RANK METRIC DECODER ARCHITECTURES FOR NONCOHERENT ERROR CONTROL IN RANDOM NETWORK CODING

Ning Chen, Maximilien Gadouleau, and Zhiyuan Yan

Department of Electrical and Computer Engineering, Bethlehem, PA, 18015
E-mail: {nic6,magc,yan}@lehigh.edu

ABSTRACT

While random network coding has proved to be a powerful tool for disseminating information in networks, it is highly susceptible to errors caused by various sources. Recently, constant-dimension codes (CDCs), especially Kötter–Kschischang (KK) codes, have been proposed for error control in random network coding. It has been shown that KK codes can be constructed from Gabidulin codes, an important class of rank metric codes used in storage and cryptography. Although rank metric decoders have been proposed for both Gabidulin and KK codes, it is not clear whether such decoders are feasible and suitable for hardware implementations. In this paper, we propose **novel** decoder architectures for both codes. The synthesis results of our decoder architectures for Gabidulin and KK codes over small fields and with limited error-correcting capabilities not only are affordable, but also achieve high throughput.

1. INTRODUCTION

While random network coding has proved to be a powerful tool for disseminating information in networks, it is highly susceptible to errors caused by various sources such as noise, malicious or malfunctioning nodes, or insufficient min-cut [1]. Thus, error control for random network coding is critical and has received growing attention recently. Nearly optimal Reed–Solomon-like constant-dimension codes (CDCs) based on the subspace metric, called Kötter–Kschischang (KK) codes, were proposed in [1] for noncoherent error control in network coding. Later it was shown [2] that KK codes correspond to "liftings" [2] of Gabidulin codes [3]. As Reed–Solomon codes achieve the Singleton bound of Hamming distance, Gabidulin codes are a class of maximum rank distance (MRD) codes, which achieve the Singleton bound of rank distance. Due to the connection between Gabidulin and KK codes, the decoding of KK codes can be viewed as a generalized Gabidulin decoding, which involves errors, erasures, and deviations.

This work was supported in part by Thales Communications, Inc. and in part by a grant from the Commonwealth of Pennsylvania, Department of Community and Economic Development, through the Pennsylvania Infrastructure Technology Alliance (PITA).

Although the complexity of errors-only and generalized decoding of Gabidulin codes was analyzed [4], there are no hardware architectures for these decoders reported yet. Thus it remains unknown whether these decoding algorithms are feasible and suitable for hardware implementations. The feasibility of the generalized Gabidulin decoding algorithm in hardware implementations determines whether random network coding, along with error control, can be readily applied to certain applications.

In this paper, we propose decoder architectures for Gabidulin and KK codes. We first propose a high-throughput hardware architecture for errors-only Gabidulin decoding, then extend it to decode KK codes. To evaluate the performance of our decoder architectures, we implement our decoder architecture for an $(8, 4)$ Gabidulin code over $\mathbb{F}_{2^8}$, whose code length is the longest given the field. We also implement our decoder architecture for the corresponding KK code of the $(8, 4)$ Gabidulin code over $\mathbb{F}_{2^8}$, which can be used in network coding with various packet lengths by Cartesian product. The synthesis results of our decoder architectures show that decoder architectures for Gabidulin and KK codes over small fields and with limited error-correcting capabilities not only are affordable, but also achieve high throughput. Our decoder architectures are **novel** to the best of our knowledge.

2. PRELIMINARIES

2.1. Rank metric and Gabidulin Codes

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 the base field $\mathbb{F}_q$. Rank metric is the weight of vector difference [3].

A Gabidulin code [3] is a linear (n, k) code over $\mathbb{F}_{q^m}$ defined by a parity-check matrix

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

where $h_0, h_1, \ldots, h_{n-1} \in \mathbb{F}_{q^m}$ are linearly independent over

$\mathbb{F}_q$ and $[i]$ denotes q^i. Since $\mathbb{F}_{q^m}$ is an m-dimensional vector space over $\mathbb{F}_q$, we always have $n \leq m$. The minimum rank distance of a Gabidulin code is $d = n-k+1$ and all Gabidulin codes are MRD codes.

