf{U}_{\ell-1}$. This fact implies that all sampled values in previous iterations can be directly used in current iteration of the WB algorithm.

For any two polynomials, $N(x)$ and $W(x)$, we define the $\mathrm{rank}[N(x), W(x)]$ as $\max[2 \cdot \deg(W(x)), 1 + 2 \cdot \deg(N(x))]$. IRD returns $\hat{k}$ error-free symbols whenever a codeword can be successfully decoded and **0** otherwise. Specifically, if the degree of the error-locator polynomial does not equal to the number of roots determined by the Chien in Line 33, then the codeword was not successfully decoded.

6 Performance Analysis

In this section, we quantitatively analyze PDR and compare it to Decentralized Erasure Codes (DEC) and Decentralized Fountain Codes (DFC) [7], [8]. Our network consists of k data nodes and n storage nodes, and each data node requires robust distributed storage of its file of size T_0. We proceed first by briefly describing each coding scheme. Then, we characterize each one over the following: the total storage and its associated overhead, communication and computation corresponding to data dissemination and data retrieval when nodes fail, and the security strength.

DEC. Each storage node selects random and independent coefficients in a finite field $\mathbb{F}_q$, and stores a linear combination of all the received data from the k data nodes. Randomized linear codes are used, where each data node routes its packet to at least $\frac{n}{k} \log k$ storage nodes. A data collector queries at least k storage nodes for data retrieval. In the absence of Byzantine storage nodes, data retrieval is successful with high likelihood if the finite field size is large.

DFC. Each storage node s_i, chooses a degree d_i, defined as the number of data symbols required to form a linear combination. Node s_i then XORs d_i data symbols, from d_i arbitrarily chosen data nodes, where each data symbol is selected from $\mathbb{F}_q$. DFC trades off communication for computation in that decoding requires more than k storage nodes to be contacted, though both encoding and decoding computations are linear in the number of original symbols. We will show that the probability of successful decoding is given by a parameter, δ.

PDR. The k data nodes collectively generate kT_0 bits, T_0 bits per data node. Since each PDR data node adds r CRC bits to its data, the file size at each data node is $T = T_0 + r$. In our analysis, the product of an m_1 and m_2-bit symbol requires $m_1 m_2$ XORs, and an addition requires only m_1 XORs, where the field of operation for the symbols is $\mathbb{F}_{m_1}$ and $m_1 \geq m_2$. DEC and DFC both have one data symbol per data node for each coding group, while PDR encodes k data symbols per data node for each group. Consequently, each data node in DEC/DFC and PDR has $\lceil \frac{T_0}{\log q} \rceil$ and d groups per data node, respectively, where d is given in (13). We show later in this section that, although fountain codes have low encoding and decoding computational complexities, PDR offers a much lower communication cost for data retrieval.

6.1 Storage

The storage complexity for DEC is $\lceil \frac{T_0}{\log q} \rceil \cdot \log q \approx T_0$ bits for each storage node, since each storage symbol contains $\log q$ bits, the field of operation is $\mathbb{F}_q$, and there are $\lceil \frac{T_0}{\log q} \rceil$ groups. Since each storage node stores linear combination coefficients, where $\log k$ data nodes map to one storage node, the overhead to store the coefficients, per storage node is

$$\log k \cdot \left(\log q + \lceil \log k \rceil + \left\lceil \log \left\lceil \frac{T_0}{\log q} \right\rceil \right\rceil \right)$$
$$\approx \log k \log q \text{ bits.}$$

The outer $\log k$ term corresponds to the set of data nodes mapped to a storage node, while the inner one corresponds to the bits required to identify any mapped data node. The last term identifies the coding group.

Similar to DEC, the storage complexity for DFC per storage node is also T_0 bits. The overhead complexity per storage node is given by

$$\log \frac{k}{\delta} \cdot \left(\lceil \log k \rceil + \left\lceil \log \left\lceil \frac{T_0}{\log q} \right\rceil \right\rceil \right) \text{bits.}$$

The derivation is similar to that of DEC except that the average degree of a storage symbol is $\log \frac{k}{\delta}$ and there are no linear combination coefficients, since every linear combination is simply an XOR of a set of data symbols.

Each data node in PDR encodes its own data with $\hat{k}$ symbols per group to all storage nodes. Given there are d groups and the application of the CRC, the per-node storage complexity is T bits. Because PDR utilizes a code structure known to all storage nodes, its overhead complexity per storage node is $\lceil \log d \rceil$, the bits required to index each group.

