Low Complexity Opportunistic Decoder for Network Coding

Bei Yin, Michael Wu, Guohui Wang, and Joseph R. Cavallaro
ECE Department, Rice University, 6100 Main St., Houston, TX 77005
Email: {by2, mbw2, wgh, cavallar}@rice.edu

***Abstract*—In this paper, we propose a novel opportunistic decoding scheme for network coding decoder which significantly reduces the decoder complexity and increases the throughput. Network coding was proposed to improve the network throughput and reliability, especially for multicast transmissions. Although network coding increases the network performance, the complexity of the network coding decoder algorithm is still high, especially for higher dimensional finite fields or larger network codes. Different software and hardware approaches were proposed to accelerate the decoding algorithm, but the decoder remains to be the bottleneck for high speed data transmission. We propose a novel decoding scheme which exploits the structure of the network coding matrix to reduce the network decoder complexity and improve throughput. We also implemented the proposed scheme on Virtex 7 FPGA and compared our implementation to the widely used Gaussian elimination.**

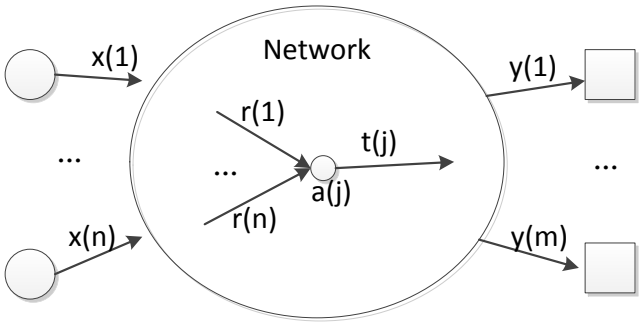

Fig. 1. Network coding system model

I. Introduction

Network coding was first proposed in [1] to increase the efficiency of multicast transmissions in a network by alleviating traffic at the shared links. In contrast to routing and packet forwarding in a traditional network, intermediate nodes with network coding encode the incoming packets before forwarding them toward the destination. The authors in [2] showed that linear network coding can achieve the max-flow bound from the source to each destination node. To simplify network code design, authors in [3] introduced random linear network coding. The first practical protocol of network coding was described in [4]. We adopt the framework described in [4], where the encoded packets and the corresponding network coding coefficients are transmitted within the same packet. In this paper, we attempt to reduce the complexity of the network decoder, which is required to recover the original messages at the destination.

Compared with nodes in a traditional network, nodes that employ network coding require additional computations to encode and decode the packets. The decoding algorithm at the destination is particularly intensive. A decoder at the destination needs to solve systems of linear equations over a finite field to recover the original information. This limits the decoding throughput. To address the bottleneck, a number of publications have discussed different implementations in software and hardware. In [5], Gaussian elimination is used to solve a system of linear equations on GPU. To reduce the latency further, in [6], the authors adopted the Gauss-Jordan elimination on GPU. In [7], [8], matrix inversions are performed on the CPU before solving the linear equations on GPU. Although a GPU provides massive computational power, hardware approaches can outperform these software solutions. In [9], a hardware network coding decoder on FPGA was proposed in which Cramer's rule was used for solving the linear system of equations. In [10], a network decoder was implemented on FPGA by using the Gaussian elimination method. Although these methods increase the throughput and alleviate traffic bottlenecks, they all inherently have complexity of $\mathcal{O}(n^3)$, and they perform a new matrix inversion for every new set of network coding coefficients. As a result, the complexity of these schemes increases rapidly as the size of the network coding coefficient matrix becomes larger and the dimension of the finite field becomes higher. This will also limit the throughput and the usage of these designs for high data rate transmission.

In order to reduce the decoding complexity and increasing the throughput of the network coding decoder, we propose a new decoding scheme based on Sherman–Morrison formula which was summarized in [11]. Instead of performing a new matrix inversion for every new set of network coding coefficients with Gaussian elimination or Cramer's rule, we compute a new inverse by updating the previous inversion result. Because only the updated elements in the new matrix affect the new inversion result, we can at most reduce the decoding complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

In section II, an overview of the network coding system model is introduced. Section III presents our low complexity opportunistic decoding scheme. Implementation and complexity analysis is given in Section IV. Section V draws the conclusions.