The decoding process of Gabidulin codes includes five major steps: syndrome computation, key equation solver based on a modified Berlekamp–Massey algorithm (BMA) [5], finding the root space, finding the error locators by Gabidulin's algorithm [3], and finding error locations. Note that all polynomials involved in the decoding process have only non-zero terms with degrees $[j]$, and such polynomials are called *linearized polynomials*. The greatest value of j of non-zero terms of a linearized polynomial is defined as its q-degree.

The data flow of a Gabidulin decoder is given in Figure 1. As in Reed–Solomon decoding, we can compute syndromes for Gabidulin codes as $\boldsymbol{S} = (S_0, S_1, \ldots, S_{d-2}) \triangleq \boldsymbol{Hr}$ 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 $\sigma(x) \otimes S(x) \equiv \omega(x) \bmod x^{[d-1]}$, where $\sigma(x)$ is the error span polynomial and $\otimes$ denotes the *symbolic product* of two linearized polynomials: $a(x) \otimes b(x) = a(b(x))$. After solving the key equation by the BMA, up to $t = \lfloor (d-1)/2 \rfloor$ error values E_j's can be obtained by computing a basis $E_0, E_1, \ldots$ for the root space of $\sigma(x)$ using the methods in [6, 7]. Then we can find the error locators X_j's corresponding to E_j's by solving a system of equations

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

where τ is the number of errors. After solving (1) using Gabidulin's algorithm, the error locations $\boldsymbol{L}_j$'s are revealed from X_j's by solving

$$X_j = \sum_{i=0}^{n-1} L_{j,k} h_i, \quad j = 0, 1, \ldots, t-1. \tag{2}$$

2.2. KK Codes

By lifting construction [2], KK codes can be constructed from MRD codes. Lifting can also be seen as a generalization of the standard approach to random network coding , which transmits matrices in the form $\boldsymbol{X} = [\boldsymbol{I} \mid \boldsymbol{x}]$, where $\boldsymbol{X} \in \mathbb{F}_q^{n\times M}$, $\boldsymbol{x} \in \mathbb{F}_q^{n\times m}$, and $m = M - n$. Hence by adding the constraint that $\boldsymbol{x}$ is the row expansion of codewords from a Gabidulin code $\mathcal{C}$ over $\mathbb{F}_{q^m}$, error control is enabled in network coding.

Let the received matrix be $\boldsymbol{Y} = [\hat{\boldsymbol{A}} \mid \boldsymbol{y}]$, where $\hat{\boldsymbol{A}} \in \mathbb{F}_q^{N\times n}$ and $\boldsymbol{y} \in \mathbb{F}_q^{N\times m}$. In [2], the matrix $\boldsymbol{Y}$ is first turned into a reduced row-echelon (RRE) form. Let $\mathcal{U}^c$ denote the column positions of leading entries in each row of $\mathrm{RRE}(\boldsymbol{Y})$ and $\mathcal{U} = \{0, 1, \ldots, n-1\} \setminus \mathcal{U}^c$. Let $\boldsymbol{I}_{\mathcal{U}^c}$ denote the columns of $\boldsymbol{I}_n$ in $\mathcal{U}^c$. Then the RRE form is expanded into $\bar{\boldsymbol{Y}} = \left[\begin{smallmatrix} \boldsymbol{I}_{\mathcal{U}^c} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_\delta \end{smallmatrix}\right] \mathrm{RRE}(\boldsymbol{Y}) = \left[\begin{smallmatrix} \boldsymbol{I}_n + \hat{\boldsymbol{L}}\boldsymbol{I}_{\mathcal{U}}^T & \boldsymbol{r} \\ \boldsymbol{0} & \hat{\boldsymbol{E}} \end{smallmatrix}\right]$ where $\delta = N - |\mathcal{U}|$. It was proved [2] that the subspace distance [1] $d_S(\langle \boldsymbol{X} \rangle, \langle \boldsymbol{Y} \rangle) = 2\,\mathrm{rank}\left[\begin{smallmatrix} \hat{\boldsymbol{L}} & \boldsymbol{r}-\boldsymbol{x} \\ \boldsymbol{0} & \hat{\boldsymbol{E}} \end{smallmatrix}\right] - \mu - \delta$ where $\mu = n - |\mathcal{U}|$ and $\langle \boldsymbol{X} \rangle$ and $\langle \boldsymbol{Y} \rangle$ are subspaces spanned by rows of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. Now the decoding problem to minimize the subspace distance becomes a problem to minimize the rank distance.