6.2 Data Dissemination

The DEC dissemination cost is

$$k \cdot \frac{n}{k} \log k \cdot \left\lceil \frac{T_0}{\log q} \right\rceil \cdot \log q \approx n T_0 \log k \text{bits,}$$

given that there are k data nodes, and each one sends its data $\frac{n}{k} \log k$ times, for all $\lceil \frac{T_0}{\log q} \rceil$ groups. Similarly, the dissemination cost for DFC is

$$n \cdot \log \frac{k}{\delta} \cdot \left\lceil \frac{T_0}{\log q} \right\rceil \cdot \log q \approx n T_0 \log \frac{k}{\delta} \text{bits,}$$

since there are n storage nodes, and each one stores $\log \frac{k}{\delta}$ symbols on average. Unlike the decentralized erasure and fountain codes, PDR does not replicate transmissions to storage nodes. Therefore, its dissemination matches its storage cost.

6.3 Encoding

Assume a software implementation on field operations without lookup tables. For DEC, a linear combination coefficient and a data symbol contain $\log q$ bits. Hence, the computation cost of encoding per storage node is

$$(\log^2 q) \cdot \log k \cdot \left\lceil \frac{T_0}{\log q} \right\rceil \text{XORs,}$$

since the cost of a multiplication is $\log^2 q$ XORs, each symbol is the result of $\log k$ linear combinations, for all groups.

For DFC, the encoding complexity per storage node is

$$\log q \cdot \log \frac{k}{\delta} \cdot \left\lceil \frac{T_0}{\log q} \right\rceil \text{XORs,}$$

since the cost of an XOR of two $\log q$-bit symbols is $\log q$ XORs, and each encoded symbol is the XOR of $\log \frac{k}{\delta} \log q$-bit data symbols.

For PDR, each finite field multiplication costs m^2 XORs. Encoding for all groups and each data node yields

$$\hat{k} \cdot (m^2 + m) \cdot n \cdot d \text{ XORs,}$$

because of the matrix multiplications at each data node.

6.4 Data Retrieval: Crash-Stop Nodes

A data collector for DEC must receive the coefficients stored in at least k storage nodes for data reconstruction. Hence, the retrieval cost is simply $k(\lceil \frac{T_0}{\log q} \rceil \log q + \log k \log q)$ bits in total. Similarly for DFC, at least k storage nodes are required for data reconstruction. The communication cost to retrieve any data node's file is

$$T_0 \left(1 + \frac{\log^2 \frac{k}{\delta}}{\sqrt{k}} \right) \text{bits,}$$

since $k + \sqrt{k} \log^2 \frac{k}{\delta}$ symbols must be retrieved [7]. Similar to DEC, the retrieval cost for PDR is $\hat{k} d k m \approx kT$. Moreover, PDR can allow for partial data retrieval. In particular, since k storage nodes can store the original data in its initial form, any one of these storage nodes can be requested to retrieve $1/k$-portion for any group.

6.5 Data Retrieval: Crash-Stop and Byzantine Nodes

In this section, the worst case is assumed: validation of the degree of the error-locator polynomial and the Chien search condition, Lines 31 and 33, respectively, of IRD do not terminate the PDR prematurely.

We provide a probabilistic analysis of the cost of communication by determining the number of stages the algorithm requires, and the probability of successful decoding. Given n storage nodes and an $(n, \hat{k})$ – MDS code, the minimum and maximum number of storage nodes to access in the proposed scheme is $\hat{k}$ and n, respectively. We assume that the CRC code always detects an error if it occurs. Without loss of generality, we assume that all failures are Byzantine failures, since s crash-stop failures can be easily modeled by replacing n with $n - s$. An important metric of the decoding efficiency is the average number of accessed storage nodes when the probability of compromising each storage node is p. Failure to recover data correctly may occur in two cases: 1) $v > n - \hat{k}$, where there is an insufficient number of non-Byzantine storage nodes that are not crash-stop, 2) $\lfloor \frac{n - \hat{k}}{2} \rfloor < v < n - \hat{k}$, where the sequential order of data accesses from the data collector determines whether PDR can successfully decode. As an example, suppose the first v nodes accessed are all Byzantine. Then, successful decoding is impossible, PDR terminates, and the communication cost is at its maximum since all reachable nodes are accessed. The main result is summarized in Theorem 1, and its proof is in the Appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.67.

Theorem 1. *For any codeword, the average number of storage nodes accessed and probability of successful decoding for the PDR algorithm is given in*

