Rank Metric Decoder Architectures for Random Linear Network Coding With Error Control

Ning Chen, *Member, IEEE*, Zhiyuan Yan, *Senior Member, IEEE*, Maximilien Gadouleau, *Member, IEEE*, Ying Wang, *Member, IEEE*, and Bruce W. Suter, *Senior Member, IEEE*

Abstract—While random linear network coding is a powerful tool for disseminating information in communication networks, it is highly susceptible to errors caused by various sources. Due to error propagation, errors greatly deteriorate the throughput of network coding and seriously undermine both reliability and security of data. Hence, error control for network coding is vital. Recently, constant-dimension codes (CDCs), especially Kötter–Kschischang (KK) codes, have been proposed for error control in random linear network coding. KK codes can also be constructed from Gabidulin codes, an important class of rank metric codes. Rank metric decoders have been recently proposed for both Gabidulin and KK codes, but they have high computational complexities. Furthermore, it is not clear whether such decoders are feasible and suitable for hardware implementations. In this paper, we reduce the complexities of rank metric decoders and propose novel decoder architectures for both codes. The synthesis results of our decoder architectures for Gabidulin and KK codes with limited error-correcting capabilities over small fields show that our architectures not only are affordable, but also achieve high throughput.

Index Terms—Constant-dimension codes (CDCs), decoding, error correction coding, Gabidulin codes, Galois fields, integrated circuits, Kötter–Kschischang (KK) codes, network coding, rank metric codes, subspace codes.

I. INTRODUCTION

NETWORK coding [1] is a promising candidate for a new unifying design paradigm for communication networks, due to its advantages in throughput and robustness to network failures. Hence, network coding is already used or considered in gossip-based data dissemination, 802.11 wireless ad hoc networking, peer-to-peer networks, and mobile ad hoc networks (MANETs).

Random linear network coding (RLNC) [2] is arguably the most important class of network coding. RLNC treats all packets as vectors over some finite field and forms an outgoing packet by linearly combining incoming packets using random coefficients. Due to its random linear operations, RLNC not only achieves network capacity in a distributed manner, but also provides robustness to changing network conditions. Unfortunately, it is highly susceptible to errors caused by various reasons, such as noise, malicious or malfunctioning nodes, or insufficient min-cut [3]. Since linearly combining packets results in error propagation, errors greatly deteriorate the throughput of network coding and seriously undermine both reliability and security of data. Thus, error control for random linear network coding is critical.

Error control schemes proposed for RLNC assume two types of transmission models. The schemes of the first type (see, for example, [4]) depend on and take advantage of the underlying network topology or the particular linear network coding operations performed at various network nodes. The schemes of the second type [3], [5] assume that the transmitter and receiver have no knowledge of such channel transfer characteristics. The two transmission models are referred to as coherent and noncoherent network coding, respectively.

It has been recently shown [3] that an error control code for noncoherent network coding, called a subspace code, is a set of subspaces (of a vector space), and information is encoded in the choice of a subspace as a codeword; a set of packets that generate the chosen subspace is then transmitted [3]. A subspace code is called a constant-dimension code (CDC) if its subspaces are of the same dimension. CDCs are of particular interest since they lead to simplified network protocols due to the fixed dimension. A class of asymptotically optimal CDCs has been proposed in [3], and this class is referred to as the KK codes. A decoding algorithm based on interpolation for bivariate linearized polynomials is also proposed in [3] for the KK codes. It was shown that KK codes correspond to lifting [5] of Gabidulin codes [6], [7], which are a class of optimal rank metric codes. Gabidulin codes are also called maximum rank distance (MRD) codes, since they achieve the Singleton bound in the rank metric [6], as Reed–Solomon (RS) codes achieve the Singleton bound of Hamming distance. Due to the connection between Gabidulin and KK codes, the decoding of KK codes can be viewed as generalized decoding of Gabidulin codes, which involves deviations as well as errors and erasures [5]. Gabidulin codes are significant in themselves: For coherent network coding, the error correction capability of error control schemes is succinctly described by the rank metric [8]; thus, error control codes for coherent network coding are essentially rank metric codes.

The benefits of network coding above come at the price of additional operations needed at the source nodes for encoding, at

N. Chen is with SandForce Inc., Saratoga, CA 95070 USA (e-mail: nchen@sandforce.com).

Z. Yan is with the Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015 USA (e-mail: yan@lehigh.edu).

M. Gadouleau is with the Department of Computer Science, Queen Mary, University of London, E1 4NS U.K. (e-mail: mgadouleau@eecs.qmul.ac.uk).

Y. Wang is with the New Jersey Research Center, QUALCOMM, Inc., Bridgewater, NJ 08807 USA (e-mail: aywang11@gmail.com).

B. W. Suter is with Air Force Research Laboratory, Rome, NY 13441 USA (e-mail: bruce.suter@rl.af.mil).

the intermediate nodes for linear combining, and at the destination node(s) for decoding. In practice, the decoding complexities at destination nodes are much greater than the encoding and combining complexities. The decoding complexities of RLNC are particularly high when large underlying fields are assumed and when additional mechanisms such as error control are accounted for. Clearly, the decoding complexities of RLNC are critical to both software and hardware implementations. Furthermore, area/power overheads of their VLSI implementations are important factors in system design. Unfortunately, prior research efforts have mostly focused on theoretical aspects of network coding, and complexity reduction and efficient VLSI implementation of network coding decoders have not been sufficiently investigated so far. For example, although the decoding complexities of Gabidulin and KK codes were analyzed in [9], [10] and [3], [5], respectively, they do not reflect the impact of the size of the underlying finite fields. To ensure high probability of success for RLNC, a field of size 2^8 or 2^{16} is desired [11]. However, these large field sizes will increase decoding complexities and hence complicate hardware implementations. Finally, to the best of our knowledge, hardware architectures for these decoders have not been investigated in the open literature.

In this paper, we fill this significant gap by investigating complexity reduction and efficient hardware implementation for decoders in RLNC with error control. This effort is significant to the evaluation and design of network coding for several reasons. First, our results evaluate the complexities of decoders for RLNC as well as the area, power, and throughput of their hardware implementations, thereby helping to determine the feasibility and suitability of network coding for various applications. Second, our research results provide instrumental guidelines to the design of network coding from the perspective of complexity as well as hardware implementation. Third, our research results lead to efficient decoders and hence reduce the area and power overheads of network coding.

In this paper, we focus on the generalized Gabidulin decoding algorithm [5] for the KK codes and the modified Berlekamp-Massey decoding algorithm in [12] for Gabidulin codes for two reasons. First, compared with the decoding algorithm in [3], the generalized Gabidulin decoding [5] has a smaller complexity, especially for high-rate KK codes [5]. Second, components in the errors-only Gabidulin decoding algorithm in [12] can be easily adapted in the generalized Gabidulin decoding of KK codes. Thus, among the decoding algorithms for Gabidulin codes, we focus on the decoding algorithm in [12].

Although we focus on RLNC with error control in this paper, our results can be easily applied to RLNC without error control. For RLNC without error control, the decoding complexity is primarily due to inverting of the global coding matrix via Gauss–Jordan elimination, which is also considered in this paper.

Our main contributions include several algorithmic reformulations that reduce the computational complexities of decoders for both Gabidulin and KK codes. Our complexity-saving algorithmic reformulations are given here.

- We first adopt normal basis representations for all finite field elements and then significantly reduce the complexity of bit-parallel normal basis multipliers by using our common subexpression elimination (CSE) algorithm [13];

- The decoding algorithms of both Gabidulin and KK codes involve solving key equations. We adapt the inversionless Berlekamp–Massey algorithm (BMA) in [14], [15] to solving key equations for rank metric codes. Our inversionless BMA leads to reduced complexities as well as efficient architectures;

- The decoding algorithm of KK codes requires that the input be arranged in a row reduced echelon (RRE) form [16]. We define a more generalized form called n-RRE form, and show that it is sufficient if the input is in the n-RRE form. This change not only reduces the complexity of reformulating the input, but also enables parallel processing of decoding KK codes based on Cartesian products.

Another main contribution of this paper is efficient decoder architectures for both Gabidulin and KK codes. Aiming to reduce the area and to improve the regularity of our decoder architectures, we have also reformulated other steps in the decoding algorithm. To evaluate the performance of our decoder architectures for Gabidulin and KK codes, we implement our decoder architecture for two rate-1/2 Gabidulin codes and their corresponding KK codes. Our KK decoders can be used in network coding with various packet lengths by Cartesian product [5]. The synthesis results of our decoders show that our decoder architectures for Gabidulin and KK codes over small fields with limited error-correcting capabilities not only are affordable, but also achieve high throughput. Our decoder architectures and implementation results are novel to the best of our knowledge.

The decoders considered in this work are bounded distance decoders, and their decoding capability is characterized in [5, Theorem 11]. The thrust of our work is to reduce complexities and to devise efficient architectures for such decoders, while maintaining their decoder capability. To this end, our reformulations of the decoding algorithms do not affect the decoding capability of the bounded distance decoders of Gabidulin and KK codes. The error performance of the bounded distance decoders has been investigated in our previous works [17]–[19]. Hence, despite its significance, a detailed error performance analysis is beyond the scope of this paper, and we do not include it due to limited space.

The remainder of this paper is organized as follows. After briefly reviewing the background in Section II, we present our complexity-saving algorithmic reformulations and efficient decoder architectures in Sections III and IV, respectively. In Section V, the proposed architectures are implemented in Verilog and synthesized for area/performance evaluation. The conclusion is given in Section VI.

II. PRELIMINARIES

A. Notation

Let q denote a power of prime and $\mathbb{F}_{q^m}$ denote a finite field of order q^m. We use $\boldsymbol{I}_n$, $\mathbb{F}_q^n$, and $\mathbb{F}_q^{n \times m}$ to denote an $n \times n$ identity matrix, an n-dimensional vector space over $\mathbb{F}_q$, and the set of all $n \times m$ matrices over $\mathbb{F}_q$, respectively. For a set $\mathcal{U} \subseteq \{0, 1, \ldots, n-1\}$, $\mathcal{U}^c$ denotes the complement subset $\{0, 1, \ldots, n-1\} \setminus \mathcal{U}$ and $\boldsymbol{I}_{\mathcal{U}}$ denotes the columns of $\boldsymbol{I}_n$ in $\mathcal{U}$. In this paper, all vectors and matrices are in bold face.

The rank weight of a vector over $\mathbb{F}_{q^m}$ is defined as the maximal number of its coordinates that are linearly independent over
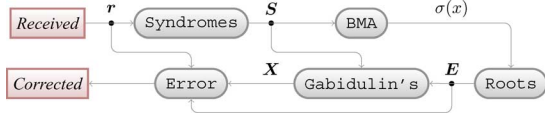
Fig. 1. Data flow of Gabidulin decoding.

the base field $\mathbb{F}_q$. Rank metric between two vectors over $\mathbb{F}_{q^m}$ is the rank weight of their difference [20]. For a column vector $\boldsymbol{X} \in \mathbb{F}_{q^m}^n$, we can expand each of its component into a row vector over the base field $\mathbb{F}_q$. Such a row expansion leads to an $n \times m$ matrix over $\mathbb{F}_q$. In this paper, we slightly abuse the notation so that $\boldsymbol{X}$ can represent a vector in $\mathbb{F}_{q^m}^n$ or a matrix in $\mathbb{F}_q^{n \times m}$, although the meaning is usually clear given the context.

Given a matrix $\boldsymbol{X}$, its row space, rank, and RRE form are denoted by $\langle \boldsymbol{X} \rangle$, rank $\boldsymbol{X}$, and $\mathrm{RRE}(\boldsymbol{X})$, respectively. For a subspace $\langle \boldsymbol{X} \rangle$, its dimension is denoted by $\dim \langle \boldsymbol{X} \rangle$ and rank $\boldsymbol{X} = \dim \langle \boldsymbol{X} \rangle$. The rank distance of two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{F}_{q^m}^n$ is defined as $d_R(\boldsymbol{X}, \boldsymbol{Y}) \triangleq \mathrm{rank}(\boldsymbol{X} - \boldsymbol{Y})$. The subspace distance [3] of their row spaces $\langle \boldsymbol{X} \rangle, \langle \boldsymbol{Y} \rangle$ is defined as $d_S(\langle \boldsymbol{X} \rangle, \langle \boldsymbol{Y} \rangle) \triangleq \dim \langle \boldsymbol{X} \rangle + \dim \langle \boldsymbol{Y} \rangle - 2 \dim(\langle \boldsymbol{X} \rangle \cap \langle \boldsymbol{Y} \rangle)$.