II. Network coding system model

Consider a communication network with n source nodes, m destination nodes, and a random network of intermediate nodes between the source and destination nodes. As shown in Fig. 1, the n source nodes want to transmit packets to m destination nodes. The i-th source node first constructs a vector $\mathbf{x}(i)$ of length l, where each element of $\mathbf{x}(i)$ is in the finite field $\mathrm{GF}(2^b)$. The packets from the source nodes are then sent to the destination nodes through the network.

As the packets propagate through the network, the intermediate nodes will not simply forward the incoming packets. To improve throughput, the intermediate nodes in the network adopt random linear network coding [3]. In this case, the intermediate node creates a new packet by multiplying n incoming packets with a random network coding vector of coefficients, $\mathbf{a}(j)$ of length n, where each element of $\mathbf{a}(j)$ is randomly generated from the finite field $\mathrm{GF}(2^b)$. The resulting encoded packet at immediate node j, $\mathbf{t}(j)$, is a length l vector which can be expressed as:

$$\mathbf{t}(j) = \mathbf{a}(j) \begin{bmatrix} \mathbf{r}(1) \\ \vdots \\ \mathbf{r}(n) \end{bmatrix}.$$

When forwarding the encoded packet, the coding coefficients $\mathbf{a}(j)$ are transmitted along with encoded data $\mathbf{t}(j)$.

At subsequent intermediate nodes, the coding coefficients and encoded packet are both updated by the new network coding coefficients. This will be explained in details in Section III.

When the packets reach the destination node, the i-th received encoded packet, $\mathbf{y}(i)$, can be expressed as a linear combination of information packets $\mathbf{x}(1), \ldots \mathbf{x}(n)$,

$$\mathbf{y}(i) = \mathbf{g}(i) \begin{bmatrix} \mathbf{x}(1) \\ \vdots \\ \mathbf{x}(n) \end{bmatrix},$$

where $\mathbf{y}(i)$ is a length l vector, and $\mathbf{g}(i)$ is a length n vector of the corresponding network coding coefficients. The $\mathbf{g}(i)$ is a linear combination of random generated coefficients $\mathbf{a}(j)$ along the propagation path. To recover the packets $\mathbf{x}(1), \ldots, \mathbf{x}(n)$ sent by the source nodes, a destination node needs to receive n independent packets. All the received packets are put in an $n \times l$ matrix $\mathbf{Y}$, and all the received network coding coefficients are put in an $n \times n$ $\mathbf{G}$ matrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}(1) \\ \vdots \\ \mathbf{g}(n) \end{bmatrix},$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}(1) \\ \vdots \\ \mathbf{y}(n) \end{bmatrix}.$$

Then the decoder at the destination node can multiply the received packets $\mathbf{Y}$ by the inverse of $\mathbf{G}$ to decode the original packets $\mathbf{x}(1), \ldots \mathbf{x}(n)$,

$$\begin{bmatrix} \mathbf{x}(1) \\ \vdots \\ \mathbf{x}(n) \end{bmatrix} = \mathbf{G}^{-1}\mathbf{Y}. \quad (1)$$

The above decoding will be performed at each destination node to recover the original packets. As the network coding coefficients are randomly generated in the network, the coefficient matrix $\mathbf{G}$ may not be invertible. To alleviate this problem, higher dimensional finite fields and larger network coding coefficients matrices are suggested in the literature [4]. However, these two methods increase complexity of the corresponding network coding decoder.

III. Low complexity opportunistic decoder

To recover the original packets, each destination node needs to solve systems of linear equations as shown in (1) in a finite field. The simplest method is to always use Gaussian elimination to invert $\mathbf{G}$ for each new set of packets. However, the complexity of this method is $\mathcal{O}(n^3)$. For larger network coding coefficient matrices and higher dimensional finite fields, this method results in high hardware complexity and low throughput. To solve this problem, we propose a method which does not always invert $\mathbf{G}$ from scratch. We observe that the final network coding coefficients at the destination nodes are related to the path that the packets traverse through the network. Particularly, the routes that the packets take from the source nodes to the destination nodes may not change significantly from transmission to transmission, especially for low mobility networks. As a result, network coding coefficients are not completely different from one transmission to the next. By exploiting this feature, we can reduce the complexity of the network coding decoder to $\mathcal{O}(n^2)$ and also increase the throughput.