$$\begin{aligned}\bar{N}(n,\hat{k}) &= \sum_{v=0}^{n-\hat{k}} \binom{n}{v} p^v (1-p)^{n-v} \\ &\quad \sum_{i=0}^{\min(v,\lfloor\frac{n-\hat{k}}{2}\rfloor, n-v-\hat{k})} (\hat{k}+2i) \frac{\binom{n-v}{i+\hat{k}-1}\binom{v}{i}}{\binom{n}{2i+\hat{k}-1}} \cdot \frac{\hat{k}}{i+\hat{k}} \\ &\quad \cdot \frac{n-v-(i+\hat{k}-1)}{n-(2i+\hat{k}-1)} \\ &\quad + \sum_{v=0}^{n-\hat{k}} n\binom{n}{v} p^v (1-p)^{n-v} \left(1 - \sum_{i=0}^{\min(v,\lfloor\frac{n-\hat{k}}{2}\rfloor, n-v-\hat{k})} \right. \\ &\quad \left. \frac{\binom{n-v}{i+\hat{k}-1}\binom{v}{i}}{\binom{n}{2i+\hat{k}-1}} \cdot \frac{\hat{k}}{i+\hat{k}} \cdot \frac{n-v-(i+\hat{k}-1)}{n-(2i+\hat{k}-1)} \right) \\ &\quad + \sum_{v=n-\hat{k}+1}^{n} n\binom{n}{v} p^v (1-p)^{n-v}\end{aligned} \tag{29}$$

$$\begin{aligned}\Pr_{suc}(n,\hat{k}) &= \sum_{v=0}^{n-\hat{k}} \binom{n}{v} p^v (1-p)^{n-v} \sum_{i=0}^{\min(v,\lfloor\frac{n-\hat{k}}{2}\rfloor, n-v-\hat{k})} \\ &\quad \frac{\binom{n-v}{i+\hat{k}-1}\binom{e}{i}}{\binom{n}{2i+\hat{k}-1}} \cdot \frac{\hat{k}}{i+\hat{k}} \cdot \frac{n-v-(i+\hat{k}-1)}{n-(2i+\hat{k}-1)}.\end{aligned} \tag{30}$$

6.6 Decoding

If there are only crash-stop failures, then the decoding complexity for DEC to reconstruct the entire data file is

$$\frac{T_0}{\log q} \cdot k^3 \cdot \log^2 q = k^3 T_0 \log q \text{ XORs},$$

given that one multiplication costs $\log^2 q$ XORs and there is a matrix inversion for each group. For DFC codes, we have

$$\frac{T_0}{\log q} \cdot k \log\frac{k}{\delta} \cdot \log q = kT_0 \log\frac{k}{\delta} \text{ XORs},$$

because DFC uses the $k\log\frac{k}{\delta}$ belief propagation decoding algorithm.

Detection and correction of errors, due to Byzantine nodes, are computationally infeasible for both the DEC and DFC codes. In particular, if we apply a CRC code for error detection in a DEC/DFC code, a data-collector must enumerate all possible k symbols from k corresponding storage nodes until the original data can be reconstructed correctly. Therefore, DEC and DFC are both unsuitable for robust distributed storage systems with Byzantine nodes. PDR is the only erasure coding scheme to efficiently reconstruct data in a network of Byzantine nodes.

For PDR, we generalize the decoding complexity by allowing for Byzantine and crash-stop failures. In testing the CRC code in Section 6.5, one polynomial division is performed. Since the degree of the dividend and divisor is $T-1$ and r, respectively, the computation complexity for testing the CRC code is $O(Tr)$. Let v be the number of Byzantine nodes, or errors, when PDR terminates. In the ℓth iteration or call of the IRD procedure, ℓ errors are assumed and the number of erasures is $n-\hat{k}-2\ell$. In the first iteration only, all F_j values are computed. The associated computation complexity is $O(\hat{k}(n-\hat{k}))$, since there are $\hat{k}$ values F_j to compute, and each is a product of $n-\hat{k}$ terms. Then, for every iteration, IRD computes two syndrome values, determined by the F_j values computed. In the next iteration, two more symbols are added to (16). Hence, the updated syndrome values can be obtained by an extra $O(\hat{k})+O(n-\hat{k})$ computations. To determine the error-locator polynomial, the loop on (Line 8) of IRD is executed twice, with complexity $O(\ell)$. Since we consider a software implementation, the Chien search can be implemented substituting powers of α into the error-locator polynomial. The Chien search tests at most $\hat{k}+\ell$ positions to locate $\hat{k}$ error-free positions such that it takes $O((\hat{k}+\ell)\cdot\ell)$ computations. To invert $\hat{\boldsymbol{G}}$, we use an algorithm closely related to the Lagrangian interpolation formula with complexity, $O(\hat{k}^2)$ [17].