A linearized polynomial [21], [22] (or q-polynomial) over $\mathbb{F}_{q^m}$ is a polynomial of the form $f(x) = \sum_{i=0}^{p} f_i x^{q^i}$, where $f_i \in \mathbb{F}_{q^m}$. For a linearized polynomial $f(x)$, its q-degree is defined to be the greatest value of i for which f_i is nonzero. For convenience, let $[i]$ denote q^i. The symbolic product of two linearized polynomials $a(x)$ and $b(x)$, denoted by $\otimes$ (that is, $a(x) \otimes b(x) = a(b(x))$), is also a linearized polynomial. The q-reverse of a linearized polynomial $f(x) = \sum_{i=0}^{p} f_i x^{[i]}$ is given by the polynomial $\bar{f}(x) = \sum_{i=0}^{p} \bar{f}_i x^{[i]}$, where $\bar{f}_i = f_{p-i}^{[i-p]}$ for $i = 0, 1, \ldots, p$ and p is the q-degree of $f(x)$. For a set $\boldsymbol{\alpha}$ of field elements, we use $\mathrm{minpoly}(\boldsymbol{\alpha})$ to denote its minimal linearized polynomial, which is the monic linearized polynomial of least degree such that all the elements of $\boldsymbol{\alpha}$ are its roots.

B. Gabidulin Codes and Their Decoding

A Gabidulin code [6] is a linear (n, k) code over $\mathbb{F}_{q^m}$, whose parity-check matrix has a form as

$$\boldsymbol{H} = \begin{bmatrix} h_0^{[0]} & h_1^{[0]} & \cdots & h_{n-1}^{[0]} \\ h_0^{[1]} & h_1^{[1]} & \cdots & h_{n-1}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ h_0^{[n-k-1]} & h_1^{[n-k-1]} & \cdots & h_{n-1}^{[n-k-1]} \end{bmatrix} \quad (1)$$

where $h_0, h_1, \ldots, h_{n-1} \in \mathbb{F}_{q^m}$ are linearly independent over $\mathbb{F}_q$. Let $\boldsymbol{h}$ denote $(h_0, h_1, \ldots, h_{n-1})^T$. Since $\mathbb{F}_{q^m}$ is an m-dimensional vector space over $\mathbb{F}_q$, it is necessary that $n \leq m$. The minimum rank distance of a Gabidulin code is $d = n - k + 1$, and hence Gabidulin codes are MRD codes.

The decoding process of Gabidulin codes includes five major steps: syndrome computation, key equation solver, finding the root space, finding the error locators by Gabidulin's algorithm [6], and finding error locations. The data flow of Gabidulin decoding is shown in Fig. 1.

Key equation solvers based on a modified BMA [12] or a modified Welch–Berlekamp algorithm (WBA) [23] have been proposed. In this paper, we focus on the modified BMA due to its low complexity.

As in RS decoding, we can compute syndromes for Gabidulin codes as $\boldsymbol{S} = (S_0, S_1, \ldots, S_{d-2}) \triangleq \boldsymbol{H}\boldsymbol{r}$ for any received vector $\boldsymbol{r}$. Then the syndrome polynomial $S(x) = \sum_{j=0}^{d-2} S_j x^{[j]}$ can be used to solve the key equation [12, Theorem 3]

$$\sigma(x) \otimes S(x) \equiv \omega(x) \bmod x^{[d-1]} \quad (2)$$

for the error span polynomial $\sigma(x)$, using the BMA. Up to $t = \lfloor (d-1)/2 \rfloor$ error values E_j's can be obtained by finding a basis $E_0, E_1, \ldots$ for the root space of $\sigma(x)$ using the methods in [24], [25]. Then we can find the error locators X_j's corresponding to E_j's by solving a system of equations

$$S_l = \sum_{j=0}^{\tau-1} X_j^{[l]} E_j, \quad l = 0, 1, \ldots, d-2 \quad (3)$$

where τ is the number of errors. Gabidulin's algorithm [6] in Algorithm 1 can be used to solve (3). Finally, the error locations $\boldsymbol{L}_j$'s are obtained from X_j's by solving

$$X_j = \sum_{i=0}^{n-1} L_{j,i} h_i, \quad j = 0, 1, \ldots, \tau-1. \quad (4)$$

Algorithm 1 (Gabidulin's Algorithm [6]).

Input: $S_0, S_1, \ldots, S_{d-2}$ and $E_0, E_1, \ldots, E_{\tau-1}$

Output: $X_0, X_1, \ldots, X_{\tau-1}$

1.1 Compute $\tau \times \tau$ matrices $\boldsymbol{A}$ and $\boldsymbol{Q}$ as

$$A_{i,j} = \begin{cases} E_j & i = 0 \\ 0, & i \neq 0, j < i \\ A_{i-1,j} - A_{i-1,i-1} \left(\frac{A_{i-1,j}}{A_{i-1,i-1}} \right)^{[-1]} & i \neq 0, j \geq i \end{cases}$$

$$Q_{i,j} = \begin{cases} S_j & i = 0 \\ Q_{i-1,j} - A_{i-1,i-1} \left(\frac{Q_{i-1,j+1}}{A_{i-1,i-1}} \right)^{[-1]} & \text{otherwise.} \end{cases}$$

1.2 Compute X_i's recursively as $X_{\tau-1} = Q_{\tau-1,0}/A_{\tau-1,\tau-1}$ and $X_i = (Q_{i,0} - \sum_{j=i+1}^{\tau-1} A_{i,j} X_j)/A_{i,i}$, for $i = \tau-2, \tau-3, \ldots, 0$.

In total, the decoding complexity of Gabidulin codes is roughly $O(n^2(1 - R))$ operations over $\mathbb{F}_{q^m}$ [9], where $R = k/n$ is the code rate, or $O(dm^3)$ operations over $\mathbb{F}_q$ [10]. Note that all polynomials involved in the decoding process are linearized polynomials.

Gabidulin codes are often viewed as the counterpart in rank metric codes of the well-known RS codes. As shown in Table I, an analogy between RS and Gabidulin codes can be established in many aspects. Such an analogy helps us understand the decoding of Gabidulin codes and, in some cases, allows us to adapt innovations proposed for RS codes to Gabidulin codes.

C. KK Codes and Their Decoding

By the *lifting* operation [5], KK codes can be constructed from Gabidulin codes. Lifting can also be seen as a generalization of the standard approach to random linear network coding
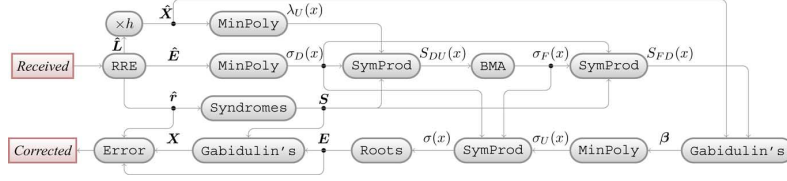
Fig. 2. Data flow of KK decoding.

TABLE I
ANALOGY BETWEEN RS AND GABIDULIN CODES

	Reed–Solomon	Gabidulin
Metric	Hamming	Rank
Ring of	Polynomials	Linearized Polynomials
Degree	i	$[i] = q^i$
Key Operation	Polynomial Multiplication	Symbolic Product
Generation Matrix	$[g_j^i]$	$[g_j^{[i]}]$
Parity Check Matrix	$[h_j^i]$	$[h_j^{[i]}]$
Key Equation Solver	BMA	Modified BMA
Error Locations	Roots	Root Space Basis
Error Value Solver	Forney's Formula	Gabidulin's Algorithm

[2], which transmits matrices in the form $\boldsymbol{X} = [\boldsymbol{I}|\boldsymbol{x}]$, where $\boldsymbol{X} \in \mathbb{F}_q^{n \times M}$, $\boldsymbol{x} \in \mathbb{F}_q^{n \times m}$, and $m = M - n$.

In practice, the packet length could be very long. To accommodate long packets based on the KK codes, very large m and n are needed, which results in prohibitively high complexity due to the huge field size of $\mathbb{F}_{q^m}$. A low-complexity approach in [5] suggested that instead of using a single long Gabidulin code, a Cartesian product of many short Gabidulin codes with the same distance can be used to construct CDCs for long packets via the lifting operation.

Let the received matrix be $\boldsymbol{Y} = [\hat{\boldsymbol{A}}|\boldsymbol{y}]$, where $\hat{\boldsymbol{A}} \in \mathbb{F}_q^{N \times n}$ and $\boldsymbol{y} \in \mathbb{F}_q^{N \times m}$. Note that we always assume the received matrix is full-rank [5]. The row and column rank deficiencies of $\hat{\boldsymbol{A}}$ are $\delta = N - \mathrm{rank}\,\hat{\boldsymbol{A}}$ and $\mu = n - \mathrm{rank}\,\hat{\boldsymbol{A}}$, respectively. In the decoding algorithm of [5], the matrix $\boldsymbol{Y}$ is first turned into an RRE form, and then the RRE form of $\boldsymbol{Y}$ is expanded into $\bar{\boldsymbol{Y}} = \begin{bmatrix} \boldsymbol{I}_{\mathcal{U}^c} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_\delta \end{bmatrix} \mathrm{RRE}(\boldsymbol{Y}) = \begin{bmatrix} \boldsymbol{I}_n + \hat{\boldsymbol{L}}\boldsymbol{I}_{\mathcal{U}}^T & \boldsymbol{r} \\ \boldsymbol{0} & \hat{\boldsymbol{E}} \end{bmatrix}$, where $\mathcal{U}^c$ denotes the column positions of leading entries in the first n rows of $\mathrm{RRE}(\boldsymbol{Y})$. The tuple $(\boldsymbol{r}, \hat{\boldsymbol{L}}, \hat{\boldsymbol{E}})$ is called a *reduction* of $\boldsymbol{Y}$ [5]. It was proved [5] that $d_S(\langle \boldsymbol{X} \rangle, \langle \boldsymbol{Y} \rangle) = 2\,\mathrm{rank} \begin{bmatrix} \hat{\boldsymbol{L}} & \boldsymbol{r} - \boldsymbol{x} \\ \boldsymbol{0} & \hat{\boldsymbol{E}} \end{bmatrix} - \mu - \delta$, where $\mu = n - \mathrm{rank}\,\hat{\boldsymbol{L}}$ and $\delta = N - \mathrm{rank}\,\hat{\boldsymbol{L}}$. Now the decoding problem to minimize the subspace distance becomes a problem to minimize the rank distance.

For a KK code $\mathcal{C}$, the generalized rank decoding [5] finds an error word $\hat{\boldsymbol{e}} = \arg\min_{\boldsymbol{e} \in \boldsymbol{r} - \mathcal{C}} \mathrm{rank} \begin{bmatrix} \hat{\boldsymbol{L}} & \boldsymbol{e} \\ \boldsymbol{0} & \hat{\boldsymbol{E}} \end{bmatrix}$. The error word $\hat{\boldsymbol{e}}$ is expanded as a summation of products of column and row vectors [5] such that $\hat{\boldsymbol{e}} = \sum_{j=0}^{\tau-1} \boldsymbol{L}_j \boldsymbol{E}_j$. Each term $\boldsymbol{L}_j \boldsymbol{E}_j$ is called either an *erasure*, if $\boldsymbol{L}_j$ is known, or a *deviation*, if $\boldsymbol{E}_j$ is known, or an *error*, if neither $\boldsymbol{L}_j$ nor $\boldsymbol{E}_j$ is known. In this general decoding problem, $\bar{\boldsymbol{L}}$ has μ columns from $\hat{\boldsymbol{L}}$ and $\bar{\boldsymbol{E}}$ has δ rows from $\hat{\boldsymbol{E}}$. Given a Gabidulin code of minimum distance d, the corresponding KK code is able to correct ϵ errors, μ erasures, and δ deviations as long as if $2\epsilon + \mu + \delta < d$.

Algorithm 2 was proposed [5] for generalized decoding of the KK codes, and its data flow is shown in Fig. 2. It requires $O(dm)$ operations in $\mathbb{F}_{q^m}$ [5].

Algorithm 2 (General Rank Decoding [5]).

Input: received tuple $(\boldsymbol{r}, \hat{\boldsymbol{L}}, \hat{\boldsymbol{E}})$

Output: error word $\hat{\boldsymbol{e}}$

2.1 Compute $\boldsymbol{S} = \boldsymbol{H}\boldsymbol{r}$, $\hat{\boldsymbol{X}} = \hat{\boldsymbol{L}}^T \boldsymbol{h}$, $\lambda_U(x) = \mathrm{minpoly}(\hat{\boldsymbol{X}})$, $\sigma_D(x) = \mathrm{minpoly}(\hat{\boldsymbol{E}})$, and $S_{DU}(x) = \sigma_D(x) \otimes S(x) \otimes \zeta_U(x)$, where $\zeta_U(x)$ is the q-reverse of $\lambda_U(x)$.

