## K. Prasad and B. Sundar Rajan, Dept. of ECE, IISc, Bangalore 560012, India Email: {prasadk5,bsrajan}@ece.iisc.ernet.in *Abstract*—Existing construction algorithms of block networkerror correcting codes require a rather large field size, which grows with the size of the network and the number of sinks, and thereby can be prohibitive in large networks. In this work, we give an algorithm which starting from a given network-

grows with the size of the network and the number of sinks, and thereby can be prohibitive in large networks. In this work, we give an algorithm which, starting from a given networkerror correcting code, can obtain another network code using a small field, with the same error correcting capability as the original code. An algorithm for designing network codes using small field sizes proposed recently by Ebrahimi and Fragouli can be seen as a special case of our algorithm. The major step in our algorithm is to find a least degree irreducible polynomial which is coprime to another large degree polynomial. We utilize the algebraic properties of finite fields to implement this step so that it becomes much faster than the brute-force method. As a result the algorithm given by Ebrahimi and Fragouli is also quickened.

#### I. INTRODUCTION

Network coding was introduced in [1] as a means to improve the rate of transmission in networks. Linear network coding was introduced in [2]. Deterministic algorithms exist [3]–[5] to construct scalar network codes (in which the input symbols and the network coding coefficients are scalars from a finite field) which achieve the maxflow-mincut capacity in the case of acyclic networks with a single source which wishes to multicast a set of finite field symbols to a set of N sinks, as long as the field size q > N. Finding the minimum field size over which a network code exists for a given network is known to be NP hard [6]. An algorithm was proposed in [7] which attempts to find network codes using small field sizes, given a network coding solution for the network over some larger field size q > N. The algorithms of [7] also apply to linear deterministic networks [8], and for vector network codes (where the source seeks to multicast a set of vectors, rather than just finite field symbols). In this work, we are explicitly concerned about the scalar network coding problem, although the same techniques can be easily extended to accommodate for vector network coding and linear deterministic networks, if permissible, as in the case of [7].

Network-error correction, which involved a trade-off between the rate of transmission and the number of correctable network-edge errors, was introduced in [9] as an extension of classical error correction to a network setting. Along with subsequent works [10] and [11], this generalized the classical notions of the Hamming weight, Hamming distance, minimum distance and various classical error control coding bounds to their network counterparts. Algorithms for constructing network-error correcting codes which meet a generalization of the classical Singleton bound for networks can be found in [10]–[13]. Using the algorithm of [12], a network code which can correct any errors occurring in at most  $\alpha$  edges can be constructed, as long as the field size q is such that

Network-Error Correcting Codes using Small Fields

$$q > N \left( \begin{array}{c} |\mathcal{E}| \\ 2\alpha \end{array} \right)$$

where  $\mathcal{E}$  is the set of edges in the network. The algorithms of [10], [11] have similar requirements to construct such networkerror correcting codes. This can be prohibitive when  $|\mathcal{E}|$  is large, as the sink nodes and the coding nodes of the network have to perform operations over this large field, possibly increasing the overall delay in communication. In [13], the bound on the field size was further tightened. However, this bound in [13] too potentially grows with the size of the network.

In this work, we propose an algorithm for block networkerror correction using small fields. We shall restrict our algorithms and analysis to fields with binary characteristic. The techniques presented can be extended to finite fields of other characteristics without much difficultly. The contributions of this work are as follows.

- We propose an algorithm to construct network-error correcting codes using small fields, by first designing a network-error correcting code over a large field size using known techniques (for example, [12]) and then using algebraic techniques to obtain a network-error correcting code over a smaller field size. The network coding version of this algorithm reduces to the algorithm proposed by Ebrahimi and Fragouli in [7], which we shall refer to as the EF algorithm henceforth.
- The major step in our algorithm is to compute a polynomial of least degree coprime with a polynomial, f(X), of possibly large degree. While it is shown in [7] that this can be done in polynomial time, the complexity can still be large. Optimizing based on our requirement, we propose an alternate faster algorithm for computing the polynomial coprime with f(X). This reduces the complexity of the EF algorithm also, which simply adopts a brute force method to do the same.
- Illustrative examples are shown which indicate that parameters such as the initial network-error correcting code and the choice of representation of the initial large finite field influence the ability of our algorithm to obtain a network-error correcting code over a small field size.

The rest of this paper is organized as follows. In Section II, we give the basic notations and definitions related to network coding, required for our purpose. Also, we review the EF

Part of the content of this work was presented at ISIT 2011 held at St. Petersburg, Russia during July 31 - Aug. 5, 2011.

algorithm briefly in Section II. Section III presents our algorithm for constructing network-error correcting codes using small field sizes, along with calculations of the complexity of the algorithm. In Section III, we also propose a fast way to compute the major step of our algorithm, which is to obtain a least degree polynomial coprime with another polynomial of larger degree. We also show that this fast technique reduces the running time of the EF algorithm. Examples illustrating our algorithm for network coding and error correction are presented in Section IV. Finally, we conclude the paper in Section V with comments and directions for further research.

#### **II. PRELIMINARIES AND BACKGROUND**

The model for acyclic networks considered in this paper is as in [14]. An acyclic network can be represented as an acyclic directed multi-graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , where  $\mathcal{V}$  is the set of all nodes and  $\mathcal{E}$  is the set of all edges in the network. We assume that every edge in  $\mathcal{G}$  can carry at most one symbol from a finite field  $\mathbb{F}_q$ . Network links with capacities greater than unity are modeled as parallel edges. The network is assumed to be instantaneous, i.e., all nodes process the same generation (the set of symbols generated at the source at a particular time instant) of input symbols to the network in a given coding order (ancestral order [14]). For an edge e, let tail(e) and head(e) denote the start node and the end node of e. An ancestral ordering can be assumed on  $\mathcal{E}$  as the network is acyclic. Let  $s \in \mathcal{V}$  be the source node and  $\mathcal{T}$  be the set of  $N(=|\mathcal{T}|)$  receivers. Let  $h_{\tau}$  be the unicast capacity for a sink node  $T \in \mathcal{T}$ , i.e., the maximum number of edge-disjoint paths from s to T. Then  $h = \min_{T \in \mathcal{T}} h_T$  is the max-flow min-cut capacity of the multicast connection.