In summary, the computation in the ℓth iteration, for any $\ell > 1$, is given by (31). Summing over all v iterations and accounting for the cost of determining the F_j values on Line 2 of IRD, we have (32):

$$L_v(\ell) = O(\hat{k}^2) + O(n-\hat{k}) + O(\hat{k}\ell + \ell^2) \tag{31}$$

$$\begin{aligned}L_v &= O(v\hat{k}^2) + O(\hat{k}(n-\hat{k})) + O(v^2\hat{k}) \\ &\quad + O(v(n-\hat{k})) + O(v^3).\end{aligned} \tag{32}$$

Since v is bounded above by the error-correction capacity $(n-\hat{k})/2$, the decoding complexity is no greater than $O(\hat{k}(n-\hat{k})^2)$. When v is small, the term corresponding to computing syndromes, $O(\hat{k}(n-\hat{k}))$ dominates. PDR requires kd iterations to decode T. Hence, for the entire data, the decoding complexity is Tmk^2 XORs in the absence of Byzantine storage nodes. When v nodes are Byzantine, the complexity is given by

$$Tm(vk^2 + k(n-k) + v^2k + v(n-k) + v^3) \text{ XORs}. \tag{33}$$

6.7 Security

For generalized content distribution, decentralized erasure and fountain codes achieve subquadratic computational decoding complexity, even in the midst of Byzantine nodes. Indeed, fountain codes can support incremental decoding [18]. For distributed storage systems, however, none of these codes support data reconstruction in polynomial time when a subset of the networked storage nodes are Byzantine. In this section, we address the security strength of our proposed algorithm, PDR when there are Byzantine nodes.

Without loss of generality, assume that the Byzantine nodes are $j_0, j_1, \ldots, j_{v-1}$. Furthermore, these nodes collude to forge data that can pass a CRC test. Suppose $\boldsymbol{f}_i$ is the portion of the forged vector of information symbols in the ith group that can pass the CRC test. Let $\hat{\boldsymbol{f}}_i = \boldsymbol{f}_i + \boldsymbol{u}_i$, where $\boldsymbol{u}_i$ is the original vector of information symbols in the ith group after successful decoding. Then, (34) holds, since PDR leverages

TABLE 1
Performance Analysis

	DEC	DFC	PDR
Storage	nT_0	nT_0	nT
Storage overhead	$n\log(q+k)$	$n\log\left(\frac{k}{\delta}+k\right)$	0
Dissemination	$nT_0\log k$	$nT_0\log\frac{k}{\delta}$	nT
Crash-stop retrieval	$kT_0+k\log(q+k)$	$T_0\left(k+\sqrt{k}\log^2\frac{k}{\delta}\right)$	kT
Encoding	$nT_0\log(q+k)$	$nT_0\log\frac{k}{\delta}$	nTm
Decoding	$k^3T_0\log q$	$kT_0\log\frac{k}{\delta}$	k^2Tm
Error detection/correction	no	no	yes

Reed-Solomon codes. $\boldsymbol{c}_i$ is the original codeword and $\boldsymbol{v}$ is the compromised data due to the Byzantine nodes

$$\boldsymbol{f}_i G = (\hat{\boldsymbol{f}}_i + \boldsymbol{u}_i)G = \hat{\boldsymbol{f}}_i G + f u_i \boldsymbol{G} = \boldsymbol{v} + \boldsymbol{c}_i. \qquad (34)$$

Let the number of nonzero symbols in $\boldsymbol{v}$ be h. It is clear that $h \geq n - \hat{k} + 1$, where $n - \hat{k} + 1$ is the minimum Hamming distance of RS code, since $\boldsymbol{v}$ is a codeword. In the worst case, h matches $n - \hat{k} + 1$. An attacker must compromise a subset of storage nodes, such that they store the corresponding encoded symbols in $\boldsymbol{f}_i G$, the codeword corresponding to the forged vector of information symbols.

If the attacker compromises $\hat{k}$ storage nodes, the attacker can forge the data successfully, when the data collector accesses these compromised storage nodes using PDR. Suppose the attacker compromises $b < \hat{k}$ storage nodes. Using PDR, when $h - b$ is no more than $\lfloor\frac{n-\hat{k}}{2}\rfloor$, PDR can still decode the received vector to $\boldsymbol{f}_i G$. The minimum possible value of b is $\lceil\frac{n-\hat{k}+2}{2}\rceil$; hence, the security strength, the maximum number of compromised nodes that cannot forge the information data even when they collude, for IRD is $\min\{\hat{k}, \lceil\frac{n-\hat{k}+2}{2}\rceil\} - 1$. Moreover, any other cryptographic hash function to increase the security strength against forging can be utilized. In particular, the 32-bit CRC code can be replaced with a 128-bit MD5 code.

6.8 Summary

Table 1 summarizes the analysis in this section. The bits required to identify a coding group is negligible in the overhead of all three schemes. Clearly, PDR has the minimum dissemination cost. The encoding and decoding of DFC can be made highly efficient, but only at the expense of a high communication cost, as governed by δ. Since the CRC overhead of PDR is negligible (c.f.: Section 4), PDR achieves the minimum cost for data dissemination and retrieval.

7 Implementation and Evaluation

In Sections 7.1 and 7.2, we corroborate our findings from Theorem 1 with network simulations. Then, we compare PDR with the state-of-the-art error-erasure decoding algorithm for distributed storage systems containing both crash-stop and Byzantine storage nodes.