2.2 Compute the error span polynomial:

  a) Use the modified BMA [12] to solve the key equation $\sigma_F(x) \otimes S_{DU}(x) \equiv \omega(x) \bmod x^{[d-1]}$ such that $\deg \omega(x) < [\tau]$ where $\tau = \epsilon + \mu + \delta$.

  b) Compute $S_{FD}(x) = \sigma_F(x) \otimes \sigma_D(x) \otimes S(x)$.

  c) Use Gabidulin's algorithm [6] to find $\boldsymbol{\beta}$ that solves $S_{FD,l} = \sum_{j=0}^{\mu-1} X_j^{[l]} \beta_j, l = d-2, d-3, \ldots, d-1-\mu$.

  d) Compute $\sigma_U(x) = \mathrm{minpoly}(\boldsymbol{\beta})$ followed by $\sigma(x) = \sigma_U(x) \otimes \sigma_F(x) \otimes \sigma_D(x)$.

2.3 Find a basis $\boldsymbol{E}$ for the root space of $\sigma(x)$.

2.4 Find the error locations:

  a) Solve $S_l = \sum_{j=0}^{\tau-1} X_j^{[l]} E_j, l = 0, 1, \ldots, d-2$ using Gabidulin's algorithm [6] to find the error locators $X_0, X_1, \ldots, X_{\tau-1} \in \mathbb{F}_{q^m}$.

  b) Compute the error locations $\boldsymbol{L}_j$'s by solving (4).

  c) Compute the error word $\hat{\boldsymbol{e}} = \sum_{j=1}^{\tau} \boldsymbol{L}_j \boldsymbol{E}_j$, where each $\bar{\boldsymbol{E}}_j$ is the row expansion of E_j.

III. COMPUTATIONAL COMPLEXITY REDUCTION

In general, RLNC is carried out over $\mathbb{F}_q$, where q is any prime power. That is, packets are treated as vectors over $\mathbb{F}_q$. Since our investigation of computational complexities is for both software and hardware implementations of RLNC, where data are stored and transmitted in bits, we focus on RLNC over characteristic-2 fields in our work, i.e., q is a power of two. In some cases, we further assume $q = 2$, as it leads to further complexity reductions.

A. Finite Field Representation

Finite field elements can be represented by vectors using different types of bases: polynomial basis, normal basis, and dual basis [26]. In rank metric decoders, most polynomials involved are linearized polynomials, and hence their evaluations and symbolic products require computing their $[i]$th powers. Suppose a field element is represented by a vector over $\mathbb{F}_q$ with respect to a normal basis, computing $[i]$th powers (i is a positive or negative integer) of the element is simply cyclic shifts of the corresponding vector by i positions, which significantly reduces computational complexities. For example,

TABLE II
COMPLEXITIES OF BIT-PARALLEL NORMAL BASIS MULTIPLIERS OVER FINITE FIELDS (FOR THESE TWO FIELDS, ALL THREE IMPLEMENTATIONS HAVE THE SAME CPD.)

| | $P(x)$ | AND | XOR | | |
			direct [26]	[29]	Ours
$\mathbb{F}_{2^m}$	-	m^2	$m(C_N - 1)$	$m(C_N + m - 2)/2$	-
$\mathbb{F}_{2^8}$	$(\sum_{i=0}^{8} x^i) - x^6 - x^4 - x^2$	64	160	108	88
$\mathbb{F}_{2^{16}}$	$(\sum_{i=0}^{16} x^i) - x^{14} - x^9 - x^6 - x^4$	256	1344	792	491

the computational complexity of Algorithm 1 is primarily due to the following updates in Step 1.1:

$$
\begin{cases}
A_{i,j} = A_{i-1,j} - \left(\frac{A_{i-1,j}}{A_{i-1,i-1}}\right)^{[-1]} A_{i-1,i-1} \\
Q_{i,j} = Q_{i-1,j} - \left(\frac{Q_{i-1,j+1}}{A_{i-1,i-1}}\right)^{[-1]} A_{i-1,i-1}
\end{cases}
\tag{5}
$$

which require divisions and computing $[-1]$th powers. With normal basis representation, $[-1]$th powers are obtained by a single cyclic shift. When $q = 2$, they can be computed in an inversionless form $A_{i,j} = A_{i-1,j} - (A_{i-1,j}A_{i-1,i-1})^{[-1]}$, $Q_{i,j} = Q_{i-1,j} - (Q_{i-1,j+1}A_{i-1,i-1})^{[-1]}$, which also avoids finite field divisions or inversions. Thus, using normal basis representation also reduces the complexity of Gabidulin's algorithm.

In addition to lower complexities of finite field arithmetic operations, normal basis representation leads to reduced complexities in the decoding of Gabidulin and KK codes for several reasons. First, it was shown that using normal basis can facilitate the computation of symbolic product [9]. Second, it was also suggested [9] that solving (4) can be trivial using normal basis. If $(h_0, h_1, \ldots, h_{m-1})$ is a normal basis, the matrix $\boldsymbol{h}$, whose rows are vector representations of h_i's with respect to the basis h_i's, becomes an identity matrix with additional all-zero columns. Hence, solving (4) requires no computation. These two complexity reductions were also observed in [10]. Third, if a normal basis of $\mathbb{F}_{2^m}$ is used as h_i's and $n = m$, the parity check matrix $\boldsymbol{H}$ in (1) becomes a cyclic matrix. Thus, syndrome computation becomes part of a cyclic convolution of $(h_0, h_1, \ldots, h_{m-1})$ and $\boldsymbol{r}$, for which fast algorithms are available (see, for example, [27]). Using fast cyclic convolution algorithms are favorable when m is large.

B. Normal Basis Arithmetic Operations

We also propose finite field arithmetic operations with reduced complexities, when normal basis representation is used. When represented by vectors, the addition and subtraction of two elements are simply component-wise addition, which is straightforward to implement. For characteristic-2 fields $\mathbb{F}_{2^m}$, inverses can be obtained efficiently by a sequence of squaring and multiplying, since $\beta^{-1} = \beta^{2^m - 2} = \beta^2 \beta^4 \ldots \beta^{2^{m-1}}$ for $\beta \in \mathbb{F}_{2^m}$ [26]. Since the $[i]$-th powers require no computation, the complexity of inversion in turn depends on that of multiplication. Division can be implemented by a concatenation of inversion and multiplication: $\alpha/\beta = \alpha \cdot \beta^{-1}$, and hence the complexity of division also depends on that of multiplication in the end.

There are serial and parallel architectures for normal basis finite field multipliers. To achieve high throughput in our decoder, we consider only parallel architectures. Most normal basis multipliers are based on the Massey–Omura (MO) architecture [26], [28]. The complexity of a serial MO normal basis multiplier over

$\mathbb{F}_{2^m}$, C_N, is defined as the number of terms $a_i b_j$ in computing a bit of the product $c = ab$, where $a = \sum_{i=0}^{m-1} a_i h_i \in \mathbb{F}_{2^m}$ and $b = \sum_{j=0}^{m-1} b_j h_j \in \mathbb{F}_{2^m}$ and $(h_0, h_1, \ldots, h_{m-1})$ is a normal basis. It has been shown [29] that a parallel MO multiplier over $\mathbb{F}_{2^m}$ needs m^2 AND gates and at most $m(C_N + m - 2)/2$ XOR gates. For instance, for the fields $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{16}}$, their C_N's are minimized to 21 and 85, respectively [26]. Using a common subexpression elimination algorithm [13], we significantly reduce the number of XOR gates while maintaining the same critical path delays (CPDs) of one AND plus five XOR gates and one AND plus seven XOR gates as direct implementations, respectively. Our results are compared with those in [26] and [29] in Table II, where we also provide the prime polynomial $P(x)$ for each field.

The reduced gate count for normal basis multiplication is particularly important for hardware implementations of RLNC. This improvement is transparent to the complexity of decoders, in terms of finite field operations. When decoders for RLNC are realized in hardware, the reduced gate count for normal basis multiplication will be reflected in reduced area and power consumption.

C. Inversionless BMA

The modified BMA for rank metric codes [12] is similar to the BMA for RS codes except that polynomial multiplications are replaced by symbolic products. The modified BMA [12] requires finite field divisions, which are more complex than other arithmetic operations. Following the idea of inversionless RS decoder [14], we propose an inversionless variant in Algorithm 3.

Algorithm 3. iBMA

Input: Syndromes $\boldsymbol{S}$

Output: $\Lambda(x)$

    3.1 Initialize: $\Lambda^{(0)}(x) = B^{(0)}(x) = x^{[0]}$, $\Gamma^{(0)} = 1$, and $L = 0$.
    3.2 For $r = 0, 1, \ldots, 2t - 1$,
    a) Compute the discrepancy $\Delta_r = \sum_{j=0}^{L} \Lambda_j^{(r)} S_{r-j}^{[j]}$.
    b) If $\Delta_r = 0$, then go to (e).
    c) Modify the connection polynomial:
       $\Lambda^{(r+1)}(x) = (\Gamma^{(r)})^{[1]} \Lambda^{(r)}(x) - \Delta_r x^{[1]} \otimes B^{(r)}(x)$.
    d) If $2L > r$, go to (e). Otherwise, $L = r + 1 - L$, $\Gamma^{(r+1)} = \Delta_r$, and $B^{(r)}(x) = \Lambda^{(r)}(x)$. Go to (a).
    e) Set $\Gamma^{(r+1)} = (\Gamma^{(r)})^{[1]}$ and $B^{(r+1)}(x) = x^{[1]} \otimes B^{(r)}(x)$.
    3.3 Set $\Lambda(x) = \Lambda^{(2t)}(x)$.

Using a similar approach as in [14], we prove that the output $\Lambda(x)$ of Algorithm 3 is the same as $\sigma(x)$ produced by the modi-

fied BMA, except it is scaled by a constant $C = \prod_{i=0}^{t-1} (\Gamma^{(2i)})^{[1]}$. However, this scaling is inconsequential since the two polynomials have the same root space.

Using normal basis, the modified BMA in [12] requires at most $\lfloor (d-2)/2 \rfloor$ inversions, $(d-1)(d-2)$ multiplications, and $(d-1)(d-2)$ additions over $\mathbb{F}_{q^m}$ [9]. Our inversionless version, Algorithm 3, requires at most $(3/2)d(d-1)$ multiplications and $(d-1)(d-2)$ additions. Since a normal basis inversion is obtained by $m-1$ normal basis multiplications, the complexity of normal basis inversion is roughly $m-1$ times that of normal basis multiplication. Hence, Algorithm 3 reduces the complexity considerably. Algorithm 3 is also more suitable for hardware implementation, as shown in Section IV.

D. Finding the Root Space

Instead of finding *roots* of polynomials in RS decoding, we need to find the *root spaces* of linearized polynomials in rank metric decoding. Hence, the Chien search [30] in RS decoding will have a high complexity for two reasons. First, it requires polynomial evaluations over the whole field, whose complexity is very high. Second, it cannot find a set of linearly independent roots.

A probabilistic algorithm to find the root space was proposed in [25]. For Gabidulin codes, it can be further simplified as suggested in [9]. However, hardware implementations of probabilistic algorithms require random number generators. Furthermore, the algorithm in [25] requires symbolic long division, which is also not suitable for hardware implementations. According to [5], the average complexity of the probabilistic algorithm in [25] is $O(dm)$ operations over $\mathbb{F}_{q^m}$, while that of Berlekamp's deterministic method [24] is $O(dm)$ operations in $\mathbb{F}_{q^m}$ plus $O(m^3)$ operations in $\mathbb{F}_q$. Since their complexity difference is small, we focus on the deterministic method, which is much easier to implement.

Suppose we need to find the root space of a linearized polynomial $r(x)$, Berlekamp's deterministic method first evaluates the polynomial $r(x)$ on a basis of the field $(\alpha_0, \alpha_1, \ldots, \alpha_{m-1})$ such that $v_i = r(\alpha_i), i = 0, 1, \ldots, m-1$. Then it expands v_i's in the base field as columns of an $m \times m$ matrix V and finds linearly independent roots z such that $Vz = 0$. Using the representation based on $(\alpha_0, \alpha_1, \ldots, \alpha_{m-1})$, the roots z are also the roots of the given polynomial. Finding z is to obtain the linear dependent combinations of the columns of V, which can be done by Gaussian elimination.

E. n-RRE Form

Given a received subspace spanned by a set of received packets, the input of Algorithm 2 is a three-tuple, called a reduction of the received space represented by its generator matrix Y; the three-tuple is obtained based on Y when it is in its RRE form [5]. Thus, before the decoding starts, preprocessing is performed on the received packets so as to obtain the RRE form of Y. We show that the processed matrix needs to satisfy only a relaxed constraint, which does not affect the decoding outcome, while leading to two advantages. First, the relaxed constraint results in reduced complexities in the preprocessing step. Second and more importantly, the relaxed constraint enables parallel processing of decoding KK codes based on Cartesian products.