The generalized rank decoding finds an error word $\hat{\boldsymbol{e}} = \arg\min_{\boldsymbol{e} \in \boldsymbol{r} - \mathcal{C}} \mathrm{rank}\left[\begin{smallmatrix} \hat{\boldsymbol{L}} & \boldsymbol{e} \\ 0 & \hat{\boldsymbol{E}} \end{smallmatrix}\right]$. We can expand $\hat{\boldsymbol{e}}$ as a summation of products of column and row vectors 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, $\boldsymbol{L}$ has μ columns from $\hat{\boldsymbol{L}}$ and $\boldsymbol{E}$ has δ rows from $\hat{\boldsymbol{E}}$. Similar to Hamming weight decoding, 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 $2\epsilon + \mu + \delta < d$.

With $(\boldsymbol{r}, \hat{\boldsymbol{L}}, \hat{\boldsymbol{E}})$, the general Gabidulin decoding algorithm in [2, Fig. 1] can be used to decode the KK code. The data flow of a KK decoder is given in Figure 2. Like interpolation in errors-and-erasures RS decoding, the general Gabidulin decoding uses $\mathrm{minpoly}(\boldsymbol{\beta})$ to compute a *minimum linearized polynomial*, which satisfies two conditions: the elements of $\boldsymbol{\beta}$ are its roots and its q-degree is minimal.

3. ARCHITECTURE FOR GABIDULIN DECODING

In this section, we propose a novel decoder architecture for Gabidulin codes. We describe the key features of our decoder architecture below. In most practical applications, data are stored and transmitted in binary formats. Henceforth in this paper we assume $q = 2$.

3.1. Finite Field Arithmetic

VLSI architectures for finite field arithmetic have been well studied (see for example, [8, 9]). Finite field elements can be represented by vectors using different bases: polynomial basis, normal basis, and dual basis [8]. Under normal basis, squaring is simply cyclic shifting to more significant bits, which makes it very attractive in rank metric decoders since all polynomials involved are linearized polynomials. It was pointed out in [4] that using normal basis can facilitate the computation of symbolic product. It was also suggested that solving (2) can be trivial using normal basis.

There are additional savings due to normal basis. In Gabidulin's algorithm [3] to solve (1), the major complexity is for $A_{i,j} = A_{i-1,j} - (A_{i-1,j}/A_{i-1,i-1})^{[-1]} A_{i-1,i-1}$ and $Q_{i,j} = Q_{i-1,j} - (Q_{i-1,j+1}/A_{i-1,i-1})^{[-1]} A_{i-1,i-1}$, which requires divisions and finding square roots. Actually, when $q = 2$, they can be computed in an inversionless form $A_{i,j} = A_{i-1,j} - A_{i-1,j}^{[-1]} A_{i-1,i-1}^{[-1]}$ and $Q_{i,j} = Q_{i-1,j} - Q_{i-1,j+1}^{[-1]} A_{i-1,i-1}^{[-1]}$, which requires only finding square roots. Similar to squaring, finding square root in nor-