7.1 Numerical Results

We verify the correctness of the analytical derivations in Theorem 1 using Monte-Carlo simulations implemented in Matlab.

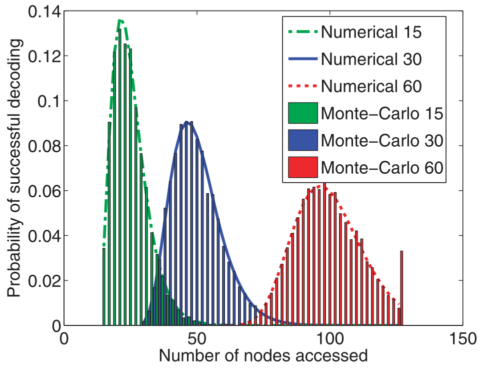


Fig. 4. Data retrieval cost for a $(127, \hat{k})$-MDS; $k = 30$. Byzantine node rate is 0.2 and $\hat{k} \in \{\frac{k}{2}, k, 2k\}$.

7.1.1 Successful Decoding

Fig. 4 shows the distribution of the number of storage nodes accessed when PDR terminates. The bars depict histograms from Monte-Carlo simulations with 5,000 runs, and the curves represent the numerical results from our analytical model. We choose $n = 127$, $k = 30$, and $p = 0.2$. From Fig. 4, the analytical model is consistent with the Monte-Carlo simulations. Moreover, increasing $\hat{k}$ reduces the storage requirements, but increases the cost of data retrieval, corroborating our analysis in Section 6.

7.1.2 Retrieval

We use 1,023 storage nodes to simulate the communication cost associated with data retrieval, and vary the number of information symbols, $\hat{k}$, from 101 to 401. We also vary the Byzantine node rate p from 0 to 0.3, while keeping the number of data nodes constant. Fig. 5 shows the increasing communication cost as the probability of failures increases. The number of crash-stop failures is set to zero, and all Byzantine failures result in incorrect data. Clearly, when the Byzantine failure rate is small, the communication cost is close to $\hat{k}$. And when p increases, the communication cost monotonically increases, as expected. We also analyze the success rate of decoding. And observe that for $\hat{k} \in \{101, 201, 301\}$ and $p \in \{0, 0.05, \ldots, 0.3\}$, decoding will always be successful. However, for $\hat{k} = 401$, decoding is always successful *only* for $p \in \{0, 0.05, \ldots, 0.25\}$. When $p = 0.3$, the probability of successful decoding is only 60 percent.

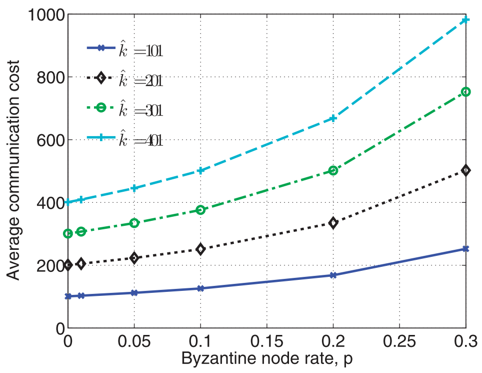


Fig. 5. Number of storage nodes accessed as a function of the probability of malicious attacks for a $(1{,}023, \hat{k})$-MDS; $k = 101$.

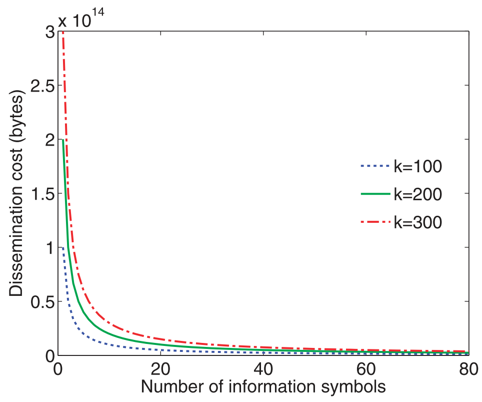


Fig. 6. The impact of the number of information symbols, $\hat{k}$, and data nodes, k, on the communication cost of dispersing storage data.

7.2 Impact of $\hat{k}$ and k

The number of information symbols, $\hat{k}$, is decoupled from the number of data nodes k; this is in contrast to our previous decentralized networked storage scheme [15]. Hence, $\hat{k}$ and k can be tuned to balance the trade off between dissemination and retrieval costs. Fig. 6 shows that the dissemination cost of a 1 GB data item is a function of $\hat{k}$ and k; the field size for the RS codes is 1,024, and there are $n = 1{,}000$ storage nodes. For any value of k, increasing $\hat{k}$ reduces the dissemination cost, as the level of redundancy decreases.