We first define an n-RRE form for received matrices. Given a matrix $Y = [\hat{A}|y]$, where $\hat{A} \in \mathbb{F}_q^{N \times n}$ and $y \in \mathbb{F}_q^{N \times m}$, the matrix Y is in its n-RRE form as long as $\hat{A}$ (its leftmost n columns) is in its RRE form. Compared with the RRE form, the n-RRE form is more relaxed as it puts no constraints on the right part. We note that an n-RRE form of a matrix is not unique.

We now show that the relaxed constraint does not affect the decoding. Similar to [5, Proposition 7], we first show that a reduction based on n-RRE form of Y always exists. Given $\bar{Y} = [\hat{A}|y]$ and $\text{RRE}(\hat{A}) = R\hat{A}$, where R represents the reducing row operations, the product $\bar{Y}' = RY = [B'|Z']$ is in its n-RRE form. We note that $B' \in \mathbb{F}_q^{N \times n}$ and $Z \in \mathbb{F}_q^{N \times m}$, where the column and row rank deficiency of B' are given by $\mu' = n - \text{rank } B'$ and $\delta' = N - \text{rank } B'$, respectively. We have the following result about the reduction based on $\bar{Y}'$.

Lemma 1: Let $\bar{Y}'$ and μ' and δ' be defined as above. There exists a tuple $(r', \hat{L}', \hat{E}') \in \mathbb{F}_q^{n \times m} \times \mathbb{F}_q^{n \times \mu'} \times \mathbb{F}_q^{\delta' \times m}$ and a set $\mathcal{U}'$ satisfying $|\mathcal{U}'| = \mu'$, $I_{\mathcal{U}'}^T r' = 0$, $I_{\mathcal{U}'}^T \hat{L}' = -I_{\mu' \times \mu'}$, and rank $\hat{E}' = \delta'$ so that $\left\langle \begin{bmatrix} I_n + \hat{L}' I_{\mathcal{U}'}^T & r' \\ 0 & \hat{E}' \end{bmatrix} \right\rangle = \langle \bar{Y}' \rangle = \langle Y \rangle$.

See Appendix A for the proof of Lemma 1. Lemma 1 shows that we can find an alternative reduction based on n-RRE form of Y, instead of an RRE form of Y. The key of our alternative reduction of Y is that the reduction is mostly determined by the first n columns of $\text{RRE}(Y)$. Also, this alternative reduction does not come as a surprise. As shown in [5, Proposition 8], row operations on $\hat{E}$ can produce alternative reductions. Next, we show that decoding based on our alternative reduction is the same as in [5]. Similar to [5, Theorem 9], we have the following results.

Lemma 2: Let $(r', \hat{L}', \hat{E}')$ be a reduction of $\bar{Y}$ determined by its n-RRE form, we have $d_S(\langle X \rangle, \langle Y \rangle) = 2 \text{ rank} \begin{bmatrix} \hat{L}' & r' - x \\ 0 & \hat{E}' \end{bmatrix} - \mu' - \delta'$.

See Appendix B for the proof. Lemma 2 shows that the subspace decoding problem is equivalent to the generalized Gabidulin decoding problem with the alternative reduction $(r', \hat{L}', \hat{E}')$, which is obtained from an n-RRE form of Y.

Our alternative reduction leads to two advantages. First, it results in reduced complexity in preprocessing. Given a matrix Y, the preprocessing needed to transform Y into its n-RRE form is only part of the preprocessing to transform Y into its RRE form. We can show that the maximal number of arithmetic operations in the former preprocessing is given by $(N-1)\sum_{i=0}^{\text{rank } \hat{A}-1}(n+m-i)$, whereas that of the latter preprocessing is $(N-1)\sum_{i=0}^{\text{rank}(Y)-1}(n+m-i)$. Since rank $Y \geq$ rank $\hat{A}$, the relaxed constraint leads to a lower complexity, and the reduction depends on rank Y and rank $\hat{A}$. Second, the reduction for n-RRE forms is completely determined by the n leftmost columns of Y instead of the whole matrix, which greatly simplifies hardware implementations. This advantage is particularly important for the decoding of CDCs that are lifted from Cartesian products of Gabidulin codes. Since the row operations to obtain an n-RRE form depend on $\hat{A}$ only, decoding $[\hat{A}|y_0|y_1|\cdots|y_{l-1}]$ can be divided into parallel and smaller decoding problems whose inputs are $[\hat{A}|y_0], [\hat{A}|y_1], \ldots, [\hat{A}|y_{l-1}]$. Thus, for these CDCs, we can decode in a serial manner with only one small decoder, or in a partly parallel fashion with more

decoders, or even in a fully parallel fashion. This flexibility allows tradeoffs between cost/area/power and throughput. Furthermore, since the erasures $\hat{L}$ are determined by $\hat{A}$ and are the same for all $[A|y_i]$, the computation of $\hat{X}$ and $\lambda_U(x)$ in Algorithm 2 can be shared among these parallel decoding problems, thereby reducing overall complexity.

F. Finding Minimal Linearized Polynomials

Minimal linearized polynomials can be computed by solving systems of linear equations. Given roots $\beta_0, \beta_1, \ldots, \beta_{p-1}$, the minimal linearized polynomial $x^{[p]} + \sum_{i=0}^{p-1} a_i x^{[i]}$ satisfies

$$\begin{bmatrix} \beta_0^{[0]} & \beta_0^{[1]} & \cdots & \beta_0^{[p-1]} \\ \beta_1^{[0]} & \beta_1^{[1]} & \cdots & \beta_1^{[p-1]} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{p-1}^{[0]} & \beta_{p-1}^{[1]} & \cdots & \beta_{p-1}^{[p-1]} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{p-1} \end{bmatrix} = \begin{bmatrix} \beta_0^{[p]} \\ \beta_1^{[p]} \\ \vdots \\ \beta_{p-1}^{[p]} \end{bmatrix}. \quad (6)$$

Thus, it can be solved by Gaussian elimination over the extension field $\mathbb{F}_{q^m}$. Gabidulin's algorithm is not applicable because the rows of the matrix are not the powers of the same element.

The complexity to solve (6) is very high. Instead, we reformulate the method from [21, Ch. 1, Theorem 7]. The main idea of [21, Ch. 1, Theorem 7] is to recursively construct the minimal linearized polynomial using symbolic products instead of polynomial multiplications in polynomial interpolation. Given linearly independent roots $w_0, w_1, \ldots, w_{p-1}$, we can construct a series of linearized polynomials as: $F^{(0)}(x) = x^{[0]}$ and $F^{(i+1)}(x) = (x^{[1]} - (F^{(i)}(w_i))^{q-1} x^{[0]}) \otimes F^{(i)}(x)$ for $i = 0, 1, \cdots, p - 1$.

Although the recursive method in [21, Ch. 1, Theorem 7] is for p-polynomials, we can adapt it to linearized polynomials readily. A serious drawback of [21, Ch. 1, Theorem 7] is that the evaluation of $F^{(i)}(w_i)$ has a rapidly increasing complexity when the degree of $F^{(i)}(x)$ gets higher. To eliminate this drawback, we reformulate the algorithm so that the evaluation $F^{(i)}(w_i)$ is done in a recursive way. Our reformulated algorithm is based on the fact that $F_i(w_{i+1}) = (x^{[1]} - (F_{i-1}(w_i))^{q-1} x^{[0]}) \otimes F_{i-1}(w_{i+1})$. Representing $F^{(i)}(w_j)$ as $\gamma_j^{(i)}$, we obtain Algorithm 4.

Algorithm 4 (Minimal Linearized Polynomials).

Input: Roots $w_0, w_1, \ldots, w_{p-1}$

Output: The minimal linearized polynomial $F^{(p)}(x)$

    4.1 Set $\gamma_j^{(0)} = w_j$, for $j = 0, 1, \ldots, p - 1$ and $F^{(0)}(x) = x^{[0]}$.

    4.2 For $i = 0, 1, \ldots, p - 1$,

      a) If $\gamma_i^{(i)} = 0$, $F^{(i+1)}(x) = F^{(i)}(x)$ and $\gamma_j^{(i+1)} = \gamma_j^{(i)}$ for $j = i + 1, i + 2, \ldots, p - 1$; Otherwise,
$$F^{(i+1)}(x) = (F^{(i)}(x))^{[1]} - (\gamma_i^{(i)})^{q-1} F^{(i)}(x)$$
and $\gamma_j^{(i+1)} = (\gamma_j^{(i)})^{[1]} - (\gamma_i^{(i)})^{q-1} \gamma_j^{(i)}$ for $j = i + 1, i + 2, \ldots, p - 1$.

Since powers of q require only cyclic shifting, the operations in Algorithm 4 are simple. Also, Algorithm 4 does not require the roots to be linearly independent. In Algorithm 4, $F^{(i)}(w_j) = 0$ for $j = 0, 1, \ldots, i - 1$ and $\gamma_i^{(i+1)} = F^{(i)}(w_i)$.

If $w_0, w_1, \ldots, w_j$ are linearly dependent, $\gamma_j^{(j)} = 0$ and hence w_j is ignored. So Algorithm 4 integrates detection of linearly dependency at no extra computational cost.

Essentially, Algorithm 4 breaks down evaluations of high q-degree polynomials into evaluations of polynomials with q-degree of one. It avoids operations with very high complexity while maintaining the same total complexity of the algorithm.

IV. ARCHITECTURE DESIGN

Aiming to reduce the storage requirement and total area as well as to improve the regularity of our decoder architectures, we further reformulate the steps in the decoding algorithms of both Gabidulin and KK codes. Again, we assume the decoder architectures are suitable for RLNC over $\mathbb{F}_q$, where q is a power of two.

A. High-Speed BMA Architecture

To increase the throughput, regular BMA architectures with shorter CPD are necessary. Following the approaches in [15], we develop two architectures based on Algorithm 3, which are analogous to the riBM and RiBM algorithms in [15].

In Algorithm 3, the critical path is in step 3.2(a). Note that Δ_r is the rth coefficient of the discrepancy polynomial $\Delta^{(r)}(x) = \Lambda^{(r)}(x) \otimes S(x)$. By using $\Theta^{(r)}(x) = B^{(r)}(x) \otimes S(x)$, $\Delta^{(r+1)}(x)$ can be computed as

$$\begin{aligned} \Delta^{(r+1)}(x) &= \Lambda^{(r+1)}(x) \otimes S(x) \\ &= \left[\left(\Gamma^{(r)}\right)^{[1]} \Lambda^{(r)}(x) - \Delta_r x^{[1]} \otimes B^{(r)}(x) \right] \otimes S(x) \\ &= \left(\Gamma^{(r)}\right)^{[1]} \Delta^{(r)}(x) - \Delta_r x^{[1]} \otimes \Theta^{(r)}(x) \quad (7) \end{aligned}$$

which has the same structure as step 3.2(c). Hence, this reformulation is more conducive to a regular implementation. Given the similarities between step 3.2(a) and (7), $\Lambda(x)$ and $\Delta(x)$ can be combined together into one polynomial $\tilde{\Delta}(x)$. Similarly, $B(x)$ and $\Theta(x)$ can be combined into one polynomial $\tilde{\Theta}(x)$. These changes are incorporated in our RiBMA algorithm, shown in Algorithm 5.

Algorithm 5. RiBMA

Input: Syndromes S

Output: $\Lambda(x)$

    5.1 Initialize: $\tilde{\Delta}^{(0)}(x) = \tilde{\Theta}^{(0)}(x) = \sum_{i=0}^{2t-1} S_i x^{[i]}$, $\Gamma^{(0)} = 1$, $\tilde{\Delta}_{3t}^{(0)} = \tilde{\Theta}_{3t}^{(0)} = 1$, and $b = 0$.

    5.2 For $r = 0, 1, \ldots, 2t - 1$,

      a) Modify the combined polynomial: $\tilde{\Delta}^{(r+1)}(x) = \Gamma^{(r)} \tilde{\Delta}^{(r)}(x) - \tilde{\Delta}_0^{(r)} \tilde{\Theta}^{(r)}(x)$;

      b) Set $b = b + 1$;

      c) If $\tilde{\Delta}_0^{(r)} \neq 0$ and $b > 0$, set $b = -b$, $\Gamma^{(r+1)} = \tilde{\Delta}_0^{(r)}$, and $\tilde{\Theta}^{(r)}(x) = \tilde{\Delta}^{(r)}(x)$;

      d) Set $\tilde{\Delta}^{(r+1)}(x) = \sum_{i=0}^{3t-1} \tilde{\Delta}_{i+1}^{(r+1)} x^{[i]}$, $\tilde{\Theta}^{(r)}(x) = \sum_{i=0}^{3t-1} \tilde{\Theta}_i^{(r)} x^{[i]}$;

      e) Set $\Gamma^{(r+1)} = (\Gamma^{(r)})^{[1]}$ and $\tilde{\Theta}^{(r+1)}(x) = x^{[1]} \otimes \tilde{\Theta}^{(r)}(x)$.