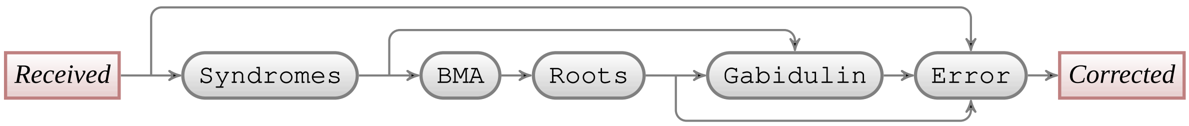

Fig. 1. Data flow of a Gabidulin decoder

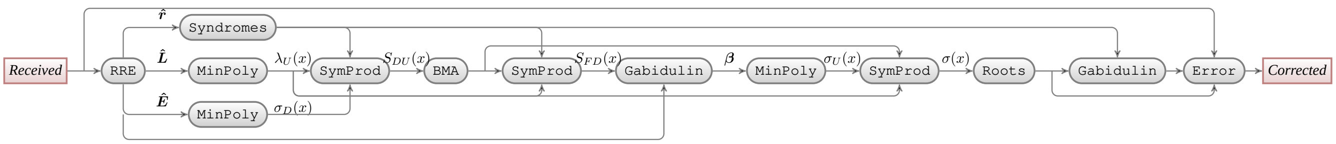

Fig. 2. Data flow of a KK decoder

mal basis is just cyclic shifting in a reverse direction. Thus using normal basis also reduces the complexity of Gabidulin's algorithm. If a normal basis of $\mathbb{F}_{2^m}$ is used as h_i's and $n = m$, $\boldsymbol{H}$ becomes a cyclic matrix and 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, and are favorable when m is large.

There are serial and parallel architectures for normal basis finite field multipliers. To achieve high throughput in our decoder, we consider only parallel architectures. The complexity of a normal basis, C_N, is defined as the number of terms a_ib_i in computing a bit of $c = ab$, where a_i's and b_i's are the bits of a and b, respectively. In this paper, we focus on the field $\mathbb{F}_{2^8}$ generated by $x^8 + x^7 + x^5 + x^3 + 1$, on which C_N is minimized to 21 [8]. Most normal basis multipliers are based on the Massey–Omura (MO) architecture. According to [9], a parallel MO multiplier needs $m^2 = 64$ AND gates and at most $m(C_N + m - 2)/2 = 108$ XOR gates. Using the common subexpression elimination algorithm from [10], we reduce the number of XOR gates to 88 while maintain the same critical path delay (CPD) as that of one AND gate and five XOR gates.

Since squaring is almost free, an efficient method to get the inverse of β is to find $\beta^{-1} = \beta^{2^m-2} = \beta^2\beta^4 \ldots$ based on multipliers. Division is simply the combination of inversion and multiplication.

3.2. BMA Architectures

The modified BMA for rank metric codes in [5] is similar to the BMA for Reed–Solomon codes except that polynomial multiplications are replaced by symbolic products.

The BMA in [5] requires finite field divisions, which are more time-consuming than other arithmetic operations. We first propose an inversionless variant in Algorithm 1, which is more suitable for hardware implementation.

Algorithm 1. `iBMA`
Input: Syndromes $\boldsymbol{S}$
Output: $\Lambda(x)$

1. Initialize as follows: $\Lambda^{(0)}(x) = B^{(0)}(x) = x$, $\Gamma^{(0)} = 1$, and $L = 0$.
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 $B^{(r+1)}(x) = x^{[1]} \otimes B^{(r)}(x)$.

To further increase the throughput, more regular architectures are necessary for shorter CPD. Following the approaches in [11], we develop two architectures based on Algorithm 1.

In Algorithm 1, the critical path is in step 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

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

which has the same CPD as step 2(c). This reformulation leads to a more regular architecture in Algorithm 2, which is analogous to the riBM architecture in [11]. Compared to Algorithm 1, its control flow is also simpler.

Algorithm 2. `riBMA`
Input: Syndromes $\boldsymbol{S}$
Output: $\Lambda(x)$

1. Initialize as follows: $\Lambda^{(0)}(x) = B^{(0)}(x) = x$, $\Gamma^{(0)} = 1$, $\Delta^{(0)}(x) = \Theta^{(0)}(x) = \sum_{i=0}^{2t-1} S_i x^{[i]}$, $k = 0$.