A h'-dimensional network code  $(h' \leq h)$  is one which can be used to transmit h' symbols simultaneously from sto all sinks  $T \in \mathcal{T}$ , and can be described [3] by the following matrices, each having elements from the finite field  $\mathbb{F}_q$ .

• A matrix A (of size  $h' \times |\mathcal{E}|$ ), which describes the way the source maps symbols onto the network. The entries of A are defined as

$$A_{i,j} = \begin{cases} \alpha_{i,e_j} & \text{if } s = tail(e_j) \\ 0 & \text{otherwise,} \end{cases}$$

where  $\alpha_{i,e_j} \in \mathbb{F}_q$  is the network coding coefficient at the source coupling input *i* with edge  $e_j$ .

• A matrix K (of size  $|\mathcal{E}| \times |\mathcal{E}|$ ), which describes how the symbols are processed between the edges of the network. The entries of K are defined as

$$K_{i,j} = \begin{cases} \beta_{i,j} & \text{if } head(e_i) = tail(e_j), \\ 0 & \text{otherwise}, \end{cases}$$

where  $\beta_{i,j} \in \mathbb{F}_q$  is the local encoding kernel coefficient between  $e_i$  and  $e_j$ .

•  $D_T$  (of size  $|\mathcal{E}| \times h'$  for every sink  $T \in \mathcal{T}$ ), which describes how the symbols received by the sink T are processed. The entries of the matrix  $D_T$  are defined as

$$D_{T_{i,j}} = \begin{cases} \epsilon_{e_j,i} & \text{if } head(e_j) = T_i \\ 0 & \text{otherwise,} \end{cases}$$

where  $\epsilon_{e_j,i} \in \mathbb{F}_q$  describes the coupling between the symbols on  $e_j$  and the  $i^{th}$  input.

Let  $F = (I - K)^{-1}$ , where I is the identity matrix of size  $|\mathcal{E}|$ . Note that F is well defined as (I - K) is an invertible matrix, as K is strictly upper-triangular. We then have the following definition.

Definition 1: [3] The network transfer matrix,  $M_T$  for a h'dimensional network code, corresponding to a sink node  $T \in \mathcal{T}$  is a full rank  $h' \times h'$  matrix defined as  $M_T := AFD_T = AF_T$ , where  $F_T := FD_T$ .

The matrix  $M_T$  governs the input-output relationship at sink T. The problem of designing a h'-dimensional network code then implies making a choice for the matrices A, F, and  $D_T$ , such that the matrices  $\{M_T : T \in \mathcal{T}\}$  have rank h' each. We thus consider each element of A, F, and  $D_T$ to be a variable  $X_i$  for some positive integer i, which takes values from the finite field  $\mathbb{F}_q$ . Let  $\{X_i\}$  be the set of all variables, whose values define the network code. The variables  $X_i$ s are known as the *local encoding coefficients* [14]. For an edge e in a network with a h'-dimensional network code in place, the *global encoding vector* [14] is a h' dimensional vector which defines the particular linear combination of the h' input symbols which flow through e. It is known [3]–[5] that deterministic methods of constructing a h-dimensional network code exist, as long as q > N.

Let  $\Lambda$  be the length of the longest path from the source to any sink. Because of the structure of the matrices A, Fand  $D_T$ , it is seen [7] that the matrix  $M_T$  has degree at most  $\Lambda$  in any particular variable  $X_i$  and also a total degree (sum of the degrees across all variables in any monomial) of  $\Lambda$ . Let  $f_T(X_1, X_2, ...X_{|\{X_i\}|})$  be the determinant of  $M_T$  and  $f(X_1, X_2, ...X_{|\{X_i\}|}) = \prod_{T \in \mathcal{T}} f_T$ . Then the degree in any variable (and the total degree) of the polynomials  $f_T$  and fare at most  $h'\Lambda$  and  $Nh'\Lambda$  respectively.

A brief version of the EF algorithm is given in Algorithm 1. Note that the key step in Algorithm 1 is step (4), where

Algorithm 1: Scalar network coding algorithm using small fields - [7]

(1) Assign values  $\alpha_i$ s to the scalar coding coefficients  $X_i$ s from an appropriate field  $\mathbb{F}_{2^k}\left(2^k = 2^{\lceil \log(N) \rceil + 1} > N\right)$  such that the network transfer matrices  $M_T$ s to all the sinks are invertible. (2) Express every  $X_i = \alpha_i$  as a binary polynomial  $p_i(X)$ of degree at most k-1 using the usual polynomial representation of the finite field  $\mathbb{F}_{2^k}$ , for a particular choice of the primitive polynomial of degree k. (3) Substituting these polynomials representing the  $X_i$ s in the matrices  $M_T$ , calculate the determinants of  $M_T$  as the polynomials  $f_{\tau}(X) \in \mathbb{F}_2[X]$ , and also find  $f(X) = \prod_{T \in \mathcal{T}} f_{\mathcal{T}}(X)$ . Then, f(X) is non-zero and has degree at most  $N(k-1)h\Lambda$  in the variable X. (4) Find an irreducible polynomial of least degree, g(X), which is coprime with f(X). (5) Let  $X_i = p_i(X) \pmod{g(X)}$ . Thus, each  $X_i$  can be viewed as an element in  $\frac{\mathbb{F}_2[X]}{(g(X))}$ . Also, for each sink T, the matrices  $M_T$  remain invertible as  $f_{\tau}(X) \pmod{g(X)} \neq 0$ , as  $f(X) \pmod{g(X)} \neq 0$ .

an irreducible polynomial g(X) of least degree is to be found. It is shown in [7] that such a coprime g(X) exists and and can be computed with  $O(n^2 log(n))$  operations, where  $n = deg(f(X)) = Nh\Lambda \lceil log(N) \rceil$ .

# III. NETWORK-ERROR CORRECTING CODES USING SMALL FIELDS

This section presents the major contribution of this work. After briefly reviewing the network-error correcting code construction algorithm in [12], we proceed to give an algorithm which can obtain network-error correcting codes using small finite fields.

#### A. Network-Error Correcting Codes - Approach of [12]

An edge is said to be in error if its input symbol and output symbol (both from some appropriate field  $\mathbb{F}_q$ ) are not the same. We model the edge error as an additive error from  $\mathbb{F}_q$ . A *network-error* is a  $|\mathcal{E}|$  length vector over  $\mathbb{F}_q$ , whose components indicate the additive errors on the corresponding edges. A network code which enables every sink to correct any errors in any set of edges of cardinality at most  $\alpha$  is said to be an  $\alpha$  *network-error correcting code*. There have been different approaches to network-error correction [9]–[13]. We concern ourselves with the notations and approach of [12], as the algorithm in [12] lends itself to be extended according to the techniques of [7].