In wireless sensor networks where data nodes are power limited, the data collector typically has no power constraint. Hence, the cost to disseminate coded symbols from data nodes to storage nodes should be minimal. This can be done by choosing $\hat{k}$ so that $\hat{k} > k$. The total dissemination cost for an item of size T, is given in (35) and is proportional to the ratio of k and $\hat{k}$, also corroborating our analysis in Section 6. If $\hat{k}$ is twice the value of k then the cost is half the cost corresponding to $\hat{k}$ equal to k. The data retrieval cost is then doubled

$$knm\left\lceil\frac{\lceil T/m\rceil}{\hat{k}}\right\rceil. \tag{35}$$

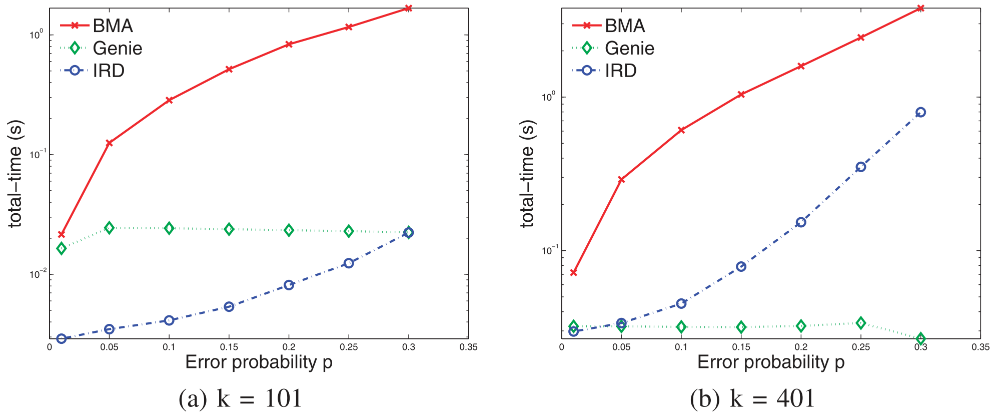
7.3 Implementation Setup

We have implemented the proposed and baseline algorithms, where each data node's information is a memory buffer in a single machine having 2.66 GHz Intel Xeon CPU, 4,096 KB cache and 2 GB RAM. A randomly generated message is first partitioned into either 101 or 401 information symbols and then encoded into $n = 1{,}023$ coded symbols of length 10,230 bits, and the finite field size is 1,024. A stored symbol is independently corrupted with error probability p.

Comparing our error-erasure code to either decentralized or fountain erasure codes for error correction performance is pointless, since these codes cannot feasibly guarantee data availability in the presence of erroneous symbols. Instead, we provide a comparison to the state-of-the-art error-erasure decoding algorithm, BMA. Here, we progressively retrieves data from each storage node and performs decoding until either the decoded symbols passes the CRC test or cannot be decoded. The work of Goodson et al. employs this algorithm [2]. In addition, we consider a genie form of BMA, BMA-genie, that assumes full knowledge of the number of symbols required for successful decoding. It decodes only once after retrieving the sufficient number of symbols. BMA-genie cannot be implemented in practice and is included for comparison only.