2. For $r = 0, 1, \ldots, 2t-1$,
   (a) Modify the connection polynomial: $\Lambda^{(r+1)}(x) = \Gamma^{(r)}\Lambda^{(r)}(x) - \Delta_0^{(r)} B^{(r)}(x)$;
   (b) Compute the discrepancy polynomial: $\Delta^{(r+1)}(x) = \Gamma^{(r)}\Delta^{(r)}(x) - \Delta_0^{(r)}\Theta(r)(x)$;
   (c) Set $k = k + 1$;
   (d) If $\Delta_0^{(r)} \neq 0$ and $k > 0$, set $k = -k$, $\Gamma^{(r+1)} = \Delta_0^{(r)}$, $B^{(r)}(x) = \Lambda^{(r)}(x)$, and $\Theta^{(r)}(x) = \Delta^{(r)}(x)$;
   (e) Set $\Delta^{(r+1)}(x) = \sum_{i=0}^{2t-2} \Delta_{i+1}^{(r+1)} x^{[i]}$, $\Theta^{(r+1)}(x) = \sum_{i=0}^{2t-2} \Theta_{i+1}^{(r)} x^{[i]}$;
   (f) Set $\Gamma^{(r+1)} = (\Gamma^{(r)})^{[1]}$, $B^{(r+1)}(x) = x^{[1]} \otimes B^{(r)}(x)$, and $\Theta^{(r+1)}(x) = x^{[1]} \otimes \Theta^{(r+1)}(x)$.

Given the similarities between steps 2(a) and 2(b), $\Lambda(x)$ and $\Delta(x)$ can be combined together into one polynomial $\tilde{\Delta}(x)$, which is more regular. Similarly, $B(x)$ and $\Theta(x)$ can be combined into one polynomial $\tilde{\Theta}(x)$. In Algorithm 3 we have the RiBMA architecture, which is closely related to the RiBM architecture in [11].

Algorithm 3. `RiBMA`
Input: Syndromes $\boldsymbol{S}$
Output: $\Lambda(x)$

1. Initialize as follows: $\tilde{\Delta}^{(0)}(x) = \tilde{\Theta}^{(0)}(x) = \sum_{i=0}^{2t-1} S_i x^{[i]}$, $\Gamma^{(0)} = 1$, $\tilde{\Delta}_{3t}^{(0)} = \tilde{\Theta}_{3t}^{(0)} = 1$, and $k = 0$.
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 $k = k + 1$;
   (c) If $\tilde{\Delta}_0^{(r)} \neq 0$ and $k > 0$, set $k = -k$, $\Gamma^{(r+1)} = \tilde{\Delta}_0^{(r)}$, and $\tilde{\Theta}^{(r)}(x) = \tilde{\Delta}^{(r)}(x)$;
   (d) Set $\tilde{\Delta}^{(r+1)}(x) = \sum_{i=0}^{3t-1} \tilde{\Delta}_{i+1}^{(r+1)} x^{[i]}$, $\tilde{\Theta}^{(r)}(x) = \sum_{i=0}^{3t-1} \tilde{\Theta}_{i+1}^{(r)} x^{[i]}$;
   (e) Set $\Gamma^{(r+1)} = (\Gamma^{(r)})^{[1]}$ and $\tilde{\Theta}^{(r+1)}(x) = x^{[1]} \otimes \tilde{\Theta}^{(r)}(x)$.
3. Set $\Lambda(x) = \sum_{i=0}^{t} \tilde{\Delta}_{i+t}^{(2t)} x^{[i]}$.

3.3. 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 Reed–Solomon 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 dimensions less than the q-degree of $\lambda(x)$.

Note that this applies to both Gabidulin and KK decoders. For KK decoders, another condition to declare decoding failure is when the total number of erasures and deviations exceeds $2t$.