It is known [9] that the number of messages M in an  $\alpha$  network-error correcting code is upper bounded according to the *network Singleton bound* as  $M \leq q^{h-2\alpha}$ . Assuming that the message set is a vector space over  $\mathbb{F}_q$  of dimension k, we have  $k \leq h - 2\alpha$ .

A brief version of the algorithm given in [12] for constructing an  $\alpha$  network-error correcting code for a given single source, acyclic network that meets the network Singleton bound is shown in Algorithm 2. The construction of [12] is based on the network code construction algorithm of [4]. The algorithm constructs a network code such that all networkerrors in up to  $2\alpha$  edges will be corrected as long as the sinks know where the errors have occurred. Such a network code is then shown [12] to be equivalent to an  $\alpha$  network-error correcting code. Other equivalent (in terms of complexity) network-error correction algorithms can be found in [10] [11].

One way to understand Algorithm 2 which is relevant to our work is as follows. For each subset  $F \in \mathcal{F}$  of  $\mathcal{E}$ , Algorithm 2 considers a subnetwork of the original network consisting of k edge-disjoint paths from the imaginary source s' to each sink  $T \in \mathcal{T}$  and also  $m_T^F$  edge-disjoint paths from s' passing through the edges of F to each sink T which are also edge-disjoint with the k paths from s'. On this subnetwork, Algorithm 2 chooses network coding coefficients such that the k information symbols can still be multicast to each sink T irrespective of whatever information may flow on the  $m_T^F$ paths. If the same choice of coefficients can be chosen to satisfy this multicast-like constraint for each  $F \in \mathcal{F}$ , then there is a valid  $\alpha$  network-error correcting code which can be used to multicast the k information symbols from the source to all sinks in the network. This understanding of Algorithm 2 is key to understanding our algorithm for obtaining networkerror correcting codes for small field sizes. For further details on Algorithm 2, the reader is referred to [12].

Algorithm 2: Algorithm of [12] for constructing a network-error correcting code that meets the network Singleton bound.

(1) Let  $\mathcal{F}$  be the set of all subsets of  $\mathcal{E}$  of size  $2\alpha$ . Add an imaginary source s' and draw  $k = h - 2\alpha$  edges from s' to s.

(2) foreach  $F \in \mathcal{F}$  do

(i) Starting from the original network, add an imaginary node v at the midpoint of each edge  $e \in F$  and add an edge of unit capacity from s' to each v. (ii) foreach sink  $T \in \mathcal{T}$  do

Draw as many edge disjoint paths from s' to T passing through the imaginary edges added at Step (i) as possible. Let  $m_T^F (\leq 2\alpha)$  be the number of such paths.

Draw k edge disjoint paths passing through s that are also edge disjoint from the  $m_T^F$  paths drawn in the previous step.

#### end

(*iii*) Based on the techniques shown in the network coding algorithm of [4] on the subnetwork comprising of the identified edge disjoint paths, obtain a network code with the following property. Let  $B_T^F$  be the  $(k + 2\alpha) \times (k + m_T^F)$  matrix, the columns of which are the *h* length global encoding vectors (representing the linear combination of the *k* input symbols and  $2\alpha$  error symbols) of the incoming edges at sink *T* corresponding to the  $k + m_T^F$  edge disjoint paths. Then  $B_T^F$  must be full rank. As proved in [12], this ensures that the network code thus obtained is  $\alpha$  network-error correcting and meets the network Singleton bound.

end

It is shown in [12] that Algorithm 2 results in a network code which is an  $\alpha$  network-error correcting code meeting the network Singleton bound, as long as the field size

$$q > |\mathcal{T}||\mathcal{F}| = N \begin{pmatrix} |\mathcal{E}| \\ 2\alpha \end{pmatrix}.$$
 (1)

The above bound on field size was further tightened in [13], where it was shown that a construction of an  $\alpha$  network-error correcting code is possible if the field size q is such that

$$q > \sum_{T \in \mathcal{T}} |R_T(\alpha)|, \tag{2}$$

where  $R_T(\alpha)$  is a set defined in [13] for the sink T in the following way.

Definition 2: For a sink T, the set  $R_T(\alpha)$  is the set of all subsets of size  $2\alpha$  of the edge set  $\mathcal{E}$  satisfying the following properties for each  $\rho \in R_T(\alpha)$ .

• A collection of k edge-disjoint paths starting from the k imaginary incoming edges at the source node s to sink node T can be found.

• A collection of  $2\alpha$  edge-disjoint paths starting from each of the  $2\alpha$  edges to the sink T in  $\rho$  can be found, such that all these paths are also edge-disjoint from the k paths from s.

An algorithm is shown in [13] to construct  $\alpha$  networkerror correcting codes if the field size is greater than  $q > \sum_{T \in \mathcal{T}} |R_T(\alpha)|$ . In many networks (see [13], for example), this bound in (2) could be smaller than the bound in (1). However, in this work, we use the Algorithm 2 which is from [12] rather than the algorithm from [13]. We shall however give the value of the bound in (2) for an example network and show that our algorithm to obtain network-error correcting codes over small fields can obtain field sizes smaller than that of the bound in (2) also.