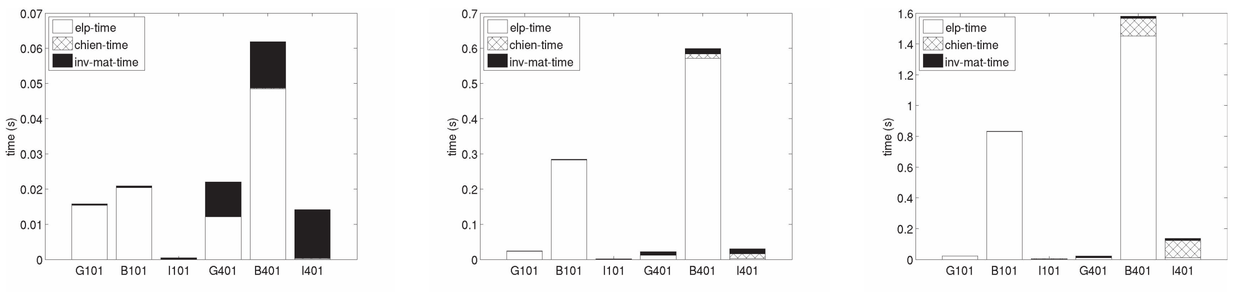
7.4 Total Computation Time

Figs. 7a and 7b illustrates the computation time, in log scale, spent in decoding when $k = 101$ and $k = 401$, respectively; in these simulations, $\hat{k}$ matches k. The storage overhead n/k is 10.13 and 2.55 with the maximum number of errors correctable being 461 and 311. From Fig. 7, we observe that the BMA and IRD computation time increases as p increases, but the rate of growth of IRD is much slower. In Fig. 7a, where the code redundancy is high, IRD is faster than the genie-aided BMA. This is because the cost, $O((n-k)^2)$, of computing erasure polynomials dominates the decoding time when p is small. Unlike BMA-genie, IRD does not compute erasure polynomials. In our implementation, the Byzantine node rate is 0.2, and there are 200 data nodes. We employ a $(1{,}023, \hat{k}) -$ MDS code, where $\hat{k} \in \{50, 100, \ldots, 550\}$. The computation cost for encoding is independent of $\hat{k}$, and negligible compared to the computation cost for decoding cost. The nonlinear increase in the decoding complexity supports our theoretical derivations in (32).



(a) k = 101 (b) k = 401

Fig. 7. Decoding computation time as a function of the Byzantine node rate, p. Each coding group utilizes a $(1{,}023, k) -$ MDS code. For the given values of k, no algorithm can successfully decode when $p > 0.3$.



(a) $p = 1\%$ (b) $p = 10\%$ (c) $p = 20\%$

Fig. 8. Average computational time breakdown for decoding one $(1{,}023, k) - \text{MDS}$ codeword, $k \in \{101, 401\}$. Because IRD progressively decodes, its performance does not deteriorate with an increasing Byzantine node rate, p.

7.5 Decoding Breakdown

We break down the decoding computation time to understand the dominant operations as the error probability increases. The breakdown includes the times to find the error-locator polynomial, find the error locations, and solve for the information polynomial. We represent these three times as *elp-time*, *chien-time*, and *inv-mat-time*, respectively. This breakdown is also illustrated in Fig. 1, where the first and second blocks correspond to the elp-time, the third block to the chien-time, and the fourth block to the inv-mat-time.

In Fig. 8a, when the error probability is low, the computation of error-locator polynomials dominates for small $\hat{k}$, while the matrix inversion time becomes significant when $\hat{k}$ is large. In our implementation, the cost of a matrix inversion is quadratic in the number of symbols decoded. Although the Chien search has the highest asymptotic complexity, it has the shortest running time. When the error probability is high, Figs. 8b and 8c show that the computation of error-locator polynomials dominates except in IRD.

The computation time in matrix inversion is negligible and on the order of tens of milliseconds in BMA and IRD; it is comparable to that of the BMA-genie, which assumes knowledge of the number of errors in advance, and thus performs a matrix inversion only once. The negligible running time in matrix inversion occurs because the decoding algorithm is likely to fail during or prior to the Chien search on Line 33 of IRD; this holds even if the Byzantine node rate is high. In most cases, BMA and IRD perform a matrix inversion only once.

8 Conclusions

We have designed a highly computation-efficient and communication-optimal algorithm, PDR, for distributed storage systems, where storage nodes can be either Byzantine or crash stop. The communication and computation costs for data retrieval are minimized by utilizing intermediate computation results and retrieving only the minimum data required for successful data reconstruction, respectively.

Our in-depth analysis shows that decentralized fountain codes and PDR are most suitable to networks without Byzantine storage nodes because of the associated minimal computation cost for fountain codes, and the minimal data retrieval cost for PDR. However, when Byzantine nodes exist, only PDR is suitable since it allows for error detection and correction.

Acknowledgments

Han's work was supported by the National Science Council of Taiwan, under grants NSC 96-2221-E-305-002-MY3 and his visit to LIVE lab at University of Texas at Austin. Omiwade and Zheng's work is supported in part by US National Science Foundation (NSF) CNS 0546391.