4. ARCHITECTURE FOR KK CODES

4.1. Left-RRE Form

We first define a left-RRE form for matrices. Given a matrix $\boldsymbol{Y} = [\hat{\boldsymbol{A}} \mid \boldsymbol{y}]$, $\hat{\boldsymbol{A}} \in \mathbb{F}_q^{N \times N}$, $\boldsymbol{y} \in \mathbb{F}_q^{N \times m}$, the matrix $\boldsymbol{Y}$ is in left-RRE form as long as $\hat{\boldsymbol{A}}$ is in RRE form. Compared with the RRE form, it puts no constraints on the right part even when $\hat{\boldsymbol{A}}$ is not full-rank. Obviously the left-RRE form of a matrix may be not unique.

Now we prove that it is enough to use left-RRE forms instead of RRE forms in decoding KK codes. Given $\boldsymbol{Y} = [\hat{\boldsymbol{A}} \mid \boldsymbol{y}]$ and $\mathrm{RRE}(\hat{\boldsymbol{A}}) = \boldsymbol{R}\hat{\boldsymbol{A}}$, the product $\boldsymbol{RY}$ is in left-RRE form. It follows $\bar{\boldsymbol{Y}}' = \left[\begin{smallmatrix} \boldsymbol{I}_{\mathcal{U}^c} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_\delta \end{smallmatrix}\right]\boldsymbol{RY} = \left[\begin{smallmatrix} \boldsymbol{I}+\hat{\boldsymbol{L}}\boldsymbol{I}_{\mathcal{U}}^T & \boldsymbol{r}' \\ 0 & \hat{\boldsymbol{E}}' \end{smallmatrix}\right]$ has the same row space as $\boldsymbol{Y}$. With a similar approach as in [2, Appendix C], we can prove $\mathrm{rank}\left[\begin{smallmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{smallmatrix}\right] = \mathrm{rank}\left[\begin{smallmatrix} \hat{\boldsymbol{L}} & \boldsymbol{r}'-\boldsymbol{x} \\ 0 & \hat{\boldsymbol{E}}' \end{smallmatrix}\right] + n - \mu$. Hence the subspace distance is given by $d_S(\langle \boldsymbol{X} \rangle, \langle \boldsymbol{Y} \rangle) = 2\,\mathrm{rank}\left[\begin{smallmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{smallmatrix}\right] - \mathrm{rank}\,\boldsymbol{X} - \mathrm{rank}\,\boldsymbol{Y} = 2\,\mathrm{rank}\left[\begin{smallmatrix} \hat{\boldsymbol{L}} & \boldsymbol{r}'-\boldsymbol{x} \\ \boldsymbol{0} & \hat{\boldsymbol{E}}' \end{smallmatrix}\right] - \mu - \delta$. Thus the subspace decoding problem is equivalent to the generalized Gabidulin decoding problem with $(\boldsymbol{r}', \hat{\boldsymbol{L}}, \hat{\boldsymbol{E}}')$.

By using the left-RRE forms instead of RRE forms, we reduce the complexity of reduction slightly. More important, the reduction for left-RRE forms is completely determined by the left part of $\boldsymbol{Y}$, which greatly simplifies hardware implementation.

4.2. KK Codes Lifted from Cartesian Gabidulin Codes

The left-RRE form also considerably simplifies the decoding of KK codes that are lifted from Cartesian Gabidulin codes.

In network practice, the packet length is very long and m is much larger than n. In such cases, the decoding complexity of KK codes is prohibitive due to the huge field size of $\mathbb{F}_{2^m}$. A low-complexity approach in [2] suggested that instead of using a single long Gabidulin code, the Cartesian product of many short Gabidulin codes with the same distance can be used to construct KK codes for long packets.

Since the reduction in our left-RRE approach is purely determined by $\hat{\boldsymbol{A}}$, decoding $[\hat{\boldsymbol{A}} \mid \boldsymbol{y}_0 \mid \boldsymbol{y}_1 \mid \cdots \mid \boldsymbol{y}_{l-1}]$ can be divided into small decoding problems whose inputs are $[\hat{\boldsymbol{A}} \mid \boldsymbol{y}_0], [\hat{\boldsymbol{A}} \mid \boldsymbol{y}_1], \ldots, [\hat{\boldsymbol{A}} \mid \boldsymbol{y}_{l-1}]$. In this way, a small decoder for $\mathbb{F}_{q^n}$ can decode packets with lengths as $(l+1)n$.