#### B. Network-Error Correction using Small Fields - Algorithm

Algorithm 3 constructs a network-error correcting code using small field sizes (conditioned on the existence of an irreducible polynomial of small degree satisfying the necessary requirements indicated in Step (5) of Algorithm 3). Note that for the case  $\alpha = 0$ , Algorithm 3 reduces to the EF algorithm, i.e., Algorithm 1. As in Algorithm 1, the major

Algorithm 3: Network-error correcting codes under small field sizes (1) With  $q = 2^{\lceil log(N|\mathcal{F}|) \rceil + 1} = 2^k$ , run Algorithm 2 to find an  $\alpha$  network-error correcting code meeting the network Singleton bound. Let the encoding coefficients for  $X_i$  be  $\alpha_i$ . (2) Express every  $X_i = \alpha_i$  as a binary polynomial  $p_i(X)$ of degree at most k-1 using the usual polynomial representation of the finite field  $\mathbb{F}_{2^k}$ . (3) foreach  $F \in \mathcal{F}$  do foreach sink  $T \in \mathcal{T}$  do Find a non-zero minor of the matrix  $B_T^F$ , obtained from a  $\left(k+m_{_T}^F\right) imes \left(k+m_{_T}^F\right)$  submatrix. At least one such minor exists as  $B_T^F$  has rank =  $k + m_T^F$ . Let the minor be  $f_T^F(X)$ , which can be of degree at most  $h\Lambda log(N|\mathcal{F}|)$ , according to Section II and the choice of our field size. end end (4) Calculate the polynomial

$$f(X) = \prod_{F \in \mathcal{F}} \prod_{T \in \mathcal{T}} f_T^F(X),$$

which has degree at most  $N|\mathcal{F}|h\Lambda log(N|\mathcal{F}|)$ . (5) Find an irreducible polynomial of least degree, g(X), which is coprime with f(X). (6) Let  $X_i = p_i(X) \pmod{g(X)}$ . Thus, each  $X_i$  can be viewed as an element in  $\frac{\mathbb{F}[X]}{(g(X))}$ . Because of the fact that  $f_T^F(X) \pmod{g(X)} \neq 0$  (as  $f(X) \pmod{g(X)} \neq 0$ ), the new  $B_T^F$  matrices obtained after the modulo operation are also full rank, which implies that the error correcting capability of the code is preserved.

step of Algorithm 3 is Step (5) which involves calculating

a polynomial g(X) coprime with a given polynomial f(X). According to the complexity calculations in [7], a brute force computation of Step (5) would require  $O(n^2 log(n))$  computations,  $n = deg(f(X)) = N|\mathcal{F}|h\Lambda[log(N|\mathcal{F}|)]$ . Before we propose our method to execute Step (5) efficiently in Subsection III-D, we give a justification for Algorithm 3.

#### C. Justification for Algorithm 3

No justification is required for the steps in Algorithm 3 except Step (5). The justification for Step (5) is as follows. Step (5) finds a g(X) which is coprime with the product polynomial f(X). In fact, in order to ensure that the error correction property of the original network code is preserved, it is sufficient if a polynomial g(X) is coprime with each polynomial  $f_T^F(X)$ , rather than their product f(X) (as shown in Step (5)). However, the following lemma shows that both are equivalent.

Lemma 1: Let  $\mathcal{U} = \{f_i : f_i \in \mathbb{F}[X], i = 1, 2, ..., n\}$  be a collection of univariate polynomials with coefficients from some field  $\mathbb{F}$ . A polynomial  $g \in \mathbb{F}[X]$  is relatively prime with all the polynomials in  $\mathcal{U}$  if and only if it is relatively prime with their product.

Proof: Appendix A.

# D. Fast algorithm for computing least degree coprime polynomial

Algorithm 4 is a fast method to compute the least degree irreducible polynomial g(X) among irreducible polynomials up to some degree m that is coprime with f(X). As a result,

<b>Algorithm 4:</b> Fast algorithm for computing $g(X)$				
(1) Let $\mathcal{P} = \left\{ X^{2^i} + X : i = 1, 2,, m \right\}.$				
(2) foreach $i = 1, 2,, m$ do				
Calculate $r(X) = f(X) \pmod{p_i(X)}$ .				
if $r(X)$ is non-zero then				
Break.				
end				
end				
(3) Pick $p_j(X)$ as the first polynomial (i.e. least degree)				
for which $r(X)$ is non-zero. Note that every $p_i(X) \in \mathcal{P}$				
is the product of all irreducible polynomials whose				
degree divides <i>i</i> . Also, all irreducible polynomials of				
degree $i < j$ divide $f(X)$ as all $p_i(X) f(X)$ for all				
i < j. Therefore, at least one of the irreducible				
polynomials of degree <i>i</i> is coprime with $f(X)$ .				
(4) Find one such polynomial $a(X)$ of degree <i>i</i> which is				
continue with $f(X)(mod \ n_{1}(X))$ and therefore				
equivalently with $f(X)$ (Subsection III E gives a				
instification of this stop)				
justification of this step).				

the key step (Step (5)) of Algorithm 3 can be performed much faster than having to compute g(X) by brute-force. Similarly, this fast algorithm also enables to quicken the key step (Step (4)) of Algorithm 1 so that its overall complexity is reduced.

Note that for using Algorithm 4 to implement Step (4) of Algorithm 1, we fix  $m = \lceil log(N) \rceil$  as any polynomial g(X)

coprime with f(X) is useful only if the degree of g(X) is less than  $\lceil log(N) \rceil + 1$ , as only such a g(X) can result in a network code using a smaller field than the one we started with. For the same reason, in using Algorithm 4 in conjunction with Algorithm 3, we choose  $m = \lceil log(N|\mathcal{F}|) \rceil$ .

#### E. Justification for Algorithm 4

The following lemma ensures that all polynomials which are found to be coprime with f(X) by directly computing the gcd (or the remainder for irreducible polynomials) in the brute force method (as done in Algorithm 1), can also be found by running Algorithm 4, using the set of polynomials  $\mathcal{P}$  up to the appropriate degree.

Lemma 2: For some field  $\mathbb{F}$ , let  $f,g \in \mathbb{F}[X]$  be two polynomials. Let  $p \in \mathbb{F}[X]$  be such that g|p. Then g is relatively prime with f if and only if g is relatively prime with  $f \pmod{p}$ .

Proof: Appendix B.

#### F. Complexity of Algorithm 4

The following proposition gives the complexity of Algorithm 4 for obtaining the coprime polynomial.

Proposition 1: The complexity of Algorithm 4 is at most  $O(2^{2m}) + O(mM)$ , where  $m = |\mathcal{P}|$ , and M = deg(f(X)). Proof: Appendix C.

Remark 1: Note that the worst-case complexity of Algorithm 4 with  $m = \lceil log(N) \rceil$  and  $M = hN \lceil log(N) \rceil \Lambda$  (corresponding to values required for running Step(4) of Algorithm 1) is  $O(N^2) + O(hN\Lambda(log(N))^2)$ . This is clearly lesser than the worst-case complexity of finding the coprime polynomial g(X) by brute-force, indicated in Section II. Even if we test for coprimeness only for polynomials up to degree  $\lceil log(N) \rceil$ , a brute-force execution of Step (4) of Algorithm 1 would have a worst-case complexity of  $O(N^2h\Lambda log(n))$  (where  $n = Nh\Lambda log(N)$ ), which is still greater than that of ours.