A. Decoding algorithm

Suppose the network coding coefficient matrix from the first set of packets is $\mathbf{G}_1$ and the network coding coefficient matrix from the second set of packets is $\mathbf{G}_2$. In the network, the packets pass through a few stages and arrive at the destination. Assume the network coding coefficients of these stages are $\mathbf{a}(1), \ldots, \mathbf{a}(n)$, $\mathbf{b}(1), \ldots, \mathbf{b}(n)$, $\mathbf{c}(1), \ldots, \mathbf{c}(n)$, $\mathbf{d}(1), \ldots, \mathbf{d}(n)$, and $\mathbf{e}(1), \ldots, \mathbf{e}(n)$. Thus, the network coding coefficient matrix $\mathbf{G}_1$ at the destination is represented as

$$\mathbf{G}_1 = \begin{bmatrix} \mathbf{g}_1(1) \\ \vdots \\ \mathbf{g}_1(n) \end{bmatrix} = \begin{bmatrix} \mathbf{e}(1) \\ \vdots \\ \mathbf{e}(n) \end{bmatrix} \begin{bmatrix} \mathbf{d}(1) \\ \vdots \\ \mathbf{d}(n) \end{bmatrix} \begin{bmatrix} \mathbf{c}(1) \\ \vdots \\ \mathbf{c}(n) \end{bmatrix} \begin{bmatrix} \mathbf{b}(1) \\ \vdots \\ \mathbf{b}(n) \end{bmatrix} \begin{bmatrix} \mathbf{a}(1) \\ \vdots \\ \mathbf{a}(n) \end{bmatrix}$$

During the second set of packet transmissions, if the network coding coefficients in one node $\mathbf{c}(i)$ are changed to $\mathbf{c}_{new}(i)$, then the network coding coefficient matrix $\mathbf{G}_2$ is

$$\mathbf{G}_2 = \mathbf{G}_1 + \Delta = \begin{bmatrix}\mathbf{g}_1(1)\\ \vdots \\ \mathbf{g}_1(n)\end{bmatrix} + \begin{bmatrix}\mathbf{e}(1)\\ \vdots \\ \mathbf{e}(n)\end{bmatrix}\begin{bmatrix}\mathbf{d}(1)\\ \vdots \\ \mathbf{d}(n)\end{bmatrix}\begin{bmatrix}0\\ \mathbf{c}_{diff}(i)\\ 0\end{bmatrix}\begin{bmatrix}\mathbf{b}(1)\\ \vdots \\ \mathbf{b}(n)\end{bmatrix}\begin{bmatrix}\mathbf{a}(1)\\ \vdots \\ \mathbf{a}(n)\end{bmatrix},$$

which can be simplified to:

$$\begin{aligned}\mathbf{G}_2 &= \mathbf{G}_1 + \mathbf{ed}_{col}(i)\mathbf{c}_{diff}(i)\mathbf{BA} \\ &= \mathbf{G}_1 + \mathbf{uv}.\end{aligned}$$

where $\mathbf{ed}_{col}(i)$ is the i-th column of the matrix product of $\mathbf{ED}$, $\mathbf{u}$ is a column vector and equals to $\mathbf{ed}_{col}(i)$, $\mathbf{v}$ is a row vector and equals to $\mathbf{c}_{diff}(i)\mathbf{BA}$, and $\mathbf{c}_{diff}(i) = \mathbf{c}_{new}(i) - \mathbf{c}(i)$.

This equation shows that if the coefficients of a stage change, the change to the network coding coefficient matrix $\mathbf{G}$ is a matrix and equals to $\mathbf{uv}$. Based on this observation, we apply Sherman-Morrison formula to obtain the matrix inverse [11] instead of inverting $\mathbf{G}$ for every transmission. When a destination node first receives the matrix $\mathbf{G}_1$, the node will perform a full matrix inversion, $\mathbf{G}_1^{-1}$, to solve the system of linear equations. For subsequent sets of packets arriving at the destination, the decoder may not need to compute $\mathbf{G}_2^{-1}$ by performing the full inversion. For example, if the difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ is one row, one column, or can be decomposed into $\mathbf{uv}$, we can compute $\mathbf{G}_2^{-1}$ by updating $\mathbf{G}_1^{-1}$:

$$\begin{aligned}\mathbf{G}_2^{-1} &= (\mathbf{G}1 + \mathbf{uv})^{-1} \\ &= \mathbf{G}_1^{-1} + \frac{\mathbf{G}_1^{-1}\mathbf{uv}\mathbf{G}_1^{-1}}{1 - \mathbf{v}\mathbf{G}_1^{-1}\mathbf{u}},\end{aligned} \quad (2)$$

where $\mathbf{u}$ is a column vector and $\mathbf{v}$ is a row vector. For example, if the difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ is one row, $\mathbf{u}$ will be a unit column vector with an one at the corresponding row and zeros at all other positions, and $\mathbf{v}$ will be a row vector which is the difference. If the difference between $\mathbf{G}_1$ and $\mathbf{G}_2$ is one column, then $\mathbf{u}$ will be a column vector which is the difference, and $\mathbf{v}$ will be a unit row vector with an one at the corresponding column and zeros at all other positions. The term $1 - \mathbf{v}\mathbf{G}_1^{-1}\mathbf{u}$ in the equation is a scalar value. Compared to the full matrix inversion case, only one inversion is needed to compute $(1 - \mathbf{v}\mathbf{G}1^{-1}\mathbf{u})^{-1}$ in our proposed algorithm.

If more rows or columns in the network coding coefficient matrix are changed, the above algorithm can be applied iteratively. The difference can be decomposed into a series of $\mathbf{u}$ vectors and $\mathbf{v}$ vectors, with

$$\mathbf{G}_2^{-1} = (\mathbf{G}_1 + \mathbf{u}_1\mathbf{v}_1 + \mathbf{u}_2\mathbf{v}_2 + ... + \mathbf{u}_n\mathbf{v}_n)^{-1}$$

where $\mathbf{u}_i$ can be a unit column vector with i-th element equivalent to one, and $\mathbf{v}_i$ is row vector of i-th row difference between $\mathbf{G}_1$ and $\mathbf{G}_2$, Alternatively, $\mathbf{v}_i$ can be a unit row vector with i-th element equivalent to one, and $\mathbf{u}_i$ is column vector of i-th column difference between $\mathbf{G}_1$ and $\mathbf{G}_2$.

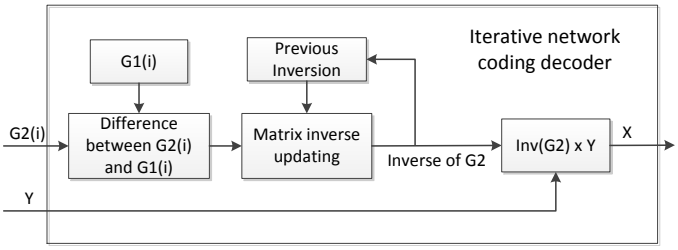

Fig. 2. Block diagram of proposed iterative decoder.

In fact, we can compute the full inversion of matrix $\mathbf{G}_2$ by performing the updating iteratively,

$$\mathbf{G}_{2_1}^{-1} = (\mathbf{G}_1 + \mathbf{u}_1\mathbf{v}_1)^{-1}$$

then:

$$\mathbf{G}_2^{-1} = (\mathbf{G}_{2_{n-1}} + \mathbf{u}_n\mathbf{v}_n)^{-1}$$

Computing the full inverse as a series of updates will take n iterations.

B. Implementation of low complexity opportunistic decoder

As the algorithm computes the inverse iteratively from row to row, the matrix inverse updating does not need to wait until the matrix $\mathbf{G}_2$ is completely received. To reduce the decoding latency, the updating process can begin as soon as a new row $\mathbf{G}_2(i)$ is received. If the received row $\mathbf{G}_2(i)$ is same as the previous stored row $\mathbf{G}_1(i)$, no update needs to be performed for the current row. If the received row $\mathbf{G}_2(i)$ is different from the previous stored row $\mathbf{G}_1(i)$, we can assume subsequent rows of $\mathbf{G}_2$ and $\mathbf{G}_1$ are identical, and then apply (2) to update the inverse. As the matrix inverse updating algorithm is performed in a finite field, this algorithm will not accumulate error on each iteration. When we receive all the rows of $\mathbf{G}_2$, we have compute the inverse of $\mathbf{G}_2$. We can recover the original $\mathbf{X}$ by multiplying $\mathbf{G}_2^{-1}$ with $\mathbf{Y}$.

The architecture of the proposed network decoder using this scheme is shown in Fig. 2. The decoder consists of a difference search block, a matrix inversion block, a matrix multiplication block, and buffers to store the previous network coding coefficients and the corresponding inverses.