For KK codes that are lifted from Cartesian Gabidulin codes, we can perform decoding in a serial manner with only one decoder, or in a semiparallel way with more decoders, or even in a fully parallel fashion. It is a tradeoff between cost/area/power and throughput.

4.3. Gaussian Elimination

We first show that finding the root space and minimum linearized polynomials can be done by Gaussian elimination.

According to [2], the complexity between the probabilistic algorithm in [7] and Berlekamp's deterministic method [6] is small for $q = 2$. So the deterministic method is preferred since it is much easier to implement.

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 $\boldsymbol{V}$ and finds linearly independent roots $\boldsymbol{z}$ such that $\boldsymbol{Vz} = \boldsymbol{0}$. Using the representation based on $(\alpha_0, \alpha_1, \ldots, \alpha_{m-1})$, the roots $\boldsymbol{z}$ are also the roots of the given polynomial. Finding $\boldsymbol{z}$ is to obtain the linear dependent combinations of the columns of $\boldsymbol{V}$, which can be done by Gaussian elimination.

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

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

Note that we assume that the coefficient of the highest degree term is one. Thus it can be solved by Gaussian elimination.

Furthermore, Gabidulin's algorithm is essentially a smart way of Gaussian elimination, which takes advantage of the properties of the matrix. So Gaussian elimination appears in most steps of the decoding process, including reduction for the RRE form, finding minimum linearized polynomials, finding the root space, and Gabidulin's algorithm. The reduction and finding the root space are Gaussian eliminations on matrices over $\mathbb{F}_q$, while linearized interpolation and Gabidulin's algorithm operate on matrices over $\mathbb{F}_{q^m}$.

For high-throughput implementations, we adopt the pivoting architecture from [12]. It 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. To apply it to singular matrices, which appear in the reduction for the RRE form and finding the root space, we adapt the architecture to detect singularity. Our architecture is also flexible about matrix sizes, which are determined by the varying numbers of errors, erasures, and deviations. Eliminations over $\mathbb{F}_{q^m}$ require divisions. By cross-multiplications, we can avoid divisions in Gaussian elimination; Divisions are used only when the row is reduced to have only one non-zero element. In Gabidulin's algorithm, the matrix is first reduced to a triangular form. 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.

4.4. Latency Analysis

We analyze the worst-case decoding latencies of our decoder architectures, in terms of clock cycles, in Table 1.

Table 1. Worst-Case Decoding Latency

	Gabidulin	KK
RRE	-	$n(n+1)/2+n$
Syndrome	n	n
$\lambda_U(x)$	-	$2(d-1)+m\mu$
$\sigma_D(x)$	-	$2(d-1)+m\delta$
$S_{DU}(x)$	-	$2(d-1)$
BMA	$2t$	$2t$
$S_{FD}(x)$	-	$d-1$
$\boldsymbol{\beta}$	-	$2(d-1)+m\epsilon$
$\sigma_U(x)$	-	$2(d-1)+m\epsilon$
$\sigma(x)$	-	$2(d-1)$
root space	$m(m+1)/2$	$m(m+1)/2$
error locator	$2t+mt$	$2\tau+m\tau$
error word	t	τ
Total ($n=m$)	$n(n+3)/2+(m+5)t$	$n(n+3)+(4m+30)t$

As in [12], the latency of Gaussian elimination for the left-RRE form is at most $n(n+1)/2$ cycles. Additionally it takes at most n cycles more to extract $\hat{\boldsymbol{L}}$ out of the left-RRE form. Similarly, the latency of finding the root space is at most $m(m+1)/2$. For minimum linearized polynomials, Gaussian elimination always works on non-singular matrices of size at most $d-1$. Hence it needs at most $d-1$ cycles for elimination. For each coefficient, it takes m cycles to perform a division. The backward pivoting scheme also needs $d-1$ cycles. Hence it needs $2(d-1)+m\mu$, $2(d-1)+m\delta$, and $2(d-1)+m\epsilon$ for $\lambda_U(x)$, $\sigma_D(x)$, and $\sigma_U(x)$, respectively. The latency of Gabidulin's algorithm can be computed similarly. 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. For our $(8,4)$ codes, the longest latency of them is no more than $2(d-1)+m\mu+m\delta$. 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.