#### G. Complexity of Algorithm 3

We now calculate the complexity of Algorithm 3 (with Algorithm 4 used to implement its key step). The complexities of all the steps of Algorithm 3 is given by Table I, along with the references and reasoning for the mentioned complexities.

The only complexity calculations of Table I which are not straightforward are the complexities involved in calculating the polynomial g(X) coprime to f(X) and in calculating the nonzero minor of the matrix  $B_T^F$ . The complexity of calculating g(X) can be calculated using Proposition 1 using the values  $m = \lceil N | \mathcal{F} | \rceil$  and  $M = Nh | \mathcal{F} | \Lambda \lceil \log (N | \mathcal{F} |) \rceil$ .

Now for calculating the non-zero minor of the matrix  $B_T^F$ . There are  $\binom{h}{k+m_T^F}$  such minors, and calculating each takes  $O\left(\left(k+m_T^F\right)^3\right)$  multiplications over  $\mathbb{F}_q$ . As  $\left(k+m_T^F\right)$ can take values up to h, clearly the function to be maximized is of the form  $f(m) = \binom{h}{m}m^3$ , for m = 0, 1, ..., h. Proposition 2 gives the value of m for which such a function is maximized, based on which the value in Table I has been calculated. Proposition 2: For some positive integer n, let m be an integer such that  $0 \le m \le n$ . The function  $f(m) = \binom{n}{m} m^3$  is maximized at  $m = \begin{cases} \left( \lceil \frac{n}{2} \rceil + 1 \right) & \text{if } n \ge 2\\ 1 & \text{if } n = 1. \end{cases}$ Proof: Appendix D.

#### IV. ILLUSTRATIVE EXAMPLE - NETWORK-ERROR CORRECTION

The performance of Algorithm 1 (together with Algorithm 4) for a network coding problem on a combination network is shown in Appendix E. We now present a network-error correction example that uses Algorithm 3 (with Algorithm 4).

*Example 1:* Consider the network, with 18 edges, shown in Fig. 1. This network is from [11], in which a 1 networkerror correcting code meeting the network Singleton bound is given by brute-force construction for this network over  $\mathbb{F}_4$ , which is the smallest possible field over which such a code exists. According to the algorithm in [12], a 1 network-error correcting code can be constructed deterministically if  $q > 2 \begin{pmatrix} 18 \\ 2 \end{pmatrix} = 306$ . In Fig. 1, let the variable  $X_1$  denote the encoding coefficient between edges  $v_1 \rightarrow v_4$  and  $v_4 \rightarrow v_6$ . Similarly, let the variable  $X_2$  ( $X_3$ ) denote the local encoding coefficients between  $v_2 \rightarrow v_5$  ( $v_6 \rightarrow v_7$ ) and  $v_5 \rightarrow v_8$  ( $v_7 \rightarrow v_9$ ).



Fig. 1. Example network for network-error correction

Let  $q = 2^9$ . Let  $\mathcal{A} = \{\beta, \beta^{130}, \beta^{130}\}$  and  $\mathcal{B} = \{\beta^{132}, \beta^{391}, \beta^{391}\}$ , where  $\beta$  is a primitive element of  $\mathbb{F}_{2^9}$ . Let  $b_1(X) = X^9 + X^4 + 1$  be the primitive polynomial of degree 9 under consideration.

Consider two such 1 network-error correcting codes obtained using Algorithm 2 for the network of Fig. 1 as follows. Let  $\mathcal{A}$  and  $\mathcal{B}$  be two choices for the set  $\{X_1, X_2, X_3\}$  with all the other local encoding coefficients being unity. It can be verified that these two network codes can be used to transmit one error-free  $\mathbb{F}_{2^9}$  symbol from the source to both sinks, as long as only single edge errors occur in the network. Table II gives the results of running Algorithm 3 for this network

Step(s)	Complexity	Reasoning
Algorithm 2	$A := O\left( \mathcal{F} Nh\left( \mathcal{E}  \mathcal{F} N+ \mathcal{E} +h+2\alpha\right)\right).$	[12]
Identifying non-zero minor of matrix $B_T^F$	$B := O\left( \begin{pmatrix} h \\ m \end{pmatrix} m^3 \right), \text{ with } m = \left( \lceil \frac{h}{2} \rceil + 1 \right)$	Theorem 2
Computing the non-zero minor (over $\mathbb{F}_2[X]$ ) of $B_T^F$	$C := O\left(h^4 \Lambda log(N \mathcal{F} )\right) + O\left(\left(h\Lambda log(N \mathcal{F} )\right)^3\right)$	[7]
from a $(k + m_{\tau}^{F})$ square submatrix		
Calculating $f(X) = \prod_{F \in \mathcal{F}} \prod_{T \in \mathcal{T}} f_T^F(X)$ .	$D := O(alog(a))$ , where $a = Nh \mathcal{F} \Lambda log(N \mathcal{F} )$	[15]
Computing the coprime polynomial $g(X)$	$E := O\left(N^2  \mathcal{F} ^2\right) + O(Nh \mathcal{F} \Lambda log\left(N \mathcal{F} \right)^2).$	Proposition 1
Total complexity	$A + N \mathcal{F} (B + C + D) + E$	

 TABLE I

 COMPLEXITY CALCULATIONS FOR ALGORITHM 3

 TABLE II

 USING ALGORITHM 3 FOR THE NETWORK IN FIG. 1

Algorithm parameter	Network code defined by ${\cal A}$	Network code defined by ${\cal B}$
Degree of $f(X)$ ,		
the product of the 306 determinant polynomials	260	978
$p(X)$ : First $p_i(X)$ for which $f(X) \pmod{p_i(X)}$ is non-zero	$X^8 + X$	$X^4 + X$
$f(X) ({ m mod} p(X))$	$X^7 + X^6 + X^3 + X^2$	$X^3 + X$
g(X): Least degree polynomial coprime to $f(X)$	$X^3 + X + 1$	$X^2 + X + 1$
$\{X_1, X_2, X_3\}$ after the algorithm	$\left\{eta_8^1,eta_8^3,eta_8^3 ight\}$	$\{eta_4,eta_4,eta_4\}$

starting from these two codes, with  $\beta_4$  and  $\beta_8$  being the primitive elements of  $\mathbb{F}_4$  and  $\mathbb{F}_8$  respectively.