After receiving a new row of the network coding coefficient matrix $\mathbf{G}_2(i)$, the difference block computes $\mathbf{G}_2(i) - \mathbf{G}_1(i)$ in a finite field. The subtraction is implemented as parallel XOR operations. In total, n XOR modules are used, and each XOR has b bits.

To compute the new matrix inverse $\mathbf{G}_2^{-1}$, the term $\mathbf{v}\mathbf{G}_1^{-1}$ is first computed from $\mathbf{G}_1^{-1}$ which is computed from the previous set of packets. Finite field multiplications and additions are used. The finite field addition is b-bit XOR operations. There are a few ways to implement the finite field multiplication. We use the polynomial based two step classic multiplication

TABLE I
COMPLEXITY COMPARISON

Design	Finite Field	Coefficient Matrix Size	Registers	LUTs	Frequency	Throughput
Gaussian Elimination (Xilinx XC4VLX60)[10]	256	4×4	1,675	19,583	50.7 MHz	0.8 Gbps
Proposed Matrix Inversion (Xilinx XC7VX330T)	256	4×4	841	2,603	365 MHz	11.68 Gbps
	256	8×8	2,601	10,403	365 MHz	23.36 Gbps
	65,536	4×4	1,644	7,432	200 MHz	12.8 Gbps

instead of the lookup table based multiplication[12]. This is because the lookup table is associated with a certain finite field, if a different finite field is used, the lookup table has to be completely redesigned. Therefore, polynomial based multiplication is more flexible. The design can switch between different finite fields. Because each number in a finite field can be represented as a polynomial, the multiplication of two finite field numbers corresponds to multiplication of two polynomials. This consists of shifters and XORs. After this, the product is moduloed the irreducible polynomial of the corresponding finite field. The irreducible polynomial is stored in the memory and the modulo operation is equivalent to a linear mapping from the product back to a b-bit polynomial. The linear mapping is computed on the fly from the irreducible polynomial.

The division in $(1 - \mathbf{v}\mathbf{G}_1^{-1}\mathbf{u})^{-1}$ is computed with a finite field inverse module. Because the term $(1 - \mathbf{v}\mathbf{G}_1^{-1}\mathbf{u})^{-1}$ is a scalar value, only one inverse module is needed. The Extended Euclidean algorithm is used find the inverse. This algorithm not only computes the *greatest common divisor (gcd)* polynomial of two polynomials $gcd(a(x), b(x))$, but also finds two polynomials, $u(x)$ and $v(x)$, which satisfy $gcd(a(x), b(x)) = a(x)u(x) + b(x)v(x)$. Since the *gcd* of the irreducible polynomial $f(x)$ and the other polynomial in the field $a(x)$ is 1, $a(x)u(x) + f(x)v(x) = 1$. This means that $mod(a(x)u(x), f(x)) = 1$, and $a(x)^{-1} = mod(u(x), f(x))$. By using this method, $1-\mathbf{v}\mathbf{G}_1^{-1}\mathbf{u}$ can be computed instead of using lookup table. Then $(1-\mathbf{v}\mathbf{G}_1^{-1}\mathbf{u})^{-1}$ is multiplied with $\mathbf{u}$. Since $\mathbf{u}$ is a unit vector, multiplication is actually not needed.

This above schedule first computes $\mathbf{G}_1^{-1}\mathbf{u}/(1 - \mathbf{v}\mathbf{G}_1^{-1}\mathbf{u})$ and $\mathbf{v}\mathbf{G}_1^{-1}$. These two terms are multiplied together. Compared to computing $\mathbf{u}\mathbf{v}$ followed by $\mathbf{G}_1^{-1}\mathbf{u}\mathbf{v}\mathbf{G}_1^{-1}$, this reduces the number of operations. This schedule uses one vector-vector multiplication and two vector-matrix multiplications, while the latter needs one vector-vector multiplication and two matrix-matrix multiplications.

IV. IMPLEMENTATION RESULT AND COMPLEXITY ANALYSIS

The complexity of the above scheme depends on how many elements are changing in the network coding coefficient matrix. In general, the complexity is $\mathcal{O}(n^2)$.