5. IMPLEMENTATION RESULTS

We implement our decoder architecture in Verilog for an $(8,4)$ Gabidulin code, which can correct up to two errors. We also implement our decoder architecture for the corresponding KK code, which can correct ϵ errors, μ erasures, and δ deviations as long as $2\epsilon + \mu + \delta < 5$. Our designs are synthesized using Cadence RTL Compiler 7.1 and MOSIS SCMOS TSMC $0.18\mu\mathrm{m}$ standard cell library [13]. The synthesis results are given in Table 2. The total area in Table 2 includes both cell area and estimated net area, and the total power in Table 2 includes both leakage and estimated dynamic power. All estimation are made by the synthesis tool. In our calculation of throughput, we consider all input bits. Each received vector of the $(8, 4)$ Gabidulin code has 64 bits and that of the $(8, 4)$ KK code has 128 bits. The gate count of our generalized decoder is close to that of the $(255, 239)$ Reed–Solomon decoder in [14], which is 115,500. Although their code lengths are quite different, both codes are the longest in each class of codes. 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 2 are affordable except for applications with very stringent area and power requirements.

Table 2. Synthesis results of decoders for $(8, 4)$ Gabidulin and KK codes over $\mathbb{F}_{2^8}$

		Gabidulin	KK
Gates		22014	96636
Area (mm^2)	Cell	0.551	2.421
	Net	0.231	1.110
	Total	0.782	3.531
CPD (ns)		5.125	5.144
Power (mW)	Leakage	0.001	0.005
	Dynamic	126.816	483.446
	Total	126.817	483.451
Latency (cycles)		70	212
Throughput ($\mathrm{Mbit/s}$)		178	117

For practical network applications, the packet size is large. For example, for a packet size of 512 bytes, we can use a KK code that is based on Cartesian product of 511 length-8 Gabidulin codes. For higher throughput, more decoders can be used to decode in parallel. We list the gate counts and throughput of serial and factor-7 parallel schemes in Table 3.

Table 3. Performance of KK decoders for 512-byte packets

	Serial	7-Parallel
Gates	96636	676452
Area (mm^2)	3.531	24.717
Power (mW)	483.451	3384.157
Latency (cycles)	108332	15476
Throughput ($\mathrm{Mbit/s}$)	59	412

Although the area and power shown in Tables 2 and 3 are affordable, they are for short Gabidulin and KK codes over a small field. The $(8, 4)$ Gabidulin and KK codes can correct at most **two** errors. Although the $(8, 4)$ KK decoder can be used for long packets by Cartesian product, the Cartesian product of $(8, 4)$ Gabidulin codes and its corresponding CDC also can correct at most **two** errors. When we increase the error correction capabilities of both Gabidulin and KK codes, longer codes are needed and thus larger fields are required. The large field size implies a higher complexity for finite field arithmetic. 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.

6. REFERENCES

[1] 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.

[2] 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.

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

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

[5] 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.

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

[7] 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.

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

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

[10] 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.

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

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

[13] J. E. Stine, J. Grad, I. Castellanos, J. Blank, V. Dave, M. Prakash, N. Iliev, and N. Jachimiec, "A framework for high-level synthesis of system-on-chip designs," in

Proc. IEEE Int. Conf. Microelectronic Systems Education (MSE'05), Anaheim, CA, Jun. 12–13, 2005, pp. 11–12.

[14] H. Lee, "High-speed VLSI architecture for parallel Reed–Solomon decoder," *IEEE Trans. VLSI Syst.*, vol. 11, no. 2, pp. 288–294, Apr. 2003.