Except for  $\{X_1, X_2, X_3\}$ , all the other coding coefficients remain 1 over the respective fields. It is seen from Table II that the initial choice of the sets  $\mathcal{A}$  and  $\mathcal{B}$  for  $\{X_1, X_2, X_3\}$ affects the complexity of the problem (i.e., degree of f(X)) and also the field size of the final network code. With  $\mathcal{B}$ , the resultant network-error correcting code is over  $\mathbb{F}_4$ , exactly the one reported in [11] by brute force construction. Also, for sink  $T_1$  and  $T_2$ , the value of  $R_T(\alpha)$  can be computed to be  $R_{T_1}(1) = R_{T_2}(1) = 65$ . Thus the bound from [13] shown in (2) for this network can be computed to be q > 130. The field size of the network-error correcting code found using our algorithm can therefore still be lesser than that of the bound in [13].

#### V. CONCLUDING REMARKS

As in the original paper [7], questions remain open about the designing of a code using the minimal field size. The hardness of calculating the minimal field size is reflected by the fact that the initial choice of the network code and the primitive polynomial of the field over which the initial code is defined (using which the local encoding coefficients are represented as polynomials) control the resultant field size after the algorithm. These issues are illustrated by the examples in Section IV and Appendix E. However, it would be interesting to see if guarantees on the reduction of the field size can be given.

#### REFERENCES

- R. Ahlswede, N. Cai, R. Li and R. Yeung, "Network Information Flow", IEEE Transactions on Information Theory, vol.46, no.4, July 2000, pp. 1204-1216.
- [2] N. Cai, R. Li and R. Yeung, "Linear Network Coding", IEEE Transactions on Information Theory, vol. 49, no. 2, Feb. 2003, pp. 371-381.

- [3] R. Koetter and M. Medard, "An Algebraic Approach to Network Coding", IEEE/ACM Transactions on Networking, vol. 11, no. 5, Oct. 2003, pp. 782-795.
- [4] S. Jaggi, P. Sanders, P.A. Chou, M. Effros, S. Egner, K. Jain and L.M.G.M. Tolhuizen, "Polynomial time algorithms for multicast network code construction", IEEE Transactions on Information Theory, vol. 51, no. 6, June 2005, pp.1973-1982.
- [5] N. Harvey, "Deterministic network coding by matrix completion", MS Thesis, 2005.
- [6] A. Lehman and E. Lehman, "Complexity classification of network information flow problems", ACM SODA, 2004, New Orleans, USA, pp. 142-150.
- [7] J. B. Ebrahimi and C. Fragouli, "Algebraic algorithms for vector network coding", IEEE Transactions on Information Theory, vol. 57, no. 02, Feb 2011, pp. 996-1007.
- [8] S. Avestimehr, S N. Diggavi and D.N.C. Tse, "Wireless network information flow" Proceedings of Allerton Conference on Communication, Control, and Computing, Illinois, September 26-28, 2007, pp. 15-22.
- [9] R.W. Yeung and N. Cai, "Network error correction, part 1 and part 2", Communications and Information and Systems, vol. 6, 2006, pp. 19-36.
- [10] Z. Zhang, "Linear network-error Correction Codes in Packet Networks", IEEE Transactions on Information Theory, vol. 54, no. 1, Jan. 2008, pp. 209-218.
- [11] S. Yang and R.W. Yeung, "Refined Coding Bounds and Code Constructions for Coherent Network Error Correction", IEEE Transactions on Information Theory, Vol. 57, No. 3, March 2011, 1409-1424.
- [12] R. Matsumoto, "Construction Algorithm for Network Error-Correcting Codes Attaining the Singleton Bound", IEICE Transactions Fundamentals, Vol. E90-A, No. 9, September 2007, pp. 1729-1735.
- [13] X. Guang, F. Fu, and Z. Zhang, "Construction of Network Error Correction Codes in Packet Networks", Available on ArXiv, http://arxiv.org/abs/1011.1377, Nov. 2010.
- [14] N. Cai, R. Li, R. Yeung and Z. Zhang, "Network Coding Theory", Foundations and Trends in Communications and Information Theory, vol. 2, no.4-5, 2006.
- [15] A. Borodin and I. Munro, "The computational complexity of algebraic and numeric problems", American Elsevier Pub. Co., 1975.

#### APPENDIX A

#### **PROOF OF LEMMA 1**

*Proof: If part:* If g is relatively prime with the product of all the polynomials in U, then there exist polynomials  $a, b \in$ 

 $\mathbb{F}[X]$  such that

$$a\left(\prod_{i=1}^{n} f_i\right) + bg = 1.$$
(3)

For each  $j \in 1, 2, ..., n$ , we can rewrite (3) as

$$\left(a\prod_{i=1,i\neq j}^{n}f_{i}\right)f_{j}+bg=1$$

which implies that g is coprime with each  $f_j \in \mathcal{U}$ .

Only if part: Suppose g is relatively prime with all the polynomials in  $\mathcal{U}$ . Then, for each  $j \in 1, 2, ..., n$ , we can find polynomials  $a_j$  and  $b_j$  such that,  $a_j f_j + b_j g = 1$ . In particular,

$$a_1 f_1 + b_1 g = 1, (4)$$

$$a_2 f_2 + b_2 g = 1. (5)$$

Using (5) in (4),

$$\begin{split} \mathbf{l} &= a_1 f_1(a_2 f_2 + b_2 g) + b_1 g \\ &= (a_1 a_2) f_1 f_2 + (a_1 f_1 b_2 + b_1) g. \end{split}$$

Thus, g is relatively prime with  $f_1f_2$ . Continuing with the same argument, it is clear that g is relatively prime with  $\prod_{i=1}^{n} f_i$ .

#### APPENDIX B Proof of Lemma 2

*Proof:* Let f = qp + r for the appropriate quotient and remainder polynomials  $q, r \in \mathbb{F}[X]$  with deg(r) < deg(p). Also, as g|p, let p = hg, for the appropriate  $h \in \mathbb{F}[X]$ .