If only one element of the network coding coefficient matrix is changed, the above algorithm needs n^2 multiplications, where n is the number of rows of $\mathbf{G}$. If only one row or one column of the network coding coefficient matrix is changed, the above algorithm requires $2n^2$ multiplications. To generalize, if m rows or columns of the network coding coefficient matrix were changed, the above algorithm can run iteratively from row to row or column to column, which needs $2mn^2$ multiplications. For Gaussian elimination, a total of $5/6 \cdot n^3$ number of multiplications are needed for the inversion. As a result, our scheme can reduce the complexity of the destination node if the number of changed rows or columns m is less than or equal to $5/12 \cdot n$. The proposed decoder is an opportunistic decoder. Because in the best case, when no element in the matrix is changed, no updating needs to be performed. In the worst case, the complexity is around $2n^3$.

Our design is implemented on a Xilinx Virtex 7 FPGA. The results are shown in Table I. The implementation results are compared with [10], which uses Gaussian elimination. With the same finite field size of 2^8 and the same 4×4 network coding coefficient matrix, our design has only 14% complexity and is around 14 times faster than the one which uses Gaussian elimination [10]. The frequency is more than 7 times faster. With an 8×8 network coding coefficient matrix size, the throughput of our design is doubled compared to the 4×4 case, because the received data $\mathbf{Y}$ is doubled. With higher finite field of 2^{16}, the throughput of our design is not doubled but slightly higher than the 4×4 case. This is because although the number of bits per symbol is doubled in 2^{16} compared to 2^8, the maximum attainable frequency is lower. As the field becomes higher, the computational time and complexity of finite field multiplication and inverse are also increased.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose a low complexity opportunistic decoder for network coding. In contrast to conventional schemes, our scheme computes the inverse of the current network coding coefficient matrix from the previously computed inverse by exploiting the matrix structure. By implementing the algorithm on FPGA, we show that our scheme can significantly reduce the complexity compared to Gaussian elimination and increases the decoding throughput to above 11.68 Gbps. With a higher field of 2^{16} , our design can achieve 12.8 Gbps. This indicates that our scheme can reduce the decoding bottleneck and is suitable for high speed data transmission.

ACKNOWLEDGMENTS

This work was supported in part by Renesas Mobile and by the US National Science Foundation under grants ECCS-

1232274, EECS-0925942 and CNS-0923479.

REFERENCES

[1] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, "Network Information Flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[2] S. R. Li, R. W. Yeung, and N. Cai, "Linear Network Coding," *IEEE Transactions on Information Theory*, vol. 49, pp. 371–381, 2003.

[3] T. Ho, M. Medard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A Random Linear Network Coding Approach to Multicast," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, 2006.

[4] P. A. Chou, Y. Wu, and K. Jain, "Practical Network Coding," *http://research.microsoft.com*, 2003.

[5] P. Vingelmann, P. Zanaty, F. Fitzek, and H. Charaf, "Implementation of Random Linear Network Coding on OpenGL-enabled Graphics Cards," in *European Wireless*, Aalborg, Denmark, May 2009.

[6] H. Shojania, B. Li, and X. Wang, "Nuclei: GPU-accelerated many-core network coding," in *Proceedings of IEEE INFOCOM*, 2009, pp. 459–467.

[7] X. Chu, K. Zhao, and M. Wang, "Massively Parallel Network Coding on GPUs," in *IEEE International Performance, Computing and Communications Conference (IPCCC)*, Dec. 2008, pp. 144–151.

[8] L. Huang, R. Wang, Y. Huang, G. Wang, and X. Zhang, "An Improved Parallelized Random Linear Network Coding Algorithm on GPU," *International Conference on Networking and Distributed Computing*, pp. 79–82, 2011.

[9] M. Zhang, H. Li, F. Chen, H. Hou, H. An, W. Wang, and J. Huang, "A General Co/Decoder of Network Coding in HDL," in *International Symposium on Network Coding (NetCod)*, July 2011, pp. 1–5.

[10] T. Yoon and J. Park, "FPGA Implementation of Network Coding Decoder," *IJCSNS International Journal of Computer Science and Network Security*, vol. 10, p. 12, Dec. 2010.

[11] W. W. Hager, "Updating the Inverse of a Matrix," *SIAM Review*, vol. 31, no. 2, pp. 221–239, 1989.

[12] J. P. Deschamps, J. L. Imana, and G. D. Sutter, *Hardware Implementation of Finite-Field Arithmetic*. McGraw Hill, Mar. 2009.