    5.3 Set $\Lambda(x) = \sum_{i=0}^{t} \tilde{\Delta}_{i+t}^{(2t)} x^{[i]}$.

Following Algorithm 5, we propose a systolic RiBMA architecture shown in Fig. 3, which consists of $3t + 1$ identical processing elements (BEs), whose circuitry is shown in Fig. 4. The central control unit BCtrl, the rightmost cell in Fig. 3, updates b, generates the global control signals $\mathrm{ct}^{(r)}$ and $\Gamma^{(r)}$, and passes along the coefficient $\Lambda_0^{(r)}$. The control signal $\mathrm{ct}^{(r)}$ is set to 1 only if $\tilde{\Delta}_0^{(r)} \neq 0$ and $k > 0$. In each processing element, there are two critical paths, both of which consist of one multiplier and one adder over $\mathbb{F}_{2^m}$.

B. Generalized BMA

The key equation of KK decoding is essentially the same as (2), but $\omega(x)$ has a q-degree less than τ instead of $\lfloor (d-1)/2 \rfloor$. Actually, in KK decoding, we do not know the exact value of τ before solving the key equation. All we need is to determine the maximum number of correctable errors t' given μ erasures and δ deviations, which is given by $t' = \lfloor (d - 1 - \mu - \delta)/2 \rfloor$. Hence, we adapt our BMA in Section III-C to KK decoding, as in Algorithm 6. To apply Algorithm 6 to Gabidulin decoding, we can simply use $\theta = \mu + \delta = 0$.

Algorithm 6 (Generalized RiBMA).

Input: $\boldsymbol{S}$ and θ

Output: $\Lambda(x)$

    6.1 Initialize as follows: $t' = \lfloor (d - 1 - \theta)/2 \rfloor$, $\tilde{\Delta}^{(0)}(x) = \tilde{\Theta}^{(0)}(x) = \sum_{i=\theta}^{\theta+2t'-1} S_i x^{[i]}$, $\tilde{\Delta}_{2t'+t}^{(0)} = \tilde{\Theta}_{2t'+t}^{(0)} = 1, \Gamma^{(0)} = 1$, and $b = 0$.
    6.2 For $r = 0, 1, \ldots, 2t' - 1$,
      a) Modify the combined polynomial: $\tilde{\Delta}^{(r+1)}(x) = \Gamma^{(r)} \tilde{\Delta}^{(r)}(x) - \tilde{\Delta}_0^{(r)} \tilde{\Theta}^{(r)}(x)$;
      b) Set $b = b + 1$;
      c) If $\tilde{\Delta}_0^{(r)} \neq 0$ and $b > 0$, set $b = -b$, $\Gamma^{(r+1)} = \tilde{\Delta}_0^{(r)}$, and $\tilde{\Theta}^{(r)}(x) = \tilde{\Delta}^{(r)}(x)$;
      d) Set $\tilde{\Delta}^{(r+1)}(x) = \sum_{i=0}^{2t'+t-1} \tilde{\Delta}_{i+1}^{(r+1)} x^{[i]}$, $\tilde{\Theta}^{(r)}(x) = \sum_{i=0}^{2t'+t-1} \tilde{\Theta}_{i+1}^{(r)} x^{[i]}$;
      e) Set $\Gamma^{(r+1)} = (\Gamma^{(r)})^{[1]}$ and $\tilde{\Theta}^{(r+1)}(x) = x^{[1]} \otimes \tilde{\Theta}^{(r)}(x)$.
    6.3 Set $\Lambda(x) = \sum_{i=0}^{t'} \tilde{\Delta}_{i+t}^{(2t')} x^{[i]}$.

Compared with Algorithm 5, we replace t by t'. The variable t' makes it difficult to design regular architectures. By carefully initializing $\tilde{\Delta}^{(0)}(x)$ and $\tilde{\Theta}^{(0)}(x)$, we ensure that the desired output $\Lambda(x)$ is always at a fixed position of $\tilde{\Delta}^{(2t')}(x)$, regardless of $\mu + \delta$. Hence, the only irregular part is the initialization. The initialization of Algorithm 6 can be done by shifting in at most θ cycles. Hence, the RiBMA architecture in Fig. 3 can be adapted to the KK decoder and keep the same worst case latency of $2t$ cycles.

C. Gaussian Elimination

We need Gaussian elimination to obtain n-RRE forms as well as to find root spaces. Furthermore, Gabidulin's algorithm in Algorithm 1 is essentially a smart way of Gaussian elimination, which takes advantage of the properties of the matrix. The reduction (to obtain n-RRE forms) and finding the root space are
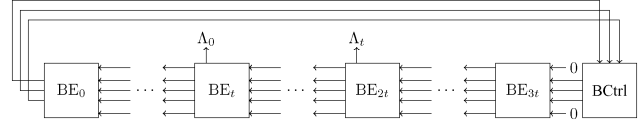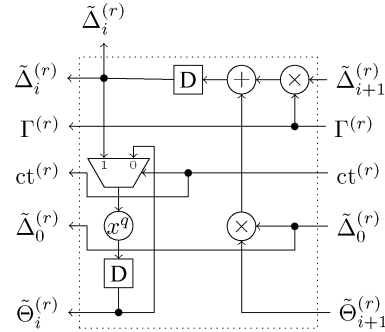

Fig. 3. RiBMA architecture.

Fig. 4. Processing element BE_i (x^q is a cyclic shift and requires no hardware but wiring).

Gaussian eliminations on matrices over $\mathbb{F}_q$, while Gabidulin's algorithm operates on matrices over $\mathbb{F}_{q^m}$. Here, we focus on Gaussian eliminations over $\mathbb{F}_q$ and Gabidulin's algorithm will be discussed in Section IV-D.

For high-throughput implementations, we adapt the pivoting architecture in [31], which was developed for non-singular matrices over $\mathbb{F}_2$. It always keeps the pivot element on the top-left location of the matrix, by cyclically shifting the rows and columns. Our Gaussian elimination algorithm, shown in Algorithm 7, has three key differences from the pivoting architecture in [31]. First, Algorithm 7 is applicable to matrices over any field. Second, and more importantly, Algorithm 7 can be used for singular matrices. This feature is necessary since singular matrices occur in the reduction for the RRE form and finding the root space. Third, Algorithm 7 is also flexible about matrix sizes, which are determined by the variable numbers of errors, erasures, and deviations.

Algorithm 7 (Gaussian Elimination for Root Space).

Input: $\boldsymbol{M} \in \mathbb{F}_q^{m \times m}$, whose rows are evaluations of $\sigma(x)$ over the normal basis, and $\boldsymbol{B} = \boldsymbol{I_m}$

Output: Linearly independent roots of $\sigma(x)$

    7.1 Set $i = 0$.
    7.2 For $j = 0, 1, \ldots, m - 1$
      a) $l = 1$
      b) While $M_{0,0} = 0$ and $l < m - i$
        $l = l + 1$, shiftup($\boldsymbol{M}, i$), and shiftup($\boldsymbol{B}, i$).
      c) If $M_{0,0}$ is not zero, eliminate($\boldsymbol{M}$), reduce($\boldsymbol{B}, \boldsymbol{M}$), and $i = i + 1$; Otherwise, shiftleft($\boldsymbol{M}$).
    7.3 The first $m - i$ rows of $\boldsymbol{M}$ are all zeros and the first $m - i$ rows of $\boldsymbol{B}$ are roots.

The eliminate and shiftup operations are quite similar to those in [31, Algorithm 2]. In eliminate($\boldsymbol{M}$), for $0 \leq j < m$, $M_{i,j} = M_{0,0} M_{i+1,(j+1) \bmod m} - M_{i+1,0} M_{0,(j+1) \bmod m}$ for $0 \leq i < m - 1$, and $M_{m-1,j} = M_{0,(j+1) \bmod m}$. Note that a cyclic row shift and a cyclic column shift are already embedded in the eliminate operation. In the shiftup($\boldsymbol{M}, \rho$) operation, the first row
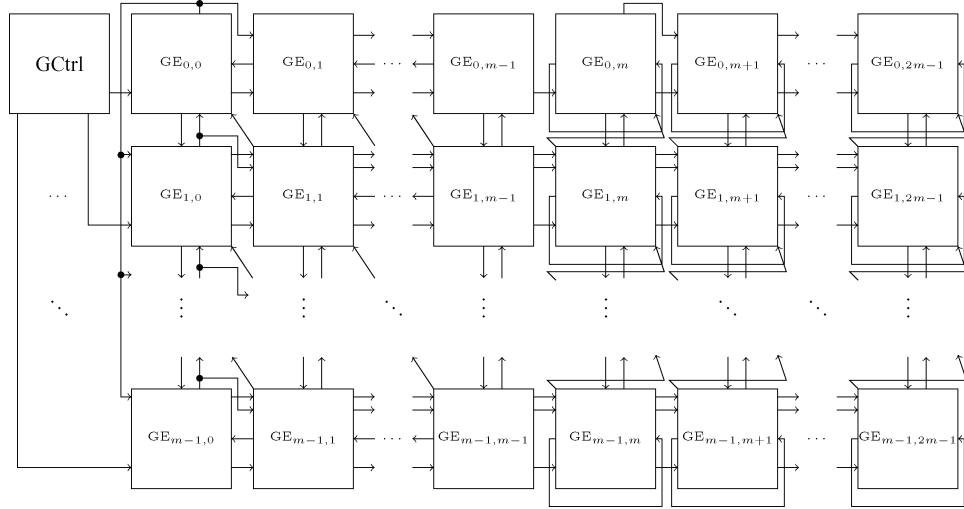
Fig. 5.   Regular architecture for Gaussian elimination.

is moved to the $(m - 1 - \rho)$th row while the second to the $(m - 1 - \rho)$th rows are moved up. That is, for $0 \leq j < m$, $M_{i,j} = M_{0,j}$ if $i = m - 1 - \rho$, and $M_{i,j} = M_{i+1,j}$ for $0 \leq i \leq m - 2 - \rho$. The operation reduce$(\boldsymbol{B}, \boldsymbol{M})$ essentially mimics all row operations in eliminate without the column shift: for $0 \leq j < m$, $B_{i,j} = M_{0,0} B_{i+1,j} - M_{i+1,0} B_{0,j}$ for $0 \leq i < m - 1$, and $B_{m-1,j} = B_{0,j}$. In the shiftleft operation, all columns are cyclically shifted to the left. In other words, for all $0 \leq i < m$ and $0 \leq j < m$, $M_{i,j} = M_{i,(j+1) \bmod m}$. By adding a shiftleft operation, Algorithm 7 handles both singular and nonsingular matrices while [31, Algorithm 2] works for nonsingular matrices only. Since $\boldsymbol{B}$ is always full rank, the roots obtained are guaranteed to be linearly independent.

We can get the root space using Algorithm 7, and we can also use it in KK decoding to reduce the received vector to an n-RRE form. However, Algorithm 7 only produces $\hat{\boldsymbol{E}}'$. We extend it to Algorithm 8 below so as to obtain $\hat{\boldsymbol{L}}'$ simultaneously.

Algorithm 8 (Gaussian Elimination for n-RRE Forms).

Input: $N \times n$ matrix $\hat{\boldsymbol{A}}$ and $N \times m$ matrix $\boldsymbol{y}$

Output: $\hat{\boldsymbol{L}}'$, $\hat{\boldsymbol{E}}'$, $\boldsymbol{r}'$, and μ'

   8.1 Set $i = 0$, $\mathcal{U}'$ and $\hat{\boldsymbol{L}}$ as empty.
   8.2 For each column $j = 0, 1, \ldots, n - 1$
    a) $l = 1$
    b) While $\hat{A}_{0,0} = 0$ and $l < n - i$
      $l = l + 1$, shiftup$(\hat{\boldsymbol{A}}, i)$, shiftup$(\boldsymbol{y}, i)$, shiftup$(\hat{\boldsymbol{L}}', i)$.
    c) If $\hat{A}_{0,0}$ is not zero, eliminate$(\hat{\boldsymbol{A}})$, reduce$(\boldsymbol{y}, \hat{\boldsymbol{A}})$, shiftup$(\hat{\boldsymbol{L}}', 0)$, $i = i + 1$.
    d) Otherwise, shiftleft$(\hat{\boldsymbol{A}})$, append the first column of $\hat{\boldsymbol{A}}$ to $\hat{\boldsymbol{L}}'$, set the top-right element of $\hat{\boldsymbol{L}}'$ to one, and add j to $\mathcal{U}'$.
   8.3 Set $\mu' = n - i$. The deviations $\hat{\boldsymbol{E}}'$ are given by the first μ' rows of $\boldsymbol{y}$.
   8.4 For each column $j \in \mathcal{U}'$, shiftup$(\hat{\boldsymbol{L}}', j)$ and shiftup$(\boldsymbol{y}, j)$.
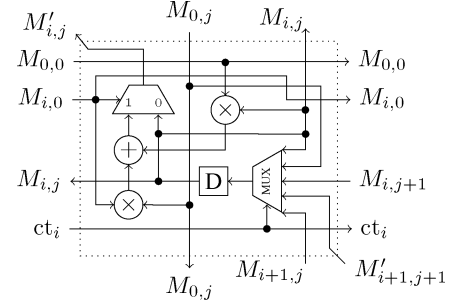   8.5 The received vector $\boldsymbol{r}'$ is given by $\boldsymbol{y}$.