References

[1] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures," *Ad Hoc Networks,* vol. 1, nos. 2/3, pp. 293-315, 2003.
[2] G.R. Goodson, J.J. Wylie, G.R. Ganger, and M.K. Reiter, "Efficient Byzantine-Tolerant Erasure-Coded Storage," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 135-144, July 2004.
[3] H. Krawczyk, "Distributed Fingerprints and Secure Information Dispersal," *PODC '93: Proc. 12th Ann. ACM Symp. Principles of Distributed Computing,* pp. 207-218, 1993.
[4] T.S.D. Sheet, "Moteiv, San Francisco, CA, 2006," 2004.
[5] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks," *Proc. IEEE INFOCOM '94,* vol. 2, pp. 680-688, June 1994.
[6] J.S. Plank, J. Luo, C.D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage," *FAST '09: Proc. Seventh Conf. File and Storage Technologies,* pp. 253-265, 2009.
[7] Y. Lin, B. Liang, and B. Li, "Data Persistence in Large-Scale Sensor Networks with Decentralized Fountain Codes," *Proc. 26th IEEE INFOCOM,* pp. 6-12, 2007.
[8] A.G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE Trans. Information Theory,* vol. 52, no. 6, pp. 2809-2816, June 2006.
[9] T.K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms.* John Wiley & Sons, 2005.
[10] S. Lin and D.J. Costello Jr., *Error Control Coding: Fundamentals and Applications,* second ed. Prentice-Hall, 2004.
[11] T.K. Truong, W.L. Eastman, I.S. Reed, and I.S. Hsu, "Simplified Procedure for Correcting both Errors and Erasures of Reed-Solomon Code Using Euclidean Algorithm," *Proc. IEE,* vol. 135, no. 6, pp. 318-324, Nov. 1988.
[12] S.-L. Shieh, S.-G. Lee, and W.-H. Sheen, "A Low-Latency Decoder for Punctured/Shortened Reed-Solomoncodes," *Proc. IEEE Int'l Symp. Personal, Indoor and Mobile Radio Comm.,* pp. 2547-2551, Sept. 2005.
[13] J. Kurose and K. Ross, *Computer Networks: A Top Down Approach Featuring the Internet.* Pearson Addison Wesley, 2005.
[14] I.S. Reed and X. Chen, *Error-Control Coding for Data Networks.* Kluwer Academic, 1999.
[15] Y.S. Han, S. Omiwade, and R. Zheng, "Survivable Distributed Storage with Progressive Decoding," *Proc. IEEE INFOCOM,* pp. 1-5, Mar. 2010.

[16] K. Araki, M. Takada, and M. Morii, "On the Efficient Decoding of Reed-Solomon Codes Based on GMD Criterion," *Proc. Int'l Symp. Multiple-Valued Logic,* pp. 138-145, May 1992.

[17] H. William, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge Univ. Press, 1988.

[18] M.N. Krohn, M.J. Freedman, and D. Mazieres, "On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution," *Proc. IEEE Symp. Security and Privacy,* pp. 226-240, May 2004.

Yunghsiang S. Han (S'90-M'93-SM'08-F'11) received the BSc and MSc degrees in electrical engineering from the National Tsing Hua University, Hsinchu, Taiwan, in 1984 and 1986, respectively, and the PhD degree from the School of Computer and Information Science, Syracuse University, NY, in 1993. He was, from 1986 to 1988, a lecturer at Ming-Hsin Engineering College, Hsinchu, Taiwan. He was a teaching assistant from 1989 to 1992, and a research associate in the School of Computer and Information Science, Syracuse University from 1992 to 1993. He was, from 1993 to 1997, an associate professor in the Department of Electronic Engineering at Hua Fan College of Humanities and Technology, Taipei Hsien, Taiwan. He was with the Department of Computer Science and Information Engineering at National Chi Nan University, Nantou, Taiwan, from 1997 to 2004. He was promoted to professor in 1998. He was a visiting scholar in the Department of Electrical Engineering at University of Hawaii at Manoa, HI, from June to October 2001, the SUPRIA visiting research scholar in the Department of Electrical Engineering and Computer Science and CASE center at Syracuse University, NY from September 2002 to January 2004, and the visiting scholar in the Department of Electrical and Computer Engineering at University of Texas at Austin, TX, from August 2008 to June 2009. He was with the Graduate Institute of Communication Engineering at National Taipei University, Taiwan, from August 2004 to July 2010. From August 2010, he is with the Department of Electrical Engineering at National Taiwan University of Science and Technology. His research interests are in error-control coding, wireless networks, and security. He was a winner of the 1994 Syracuse University Doctoral Prize. He is a fellow of the IEEE.

Soji Omiwade received the BSc degree in computer science and the BSc degree in physics at the Abilene Christian University, in 2005, where he was a University Scholar, and the recipient of the Margaret L. Teague Spirit of ACU Award. He received the MSc and PhD degrees in 2008 and 2011, respectively, from the Department of Computer Science, University of Houston. His research interests include wireless mesh networks, wireless sensor networks, error-correcting coding theory, and robust distributed storage systems. He is a member of the IEEE.

Rong Zheng (S'03-M'04-SM'10) received the BE and ME degrees in electrical engineering from Tsinghua University, PR China, and the PhD degree from the Department of Computer Science, University of Illinois at Urbana-Champaign. She has been on the faculty of the Department of Computer Science, University of Houston since 2004, where she is currently an associate professor. Her research interests include network monitoring and diagnosis, cyber-physical systems, and sequential learning and decision theory. She received the US National Science Foundation (NSF) CAREER Award in 2006. She serves on the technical program committees of leading networking conferences including INFOCOM, ICDCS, ICNP. She served as a guest editor for the *EURASIP Journal on Advances in Signal Processing*'s special issue on wireless location estimation and tracking. She also served as a guest editor for the *Elsevier Computer Communications Journal*'s special issue on cyber-physical systems. She is a senior member of the IEEE and the IEEE Computer Society.