If part: As  $r = f \pmod{p}$  and g are relatively prime with each other, we can obtain polynomials  $a', b' \in \mathbb{F}[X]$  such that a'r + b'g = 1. Then, we must have

$$1 = a'(f - qp) + b'g$$
  
= a'(f - qhg) + b'g  
= a'f + (b' - a'qh)g

Thus f and g must be coprime with each other.

Only If part: Now assume that f and g are coprime with each other. This means we can obtain polynomials  $a, b \in \mathbb{F}[X]$  such that af + bg = 1. Then,

$$1 = a(qp + r) + bg$$
$$= a(qhg + r) + bg$$
$$= ar + (aqh + b)q$$

which means that g and r are coprime with each other, hence proving the lemma.

#### APPENDIX C PROOF OF PROPOSITION 1

Towards proving Proposition 1, we first prove the following lemma.

Lemma 3: Let  $f, p \in \mathbb{F}_2[X]$ , be such that deg(f) = M and  $p = X^n + X$ , for some non-negative integers M and n. The polynomial  $f \pmod{p}$  can be calculated using at most O(M) bit additions.

where a is the largest positive integer such that  $an-a+1 \leq M$ .

Now, note that calculating the polynomial  $f(\mod p)$ , is equivalent to adding up the rows of the arrangement, while retaining the coefficient  $f_0$  as it is. There are  $\left\lceil \frac{M}{n-1} \right\rceil$  rows in the arrangement, and adding any two rows requires at most n-1 additions. Thus, the total number of bit additions is O(M).

We are now ready to prove Proposition 1.

#### A. Proof of Proposition 1

**Proof:** The worst-case for Algorithm 4 would be j = m. By Lemma 3, computing  $f(X) \pmod{p_i(X)}$  for some  $p_i(X) \in \mathcal{P}$  takes at most M operations. As there are m such  $p_i(X)$ s, evaluating the remainders r(X)s costs Mm operations at most. Let

$$f(X)(\text{mod } p_j(X)) = f(X)$$

be the non-zero polynomial of degree at most  $2^m = 2^j$ .

Now, we have to determine the complexity in obtaining the polynomial of degree m which is coprime with f(X) (or equivalently with  $\tilde{f}(X)$ ).

There are approximately  $\frac{2^j}{j}$  irreducible polynomials of order j. It is known (see [15], for example) that for any two polynomials p(X) and q(X) (with degree w of p(X) larger than degree of q(X)), the complexity of dividing p(X) by q(X) (or equivalently, calculating  $p(X) \pmod{q(X)}$ ) is wlog(w). Thus, the complexity of dividing  $\tilde{f}(X)$  by every possible irreducible polynomial of degree j = m is at most  $\frac{2^m}{m}O(2^m \log(2^m)) = O(2^{2m})$ .

Thus, the total complexity for finding the least degree polynomial g(X) coprime with f(X) (which is assured of having a coprime factor of degree m+1) is at most  $O(2^{2m})+O(Mm)$ .

## Appendix D

#### **PROOF OF PROPOSITION 2**

*Proof:* The statement of the theorem is easy to verify for n = 1. Therefore, let  $n \ge 2$ . Let g(k) = f(k) - f(k+1), for some k, such that  $0 \le k \le n - 1$ . Then,

$$g(k) = \begin{pmatrix} n \\ k \end{pmatrix} k^3 - \begin{pmatrix} n \\ k+1 \end{pmatrix} (k+1)^3$$
$$= \begin{pmatrix} n \\ k \end{pmatrix} \left(k^3 - \frac{(n-k)}{(k+1)}(k+1)^3\right)$$
$$= \begin{pmatrix} n \\ k \end{pmatrix} \left(2k^3 + k^2(2-n) + k(1-2n) - n\right)$$
$$= \begin{pmatrix} n \\ k \end{pmatrix} \tilde{g}(k),$$

TABLE III  ${}_{6}C_{3}$  Network - Algorithm 1 (together with Algorithm 4)

Algorithm parameter	Global encoding vectors $\mathcal{A}$		Global encodi	ng vectors ${\cal B}$
	Prim. poly. $b_1(X)$	<b>Prim. poly.</b> $b_2(X)$	<b>Prim. poly.</b> $b_1(X)$	Prim. poly. $b_2(X)$
Degree of $f(X)$ , the product of the				
20 determinant polynomials	20	40	30	55
$p(X)$ : First $p_i(X)$ for which				None of the form
$f(X) \pmod{p_i(X)}$ is non-zero	$X^4 + X$	$X^8 + X$	$X^8 + X$	$X^{2^i} + X$ , for $i \le 4$
$f(X) \pmod{p(X)}$	$X^2 + X$	$X^7 + X^6 + X^3 + X$	$X^7 + X^6 + X^5 + X^2$	Not applicable
g(X): Least degree				
polynomial coprime to $f(X)$	$X^2 + X + 1$	$X^3 + X + 1$	$X^3 + X + 1$	Not applicable
	$\left[\begin{array}{c}1\\0\\0\end{array}\right]\left[\begin{array}{c}0\\1\\0\end{array}\right]$	$\left[\begin{array}{c}1\\0\\0\end{array}\right]\left[\begin{array}{c}0\\1\\0\end{array}\right]$	$\left[\begin{array}{c}1\\0\\0\end{array}\right]\left[\begin{array}{c}0\\1\\0\end{array}\right]$	
Resultant network code	$\left[\begin{array}{c}0\\0\\1\end{array}\right]\left[\begin{array}{c}1\\1\\1\end{array}\right]$	$\left[\begin{array}{c}0\\0\\1\end{array}\right]\left[\begin{array}{c}1\\1\\1\end{array}\right]$	$\left[\begin{array}{c}0\\0\\1\end{array}\right]\left[\begin{array}{c}1\\1\\1\end{array}\right]$	Not applicable
	$\left[\begin{array}{c}1\\\beta_4\\\beta_4^2\\\beta_4^2\end{array}\right]\left[\begin{array}{c}1\\\beta_4^2\\\beta_4\end{array}\right]$	$\begin{bmatrix} 1\\ \beta_8\\ \beta_8^4\\ \beta_8^4 \end{bmatrix} \begin{bmatrix} 1\\ \beta_8^4\\ \beta_8^2 \end{bmatrix}$	$\begin{bmatrix} 1\\ \beta_8\\ \beta_8^3\\ \beta_8^3 \end{bmatrix} \begin{bmatrix} 1\\ \beta_8^3\\ \beta_8^6\\ \beta_8^6 \end{bmatrix}$	

where  $\tilde{g}(k) = (2k^3 + k^2(2-n) + k(1-2n) - n)$ . Proving the statement of the theorem is then equivalent to showing that both of the following two statements are true, which we shall do separately for even and odd values of n.