Fig. 6.   Processing element $\mathrm{GE}_{i,j}$.

In Algorithm 8, we incorporate the extraction of $\hat{\boldsymbol{L}}'$, $\hat{\boldsymbol{E}}'$, and $\boldsymbol{r}'$ into Gaussian elimination. Our architecture has the same worst case latency as Algorithm 7 and requires no extra cycles to extract $\hat{\boldsymbol{L}}$ out of the n-RRE form. Hence, the throughput also remains the same.

Algorithm 7 is implemented by the regular architecture shown in Fig. 5, which is a 2-D array of $m \times 2m$ processing elements (GE's). The leftmost m columns of processing elements correspond to $\boldsymbol{M}$, and the rightmost m columns $\boldsymbol{B}$. Algorithm 8 can be implemented with the same architecture with $N \times (n + m)$ GE's. The leftmost n columns of processing elements correspond to $\hat{\boldsymbol{A}}$, and the rightmost m columns $\boldsymbol{y}$. The elements for $\hat{\boldsymbol{L}}'$ are omitted in the figure. The circuitry of the processing element GE is shown in Fig. 6. The control signal ct_i for row i chooses from five inputs based on the operation: keeping the value, shiftleft, eliminate (or reduce), and shiftup (using the first row or the next row).

D. Gabidulin's Algorithm

In Algorithm 1, the matrix is first reduced to a triangular form. It takes advantage of the property of the matrix so that it requires no division in the first stage. In the first stage, we need to perform elimination on only one row. We use a similar pivoting scheme like Algorithm 7. When a row is reduced to having only one nonzero element, a division is used to obtain one coefficient of $\boldsymbol{X}$. Then it performs a backward elimination after getting each coefficient. Hence, we introduce a backward pivoting scheme, where the pivot element is always at the bottom-right corner.
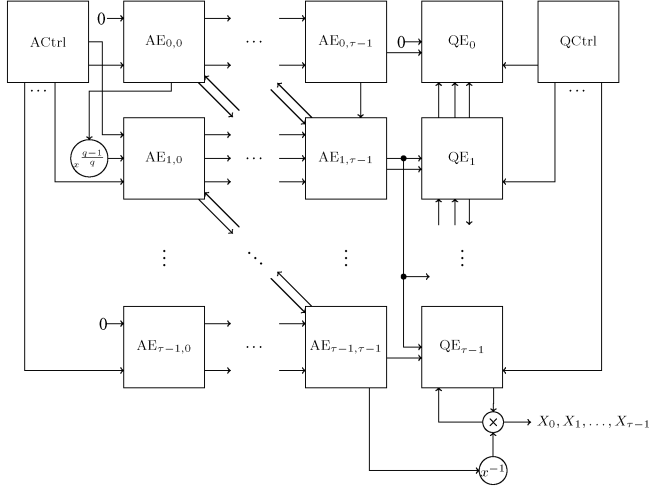
Fig. 7. Our architecture of Gabidulin's algorithm.

Fig. 8. Processing element $AE_{i,j}$.

Fig. 9. Processing element QE_i.

In Algorithm 1, there are two $\tau \times \tau$ matrices over $\mathbb{F}_{q^m}$, $\boldsymbol{A}$ and $\boldsymbol{Q}$. In step 1.2, it requires only $Q_{i,0}$'s to compute the coefficients. To compute $Q_{i,0}$ in (5), it requires only $Q_{i-1,0}$ and $Q_{i-1,1}$. And for $Q_{i,j}$ in (5), it requires only $Q_{i-1,j}$ and $Q_{i-1,j+1}$. Recursively, only those $Q_{i,j}$'s where $i + j < \tau$ are necessary. Actually, given any i, entries $Q_{i,0}, Q_{i+1,0}, \ldots, Q_{\tau-1,0}$ can be computed with the entries $Q_{i-1,0}, Q_{i-1,1}, \ldots, Q_{i-1,\tau-i}$. With $Q_{0,0}, Q_{1,0}, \ldots, Q_{i-2,0}$, we need to store only τ values to keep track of $\boldsymbol{Q}$. Hence, we reduce the storage of $\boldsymbol{Q}$ from $\tau \times \tau$ m-bit registers down to τ. We cannot reduce the storage of $\boldsymbol{A}$ to $\tau(\tau + 1)/2$ because we have to use the pivoting scheme for short critical paths.

In our decoder, Algorithm 1 is implemented by the regular architecture shown in Fig. 7, which includes an array of $\tau \times \tau$ AE's and a one-dimensional array of τ QE's. The circuitry of the processing element $AE_{i,j}$ and QE_i is shown in Figs. 8 and 9. The upper MUX in AE controls the output sending upward along the diagonal. Its control signal ctau_i is 1 for the second row and 0 for other rows since we update $\boldsymbol{A}$ one row in a cycle and we keep the pivot on the upper left corner in Step 1.1. The control of the lower MUX in AE is 0 for working on Step 1.1, and 1 for working on Step 1.2. Similarly the control of the MUX in QE is 0 for working on Step 1.1, and 1 for working on Step 1.2. However, in Step 1.1, only part of QE's need update and others should maintain their values and their control signals ctq_i's are set to 2. Initially, $A_{0,i} = E_i$ and $q_i = S_i$ for $i = 0, 1, \ldots, \tau - 1$. Step 1.1 needs τ substeps. In the first $\tau - 1$ substeps, $\text{ctal}_{i+1} = 0$, $\text{ctau}_1 = 1$, $\text{ctq}_0 = \text{ctq}_1 = \ldots = \text{ctq}_i = 2$, and $\text{ctq}_{i+1} = \text{ctq}_{i+2} = \ldots = \text{ctq}_{\tau-1} = 0$ for substep i. In the last substep, $\text{ctau}_1 = 0$ and all ctq_i's are set to 2. This substep is to put the updated $\boldsymbol{A}$ into the original position. In Step 1.2, the pivot is in the right lower corner, where we compute X_i's. Step 1.2 also needs τ substeps, in which all ctal_i's and ctq_i's are set to 1. First $X_{\tau-1}$ is computed by $A_{\tau-1,\tau-1}^{-1} q_\tau - 1$ where $q_{\tau-1} = Q_{\tau-1,0}$. Note that the inversion may need $m - 2$ clock cycles. In each substep, the matrix $\boldsymbol{A}$ is moving down the diagonal so the $A_{i,i}$ to be inverted is always at the bottom right corner. At the same time, the q_i's are also moving down. Basically, in substep p, the architecture updates q_i's to $Q_{i-p,0} - \sum_{j=\tau-1-p}^{\tau-1} A_{i,j} X_j$ for $i > p$ by doing one backward elimination at each substep.
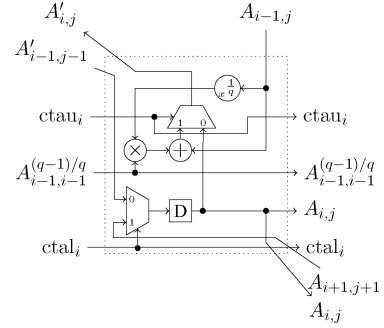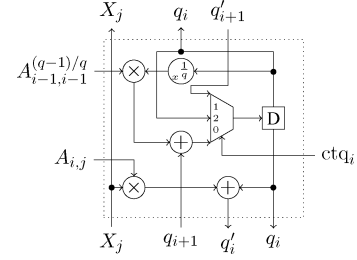
E. Low-Complexity Linearized Interpolation

It would seem that three registers are needed to store $F^{(i)}(x)$, w_j's, and $\gamma_j^{(i)}$'s, respectively, in Algorithm 4. However, we can implement Algorithm 4 with a single register of size $p + 1$. First, we note that w_j's are used to initialize $\gamma_j^{(0)}$'s, and only $\gamma_j^{(i)}$'s are used in the updates. Second, after the ith iteration of step 4.2, the q-degree of $F^{(i+1)}(x)$ is no more than $i + 1$ and we need only $\gamma_{i+1}^{(i+1)}, \gamma_{i+2}^{(i+1)}, \ldots, \gamma_{p-1}^{(i+1)}$ thereafter. Thus, we can store the coefficients of $F^{(i+1)}(x)$ and $\gamma_{i+1}^{(i+1)}, \gamma_{i+2}^{(i+1)}, \ldots, \gamma_{p-1}^{(i+1)}$ in a register of size $p + 1$. We refer to this register as η and index it $0, 1, \cdots, p$ from left to right. Note that $\gamma_{i+1}^{(i+1)}, \gamma_{i+2}^{(i+1)}, \ldots, \gamma_{p-1}^{(i+1)}$ are stored at the lower end of the η register, and the coefficients of $F^{(i+1)}(x)$ are stored at the higher end of the register. At each iteration, the content of the η register is shifted to the left by one position, so that $\gamma_i^{(i)}$ is always stored at η_0.

Algorithm 9 (Reformulated Algorithm for Minimal Linearized Polynomials).

Input: Roots $w_0, w_1, \ldots, w_{p-1}$

Output: The minimal linearized polynomial $F(x)$

    9.1 Initialization: $\eta_j^{(0)} = w_j$ for $j = 0, 1, \ldots, p - 1$, and $\eta_p^{(0)} = 1$.

    9.2 For $i = 0, 1, \ldots, p - 1$,

    a) If $\eta_0^{(i)} \neq 0$,

        i) For $j = 0, 1, \ldots, p - 1 - i$,

$$\eta_j^{(i+1)} = (\eta_{j+1}^{(i)})^{[1]} - (\eta_0^{(i)})^{q-1} \eta_{j+1}^{(i)};$$

        ii) For $j = p - i, p - i + 1, \ldots, p$,

$$\eta_j^{(i+1)} = (\eta_j^{(i)})^{[1]} - (\eta_0^{(i)})^{q-1} \eta_{j+1}^{(i)};$$

    b) Otherwise, for $j = 0, 1, \ldots, p$, $\eta_j^{(i+1)} = \eta_{j+1}^{(i)}$.

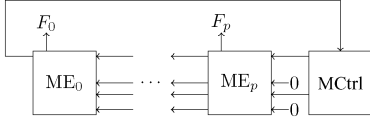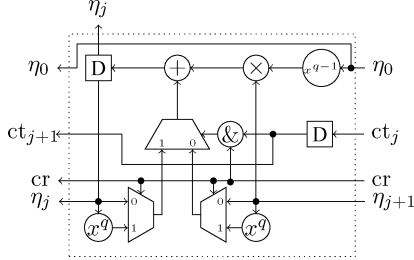   9.3 $F(x) = \sum_{i=0}^{p} \eta_i^{(p)} x^{[i]}$.

Fig. 10. Architecture of linearized polynomial interpolation.

Fig. 11. Processing element ME_j (x^q is a cyclic shift, and requires no hardware but wiring). For simplicity, we have omitted the superscripts of η_j.

We note that the updates involve $\eta_{p+1}^{(i)}$, which is always set to zero (see Fig. 10). When an input w_i is not linearly independent with $w_0, w_1, \ldots, w_{i-1}, \eta_0^{(i)} = 0$. In this case, the algorithm simply ignores the input, and the η_i registers are shifted to the left by one position. Hence, whether or not the inputs $w_0, w_1, \ldots, w_{p-1}$ are linearly independent, the minimal linearized polynomial for the inputs will be available after p iterations. This flexibility is important for our decoder architecture, since the number of linearly independent inputs varies.

Algorithm 9 is implemented by the systolic architecture shown in Fig. 10, which consists of $p + 1$ processing elements (ME's). The circuitry of the processing element ME_j is shown in Fig. 11. The cr signal is 1 only when $\gamma_0 \neq 0$. The ct_j signal for each cell is 1 only if $j < p - i$. Basically, ct_j controls whether the update is for $F^{(i+1)}(x)$ or $\gamma_{i+1}^{(i+1)}, \gamma_{i+2}^{(i+1)}, \ldots, \gamma_{p-1}^{(i+1)}$ as in Algorithm 4.