- $\tilde{g}(k) < 0$  for all integers  $0 \le k \le \lceil \frac{n}{2} \rceil$ .
- $\tilde{g}(k) > 0$  for all integers  $\lceil \frac{n}{2} \rceil + 1 \le k \le n 1$ .

Case-A (n is even): Let  $k = \frac{n}{2} + i$ , for some integer i such that  $-\frac{n}{2} \le i \le \frac{n}{2} - 1$ . Then,

$$\tilde{g}(k) = 2k^3 + k^2(2 - 2k + 2i) + k(1 - 4k + 4i) - 2k + 2i$$
  

$$\tilde{g}(k) = k^2(-2 + 2i) + k(4i - 1) + 2i.$$
(6)

For  $-\frac{n}{2} \leq i \leq 0$ , it is clear from (6) that  $\tilde{g}(k) < 0$ . If  $1 \leq i \leq \left(\frac{n}{2} - 1\right)$ , it is clear that  $\tilde{g}(k) > 0$ . Thus, for even values of n, the theorem is proved.

*Case-B* (*n* is odd): Let  $k = \lceil \frac{n}{2} \rceil + i = \left(\frac{n+1}{2}\right) + i$ , for some integer *i* such that  $-\left(\frac{n+1}{2}\right) \le i \le \left(\frac{n-3}{2}\right)$ . Then,

$$\tilde{g}(k) = 2k^3 + k^2(2 - 2k + 2i + 1) + k(1 - 4k + 4i + 2) - 2k + 2i + 1 \tilde{g}(k) = k^2(-1 + 2i) + k(1 + 4i) + 2i + 1.$$
(7)

Now, for i = 0,  $k = \left(\frac{n+1}{2}\right) \ge 2$  (as  $n \ge 2$  and is odd). Hence,  $\tilde{g}(k) = -k^2 + k + 1 < 0$  for i = 0. If  $-\left(\frac{n+1}{2}\right) \le i < 0$ , then by (7), it is clear that  $\tilde{g}(k) < 0$ . Thus for all  $-\left(\frac{n+1}{2}\right) \le i \le 0$ ,  $\tilde{g}(k) < 0$ .

 $-\left(\frac{n+1}{2}\right) \le i \le 0, \ \tilde{g}(k) < 0.$ For  $1 \le i \le \left(\frac{n-3}{2}\right)$ , again by (7), it is clear that  $\tilde{g}(k) > 0$ , and thus the theorem holds for odd values of n. This completes the proof.

### Appendix E

### EXAMPLE - NETWORK CODING

*Example 2:* Consider the  $\begin{pmatrix} 6\\ 3 \end{pmatrix}$  network shown in Fig. 2. This network has 20 sinks, each of which has 3 incoming edges from some 3-combination of the 6 intermediate nodes,



Fig. 2.  ${}_{6}C_{3}$  network with 20 sinks

thus the mincut h being 3. Using the methods in [3]–[5], a 3dimensional network code can be constructed for this network as long as the field size q > 20. Let  $q = 2^5$ . Consider the following sets of vectors in  $\mathbb{F}_{32}^3$ , with  $\beta$  being a primitive element of  $\mathbb{F}_{32}$ .

$$\mathcal{A} = \left\{ \begin{array}{c} \begin{bmatrix} 1\\0\\0 \end{bmatrix}, \begin{bmatrix} 0\\1\\0 \end{bmatrix}, \begin{bmatrix} 0\\0\\1 \end{bmatrix}, \begin{bmatrix} 0\\0\\1 \end{bmatrix}, \\ \begin{bmatrix} 1\\1\\1 \end{bmatrix}, \begin{bmatrix} 1\\\beta^{18}\\\beta^{18} \end{bmatrix}, \begin{bmatrix} 1\\\beta^{18}\\\beta^{5} \end{bmatrix} \right\}, \\ \mathcal{B} = \left\{ \begin{array}{c} \begin{bmatrix} 1\\0\\0\\0 \end{bmatrix}, \begin{bmatrix} 0\\1\\0\\0 \end{bmatrix}, \begin{bmatrix} 0\\1\\0\\0 \end{bmatrix}, \begin{bmatrix} 0\\0\\1\\0 \end{bmatrix}, \begin{bmatrix} 1\\\beta^{18}\\\beta^{5} \end{bmatrix} \right\}.$$

Let  $b_1(X) = X^5 + X^2 + 1$  and  $b_2(X) = X^5 + X^3 + X^2 + X + 1$ , both of them being primitive polynomials of degree 5. Note that  $\mathcal{A}$  and  $\mathcal{B}$  are valid choices (using either  $b_1(X)$  or  $b_2(X)$  as the primitive) for the global encoding vectors of the 6 outgoing edges from the source, representing deterministic network coding solutions for a 3-dimensional network code for this network. We assume that the intermediate nodes simply forward the incoming symbols to their outgoing edges, i.e., their local encoding coefficients are all 1.

Table III illustrates the results obtained with the execution of Algorithm 1, with Algorithm 4 being used to compute the coprime polynomial for this network with the original deterministic solutions being  $\mathcal{A}$  or  $\mathcal{B}$ , with  $b_1(X)$  and  $b_2(X)$ as the primitive polynomial of  $\mathbb{F}_{32}$ . The solutions (global encoding vectors of the 6 edges from the source) obtained for the  $\begin{pmatrix} 6\\3 \end{pmatrix}$  network, after the modulo operations of the individual coding coefficients using the polynomial g(X), are also shown in Table III. It can be checked that both of these sets of vectors are valid network coding solutions for a 3-dimensional network code for the  $\begin{pmatrix} 6\\3 \end{pmatrix}$  network.

It is seen that for the set  $\mathcal{A}$  being the choice of the network code in the first step of Algorithm 1 and with  $b_1(X)$  being the primitive polynomial, the final coprime polynomial has degree 2 and thus resulting in a code  $\mathbb{F}_4$ , which is in fact the smallest possible field for which a solution exists for this network. For  $\mathcal{B}$  with the primitive polynomial  $b_2(X)$ , no solutions are found using characteristic two finite fields of cardinality less than 32.