F. Decoding Failure

A complete decoder declares decoding failure when no valid codeword is found within the decoding radius of the received word. To the best of our knowledge, decoding failures of Gabidulin and KK codes were not discussed in previous works. Similar to RS decoding algorithms, a rank decoder can return decoding failure when the roots of the error span polynomial $\lambda(x)$ are not unique. That is, the root space of $\lambda(x)$ has a dimension smaller than the q-degree of $\lambda(x)$. Note that this applies to both Gabidulin and KK decoders. For KK decoders, another condition of decoding failure is when the total number of erasures and deviations exceeds the decoding bound $d - 1$.

G. Latency and Throughput

We analyze the worst case decoding latencies of our decoder architectures, in terms of clock cycles, in Table III.

As in [31], the latency of Gaussian elimination for the n-RRE form is at most $n(2N - n + 1)/2$ cycles. Similarly, the latency of finding the root space is at most $m(m + 1)/2$.

For Gabidulin's algorithm, it needs one cycle per row for forward elimination and the same for backward elimination. For each coefficient, it takes m cycles to perform a division. Hence, it needs at most $2(d - 1) + m(d - 1)$ and $2(d - 1) + m(d - 1)$ for $\boldsymbol{\beta}$ and $\boldsymbol{L}$ respectively. The latencies of finding the minimal

TABLE III
WORST CASE DECODING LATENCY (IN TERMS OF CLOCK CYCLES). GAUSSIAN ELIMINATION OVER $\mathbb{F}_{q^m}$ (ROOT SPACE IN GABIDULIN AND KK DECODERS) HAS THE LONGEST CRITICAL PATH OF ONE MULTIPLIER, ONE ADDER, ONE TWO-INPUT MUX, AND ONE FIVE-INPUT MUX

	Gabidulin	KK
n-RRE	-	$n(2N - n + 1)/2$
Syndrome $\boldsymbol{S}$	n	n
$\lambda_U(x)$	-	$2t$
$\sigma_D(x)$	-	$2t$
$S_{DU}(x)$	-	$2(d - 1)$
BMA	$2t$	$2t$
$S_{FD}(x)$	-	$d - 1$
$\boldsymbol{\beta}$	-	$(m + 2)(d - 1)$
$\sigma_U(x)$	-	$2t$
$\sigma(x)$	-	$d - 1$
root space basis $\boldsymbol{E}$	$m(m + 1)/2$	$m(m + 1)/2$
error locator $\boldsymbol{L}$	$2t + mt$	$(m + 2)(d - 1)$
error word $\boldsymbol{e}$	t	$2t$

linearized polynomials are determined by the number of registers, which is $2t$ to accommodate $\lambda_D(x)$, $\sigma_D(x)$, and $\sigma_U(x)$, whose degrees are μ, δ, and μ, respectively. The $2t$ syndromes can be computed by $2t$ sets of multiply-and-accumulators in n cycles. Note that the computations of $S(x)$, $\lambda_U(x)$, and $\sigma_D(x)$ can be done concurrently. The latency of RiBMA is $2t$ for $2t$ iterations. The latency of a symbolic product $a(x) \otimes b(x)$ is determined by the q-degree of $a(x)$. When computing $S_{DU}(x)$, we are concerned about only the terms of q-degree less than $d - 1$ because only those are meaningful for the key equation. For computing $S_{FD}(x)$, the result of $\sigma_D(x) \otimes S(x)$ in $S_{DU}(x)$ can be reused, so it needs only one symbolic product. In total, assuming $n = m$, the decoding latencies of our Gabidulin and KK decoders are $n(n+3)/2 + (n+5)t$ and $n(N+2) + 4(n+5)t$ cycles, respectively.

One assumption in our analysis is that the unit that computes x^{q-1} in Figs. 9 and 11 is implemented with pure combinational logic, which leads to a long CPD for large q's. To achieve a short CPD for large q's, it is necessary to pipeline the unit that computes x^{q-1}. There are two ways to pipeline it: $x^{q-1} = x \cdot x^2 \ldots x^{q/2}$ that requires $\log_2 q - 1$ multiplications, or $x^{q-1} = x^q/x$ that requires m multiplications for division. To maintain a short CPD, x^{q-1} needs to be implemented sequentially with one clock cycle for each multiplication. Let $c_{qm} = \min\{\log_2 q - 1, m\}$ and it requires at most $2(c_{qm} + 2)t$ clock cycles for getting minimal linearized polynomials $\lambda_U(x)$, $\sigma_D(x)$, and $\sigma_U(x)$. Similarly, it requires at most $c_{qm}(d - 1)$ more cycles to perform forward elimination in Gabidulin's algorithm for the error locator, and the latency of this step will be $(m + c_{qm} + 2)(d - 1)$ cycles.

In our architectures, we use a block-level pipeline scheme for high throughput. Data transfers between modules are buffered into multiple stages so the throughput is determined by only the longest latency of a single module. For brevity, we present only the data flow of our pipelined Gabidulin decoder in Fig. 12. The data in different pipeline stages are for different decoding sessions. Hence, these five units can work on five different sessions currently for higher throughput. If some block finishes before others, it cannot start another session until all are finished. So the throughput of our block-level pipeline decoders is determined by the block with the longest latency. For Gabidulin decoders, the block of finding root space is the bottleneck that requires
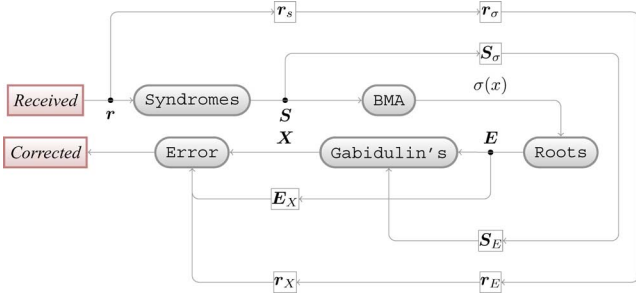
Fig. 12. Data flow of our pipelined Gabidulin decoder.

TABLE IV
SYNTHESIS RESULTS OF DECODERS FOR GABIDULIN AND KK CODES

Finite fields		$\mathbb{F}_{2^8}$		$\mathbb{F}_{2^{16}}$	
Codes		Gab.	KK	Gab.	KK
(n, k) or (n, m)		(8, 4)	(4, 4)	(16, 8)	(8, 8)
Gates		18465	71134	116413	421477
Area (mm^2)	Cell	0.035	0.133	0.219	0.791
	Net	0.053	0.202	0.320	1.163
	Total	0.088	0.335	0.539	1.954
CPD (ns)		2.309	2.199	3.490	3.617
Estimated Power (mW)	Leakage	0.281	1.084	1.690	6.216
	Dynamic	14.205	54.106	97.905	313.065
	Total	14.486	55.190	99.595	319.281
Latency (cycles)		70	216	236	752
Bottleneck (cycles)		36	68	136	264
Throughput (Mbit/s)		385	214	270	134

$m(m + 1)/2$ cycles, the longest latency in the worst case scenario. For KK decoders, the bottleneck is the RRE block, which requires $n(2N - n + 1)/2$ cycles.

V. IMPLEMENTATION RESULTS AND DISCUSSIONS

To evaluate the performance of our decoder architectures, we implement our architectures for Gabidulin and KK codes for RLNC over $\mathbb{F}_2$. Note that, although the random linear combinations are carried out over $\mathbb{F}_2$, decoding of Gabidulin and KK codes are performed over extension fields of $\mathbb{F}_2$.

We restrict N, the number of received packets, to save hardware while maintaining the error correction capability. We note that a large N leads to more rows in the architecture in Fig. 5. Note that we assume the input matrix is full rank as [5]. When $N \geq n + d$, the number of deviations $\delta = N - n$ is at least d and it is uncorrectable. Hence, in our implementation of KK decoders, we assume N is less than $n + d$.

A. Implementation Results

We implement our decoder architecture in Verilog for an (8, 4) Gabidulin code over $\mathbb{F}_{2^8}$ and a (16, 8) one over $\mathbb{F}_{2^{16}}$, which can correct errors of rank up to two and four, respectively. We also implement our decoder architecture for their corresponding KK codes, which can correct ϵ errors, μ erasures, and δ deviations as long as $2\epsilon + \mu + \delta$ is no more than five or nine, respectively. Our designs are synthesized using Cadence RTL Compiler 9.1 and FreePDK 45 nm standard cell library [32]. The synthesis results are given in Table IV. In these tables, the total area includes both cell area and estimated net area, the gate counts are in equivalent numbers of two-input NAND gates, and the total power includes both leakage and estimated dynamic power. All estimations are made by the synthesis tool. The throughput is computed as $(n \times m \times R)/(\text{Latency}_{\text{Bottleneck}} \times \text{CPD})$.

TABLE V
PERFORMANCE OF KK DECODERS FOR 512-byte PACKETS

(n, m)	(4, 4)		(8, 8)	
Decoder	Serial	7-Parallel	Serial	5-Parallel
Gates	71134	497938	421477	2107385
Area (mm^2)	0.335	2.345	1.954	9.770
CPD (ns)	2.199		3.617	
Est. Power (mW)	55.190	386.330	319.281	1596.405
Latency (cycles)	34896	5112	67808	13952
Throughput (Mbit/s)	214	1498	134	670

To provide a reference for comparison, the gate count of our (8, 4) KK decoder is only 62% to that of the (255, 239) RS decoder over the same field $\mathbb{F}_{2^8}$ in [33], which is 115,500. So for Gabidulin and KK codes over small fields, which have limited error-correcting capabilities, their hardware implementations are feasible. The area and power of decoder architectures in Table IV appear affordable except for applications with very stringent area and power requirements.

B. Implementation Results of Long Codes

Although the area and power shown in Table IV are affordable and high throughputs are achieved, the Gabidulin and KK codes used have very limited block lengths 8 and 16. For practical network applications, the packet size may be large [11]. One approach to increase the block length of a CDC is to lift a Cartesian product of Gabidulin codes [5]. We also consider the hardware implementations for this case. We assume a packet size of 512 bytes, and use a KK code that is based on Cartesian product of 511 length-8 Gabidulin codes. As observed in Section III-E, the n-RRE form allows us to either decode this long KK code in a serial, partly parallel, or fully parallel fashion. For example, more decoder modules can be used to decode in parallel for higher throughput. We list the gate counts and throughput of the serial and factor-7 parallel schemes based on the (8, 4) KK decoder and those of the serial and factor-5 parallel schemes based on the (16, 8) KK decoder in Table V.

In Table V, we simply use multiple KK decoders for parallel implementations. Parallel KK decoders actually share the same $\hat{A}, \hat{L}, \hat{X}$, and $\lambda_U(x)$. Hence, some hardware can be also shared, such as the left part of Gaussian elimination for reduction in Fig. 6 and the interpolation block for $\lambda_U(x)$. With the same latency, these hardware savings are roughly 7% of one single KK decoder.

C. Discussions

Our implementation results above show that the hardware implementations of RLNC over small fields and with limited error control are quite feasible, unless there are very stringent area and power requirements. However, small field sizes imply limited block length and limited error control. As shown above, the block length of a constant-dimension code can be increased by lifting a Cartesian product of Gabidulin codes. While this easily provides arbitrarily long block length, it does not address the limited error control associated with small field sizes. For example, a Cartesian product of (8, 4) Gabidulin codes has the same error correction capability as the (8, 4) KK decoder, and their corresponding CDCs also have the same error-correction capability. If we want to increase the error correction capabilities of both Gabidulin and KK codes, longer codes are needed

and in turn larger fields are required. A larger field size implies a higher complexity for finite field arithmetic, and longer codes with greater error correction capability also lead to higher complexity. It remains to be seen whether the decoder architectures continue to be affordable for longer codes over larger fields, and this will be the subject of our future work.

VI. CONCLUSION

This paper presents novel hardware architectures for Gabidulin and KK decoders. Our work not only reduces the computational complexity for the decoder but also devises regular architectures suitable for hardware implementations. Synthesis results using a standard cell library confirm that our designs achieve high speed and high throughput.

APPENDIX A
PROOF OF LEMMA 1

Proof: This follows the proof of [5, Proposition 7] closely. Suppose the RRE of $Y = [\hat{A} \mid y]$ is $\mathrm{RRE}(Y) = \begin{bmatrix} W & \tilde{r} \\ 0 & \hat{E} \end{bmatrix}$ and an n-RRE form of Y is $\bar{Y}' = \begin{bmatrix} W' & \tilde{r}' \\ 0 & \hat{E}' \end{bmatrix}$. Since the RRE form of $\hat{A}$ is unique, $W = W'$. Thus, $\mu = \mu'$ and $\delta = \delta'$. In the proof of [5, Proposition 7], $\mathcal{U}$ is chosen based on W. Thus, we choose $\mathcal{U} = \mathcal{U}'$. Since $\hat{L}$ is uniquely determined by W and $\hat{L}'$ is by W', we also have $\hat{L} = \hat{L}'$. Finally, choosing $r' = I_{\mathcal{U}'^c}\tilde{r}'$, the rest follows the same steps as in the proof of [5, Proposition 7]. ∎

APPENDIX B
PROOF OF LEMMA 2

Proof: This follows a similar approach as in [5, Appendix C]. We have the following.

$$
\mathrm{rank}\begin{bmatrix} X \\ Y \end{bmatrix} = \mathrm{rank}\begin{bmatrix} I + \overset{I'}{\hat{L}'I_{\mathcal{U}'}^T} & x \\ I + \hat{L}'I_{\mathcal{U}'}^T & r' \\ 0 & \hat{E}' \end{bmatrix}
$$

$$
= \mathrm{rank}\begin{bmatrix} \hat{L}'I_{\mathcal{U}'}^T & r' - x \\ I_{\mathcal{U}'^c}^T\left(I + \hat{L}'I_{\mathcal{U}'}^T\right) & I_{\mathcal{U}'^c}^T r' \\ 0 & \hat{E}' \end{bmatrix} \quad (8)
$$

$$
= \mathrm{rank}\begin{bmatrix} \hat{L}'I_{\mathcal{U}'}^T & r' - x \\ 0 & \hat{E}' \end{bmatrix} + \mathrm{rank}\begin{bmatrix} I_{\mathcal{U}'^c}^T & I_{\mathcal{U}'^c}^T x \end{bmatrix}
$$

$$
= \mathrm{rank}\begin{bmatrix} \hat{L}' & r' - x \\ 0 & \hat{E}' \end{bmatrix} + n - \mu' \quad (9)
$$

where (8) follows from $I_{\mathcal{U}'}^T[I + \hat{L}'I_{\mathcal{U}'}^T|r] = 0$ and (9) follows from $I_{\mathcal{U}'}^T I_{\mathcal{U}'^c} = 0$. Since $\mathrm{rank} X + \mathrm{rank} Y = 2n - \mu' + \delta'$, the subspace distance is given by $d_S(\langle X \rangle, \langle Y \rangle) = 2\mathrm{rank}\begin{bmatrix} X \\ Y \end{bmatrix} - \mathrm{rank} X - \mathrm{rank} Y = 2\mathrm{rank}\begin{bmatrix} \hat{L}' & r' - x \\ 0 & \hat{E}' \end{bmatrix} - \mu' - \delta'$. ∎

ACKNOWLEDGMENT

The authors would like to thank Dr. D. Silva and Prof. F. R. Kschischang for valuable discussions and the reviewers for their constructive comments.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[2] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[3] R. Kötter and F. R. Kschischang, "Coding for errors and erasures in random network coding," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3579–3591, Aug. 2008.

[4] N. Cai and R. W. Yeung, "Network coding and error correction," in *Proc. IEEE Inf. Theory Workshop (ITW'02)*, Oct. 20–25, 2002, pp. 119–122.

[5] D. Silva, F. R. Kschischang, and R. Kötter, "A rank-metric approach to error control in random network coding," *IEEE Trans. Inf. Theory*, vol. 54, no. 9, pp. 3951–3967, Sep. 2008.

[6] E. M. Gabidulin, "Theory of codes with maximum rank distance," *Probl. Inf. Transm.*, vol. 21, no. 1, pp. 1–12, Jan.–Mar. 1985.

[7] R. M. Roth, "Maximum-rank array codes and their application to criss-cross error correction," *IEEE Trans. Inf. Theory*, vol. 37, no. 2, pp. 328–336, Mar. 1991.

[8] D. Silva and F. R. Kschischang, "On metrics for error correction in network coding," *IEEE Trans. Inf. Theory*, vol. 55, no. 12, pp. 5479–5490, Dec. 2009.

[9] M. Gadouleau and Z. Yan, "Complexity of decoding Gabidulin codes," in *Proc. 42nd Ann. Conf. Inf. Sci. Syst. (CISS'08)*, Princeton, NJ, Mar. 19–21, 2008, pp. 1081–1085.

[10] F. R. Kschischang and D. Silva, "Fast encoding and decoding of Gabidulin codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT'09)*, Seoul, Korea, Jun. 28–Jul. 3 2009, pp. 2858–2862.

[11] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. 41st Ann. Allerton Conf. Commun., Control, and Computing*, Monticello, IL, Oct. 2003.

[12] G. Richter and S. Plass, "Error and erasure decoding of rank-codes with a modified Berlekamp-Massey algorithm," in *Proc. 5th Int. ITG Conf. Source and Channel Coding (SCC'04)*, Erlangen, Germany, Jan. 2004, pp. 249–256.

[13] N. Chen and Z. Yan, "Cyclotomic FFTs with reduced additive complexities based on a novel common subexpression elimination algorithm," *IEEE Trans. Signal Process.*, vol. 57, no. 3, pp. 1010–1020, Mar. 2009.

[14] H. Burton, "Inversionless decoding of binary BCH codes," *IEEE Trans. Inf. Theory*, vol. IT-17, no. 4, pp. 464–466, Jul. 1971.

[15] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed–Solomon decoders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 5, pp. 641–655, Oct. 2001.

[16] P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, ser. Comput. Sci. Appl. Math., 2nd ed. Orlando, FL: Academic, 1985.

[17] M. Gadouleau and Z. Yan, "On the decoder error probability of bounded rank-distance decoders for maximum rank distance codes," *IEEE Trans. Inf. Theory*, vol. 54, no. 7, pp. 3202–3206, Jul. 2008.

[18] M. Gadouleau and Z. Yan, "Decoder error probability of bounded distance decoders for constant-dimension codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT'09)*, Seoul, Korea, Jun. 28–Jul. 3 2009, pp. 2226–2230.

[19] M. Gadouleau and Z. Yan, "On the decoder error probability of bounded rank distance decoders for rank metric codes," in *Proc. IEEE Inf. Theory Workshop (ITW'09)*, Sicily, Italy, Oct. 11–16, 2009, pp. 485–489.

[20] P. Delsarte, "Bilinear forms over a finite field, with applications to coding theory," *J. Comb. Theory, Ser. A*, vol. 25, pp. 226–241, 1978.

[21] O. Ore, "On a special class of polynomials," *Trans. Amer. Math. Soc.*, vol. 35, no. 3, pp. 559–584, 1933.

[22] O. Ore, "Contributions to the theory of finite fields," *Trans. Amer. Math. Soc.*, vol. 36, no. 2, pp. 243–274, 1934.

[23] P. Loidreau, "A Welch-Berlekamp like algorithm for decoding Gabidulin codes," in *Proc. 4th Int. Workshop Coding and Cryptography (WCC'05)*, Bergen, Norway, Mar. 14–18, 2005, vol. 3969, Lecture Notes in Computer Science, pp. 36–45.

[24] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.

[25] V. Skachek and R. M. Roth, "Probabilistic algorithm for finding roots of linearized polynomials," *Des. Codes Cryptogr.*, vol. 46, no. 1, pp. 17–23, Jan. 2008.

[26] E. D. Mastrovito, "VLSI architectures for computations in Galois fields," Ph.D. dissertation, Dept. Electr. Eng., Linköping Univ., Linköping, Sweden, 1991.

[27] M. Wagh and S. Morgera, "A new structured design method for convolutions over finite fields—Part I," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 4, pp. 583–595, Jul. 1983.

[28] J. K. Omura and J. L. Massey, "Computational method and apparatus for finite field arithmetic," U.S. Patent 4 587 627, May 6, 1986.

[29] A. Reyhani-Masoleh and M. A. Hasan, "A new construction of Massey–Omura parallel multiplier over $GF(2^m)$," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 511–520, May 2002.

[30] E. R. Berlekamp, *Algebraic Coding Theory*, revised ed. Laguna Hills, CA: Aegean Park Pres, 1984.

[31] A. Bogdanov, M. C. Mertens, C. Paar, J. Pelzl, and A. Rupp, "A parallel hardware architecture for fast Gaussian elimination over GF(2)," in *Proc. 14th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM'06)*, Napa Valley, CA, Apr. 24–26, 2006, pp. 237–248.

[32] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Proc. IEEE Int. Conf. Microelectron. Syst. Education (MSE'07)*, San Diego, CA, Jun. 3–4, 2007, pp. 173–174.

[33] H. Lee, "High-speed VLSI architecture for parallel Reed–Solomon decoder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 2, pp. 288–294, Apr. 2003.

Ning Chen (S'06–M'10) received the B.E. and M.E. degrees from Tsinghua University, Beijing, China, in 2001 and 2004, respectively, and the Ph.D. degree from Lehigh University, in 2010, all in electrical engineering.

Currently he is with the Enterprise Storage Division, PMC-Sierra, Allentown, PA. His research interests are in the VLSI design and implementation of digital signal processing and communication systems.
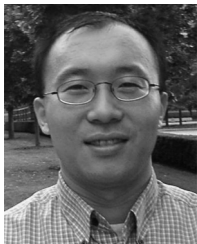
Zhiyuan Yan (S'00–M'03–SM'08) received the B.E. degree in electronic engineering from Tsinghua University, Beijing, China, in 1995, and the M.S. and Ph.D. degrees in electrical engineering from the University of Illinois, Urbana, in 1999 and 2003, respectively.

During the summers of 2000 and 2002, he was a Research Intern with Nokia Research Center, Irving, TX. He joined the Electrical and Computer Engineering Department, Lehigh University, Bethlehem, PA, as an Assistant Professor in August 2003. He has authored or coauthored over 70 technical papers in refereed journals and conference proceedings. He is an associate editor for the *Journal of Signal Processing Systems.* His current research interests are in coding theory, cryptography, wireless communications, and VLSI implementations of communication and signal processing systems.

Dr. Yan is a member of Tau Beta Pi, Sigma Xi, and Phi Kappa Phi. He served as technical program committee co-chair and general co-chair for ACM Great Lakes Symposium on VLSI in 2007 and 2008, respectively. He has served as an associate editor for the IEEE COMMUNICATIONS LETTERS since 2008. He is a member of the IEEE Information Theory, Communications, Signal Processing, and Computer Societies.

Maximilien Gadouleau (S'06–M'10) received the M.S. degree in electrical and computer engineering from Esigelec, Saint-Etienne du Rouvray, France, in 2004, and the M.S. and Ph.D. degrees in computer engineering from Lehigh University, Bethlehem, PA, in 2005 and 2009, respectively.

From 2009 to 2010, he was a Postdoctoral Researcher with the Université de Reims Champagne-Ardenne, Reims, France. In 2010, he joined the School of Electronic Engineering and Computer Science at Queen Mary, University of London, London, U.K., as a Postdoctoral Research Assistant. His current research interests are coding theory, network coding, and cryptography, and their relations to combinatorics and graph theory.

Dr. Gadouleau is a member of the IEEE Information Theory Society.

Ying Wang (S'00–M'06) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Urbana, IL, in 2006.

Currently, she is a Staff Engineer with the New Jersey Research Center of QUALCOMM Corporate Research and Development, Bridgewater, NJ. Her research interests include wireless communications, statistical signal processing, multimedia security and forensics, and detection and estimation theory.

Bruce W. Suter (S'80–M'85–SM'92) received the B.S. and M.S. degrees in electrical engineering and Ph.D. degree in computer science from the University of South Florida, Tampa, in 1972 and 1988, respectively.

In 1998, he joined the Technical Staff of the U.S. Air Force Research Laboratory, Rome, NY, where he was the founding Director of the Center for Integrated Transmission and Exploitation (CITE). He has held a visiting appointments at Harvard University and the Massachusetts Institute of Technology. His previous positions include academia at the U.S. Air Force Institute of Technology and the University of Alabama at Birmingham, together with industrial positions at Honeywell Inc. and Litton Industries. He is the author or coauthor of over 100 technical publications and the author of a widely accepted monograph, *Multirate and Wavelet Signal Processing* (Academic Press, 1998). His current research interests are focused on wireless computing networks and their applications to signal and image processing.

Dr. Suter is a member of Tau Beta Pi and Eta Kappa Nu. He was the recipient of the Air Force Research Laboratory (AFRL) Fellow, an AFRL-wide award for his accomplishments in the theory, application, and implementation of signal processing algorithms, the Arthur S. Flemming Award, a government-wide award for his pioneering Hankel transform research, the General Ronald W. Yates Award for Excellence in Technology Transfer for his patented Fourier transform processor, and the Fred I. Diamond Award for best laboratory research publication. He is a former associate editor of the IEEE TRANSACTIONS ON SIGNAL PROCESSING.