Network-Error Correcting Codes using Small Fields

K. Prasad
Dept. of ECE, Indian Institute of Science
Bangalore 560012, India
Email:prasadk5@ece.iisc.ernet.in

B. Sundar Rajan
Dept. of ECE, Indian Institute of Science,
Bangalore 560012, India
Email: bsrajan@ece.iisc.ernet.in

***Abstract*—Recently, Ebrahimi and Fragouli proposed an algorithm to construct scalar network codes using small fields (and vector network codes of small lengths) satisfying multicast constraints in a given single-source, acyclic network. The contribution of this paper is two fold. Primarily, we extend the scalar network coding algorithm of Ebrahimi and Fragouli (henceforth referred to as the EF algorithm) to block network-error correction. Existing construction algorithms of block network-error 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. We give an algorithm which, starting from a given network-error correcting code, can obtain another network code using a small field, with the same error correcting capability as the original code. Our secondary contribution is to modify the EF Algorithm itself. The major step in the EF algorithm is to find a least degree irreducible polynomial which is coprime to another large degree polynomial. We suggest an alternate method to compute this coprime polynomial, which is faster than the brute force method in the work of Ebrahimi and Fragouli.**

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 a hard problem [6]. Most recently, 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]–[12]. Using the algorithm of [12], a network code which can correct any errors occurring in at most α edges can be constructed, as long as the field size q is such that

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

where $\mathcal{E}$ is the set of edges in the network. The algorithms of [10], [11] have similar requirements to construct such network-error 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 this work, we extend the EF algorithm to block network-error correction using small fields. As in [7], 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 extend the EF algorithm of [7] to construct network-error correcting codes using small fields, by bridging the techniques of the EF algorithm and the network-error correction algorithm of [12].
- The major step in the EF 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 a alternate algorithm for computing the polynomial coprime with $f(X)$. This is shown to have lesser complexity than that of the EF algorithm, which simply adopts a brute force method to do the same.

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. In Section III, we review the EF algorithm briefly and then propose our modification to it, and prove that the modified algorithm has lesser complexity than the original technique in the EF algorithm. Section IV presents our algorithm for constructing network-error correcting codes using small field sizes, along with calculations of the complexity of the algorithm. Examples illustrating the

algorithm performance for network coding and error correction are presented in Section V. Finally, we conclude the paper in Section VI with comments and directions for further research.

II. Notations and definitions

The model for acyclic networks considered in this paper is as in [13]. An acyclic network can be represented as a acyclic directed multi-graph $G = (\mathcal{V}, \mathcal{E})$, 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 the directed multi-graph representing the network has unit *capacity* (can carry utmost one symbol from $\mathbb{F}_q$). 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 [13]).

Let $s \in \mathcal{V}$ be the source node and $\mathcal{T}$ be the set of $N(=|\mathcal{T}|)$ receivers. Let $h_{_T}$ 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.

An h'-dimensional network code ($h' \leq h$) is one which can be used to transmit h' symbols simultaneously from s to T, and can be described by the three matrices A (of size $h' \times |\mathcal{E}|$), F (of size $|\mathcal{E}| \times |\mathcal{E}|$), and D_T (of size $|\mathcal{E}| \times h'$ for every sink $T \in \mathcal{T}$), each having elements from some finite field $\mathbb{F}_q$. Further details on the structure of these matrices can be found in [3] and [7]. 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 $M_T := AFD_T = AF_T$.

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_is are known as the *local encoding coefficients* [13]. For an edge e in a network with a h'-dimensional network code in place, the *global encoding vector* [13] 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 Λ be the length of the longest path from the source to any sink. Because of the structure of the matrices A, F and D_T, it is seen [7] that the matrix M_T has degree at most Λ in any particular variable X_i and also a total degree (sum of the degrees across all variables in any monomial) of Λ. Let $f_{_T}\left(X_1, X_2, ..X_{|\{X_i\}|}\right)$ 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 f are at most $h'\Lambda$ and $Nh'\Lambda$ respectively.

III. Modifying the EF algorithm

After briefly recollecting the EF algorithm, we shall proceed to modify its key step so that the overall complexity of the algorithm is reduced.

A. The EF Algorithm - Review

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

(1) Assign values α_is to the scalar coding coefficients X_is 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_Ts 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_is in the matrices M_T, calculate the determinants of M_T as the polynomials $f_{_T}(X) \in \mathbb{F}[X]$, and also find $f(X) = \prod_{T \in \mathcal{T}} f_{_T}(X)$. 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 $X_i = p_i(X) (\text{mod } g(X))$. Thus, each X_i can be viewed as an element in $\frac{\mathbb{F}[X]}{(g(X))}$. Also, for each sink T, the matrices M_T remain invertible as $f_{_T}(X) (\text{mod } g(X)) \neq 0$, as $f(X) (\text{mod } g(X)) \neq 0$.

The following lemma ensures that such a coprime $g(X)$ exists and can be found in polynomial time.

Lemma 1: [7] If $f(X)$ is a non-zero binary polynomial of degree n, there exists a coprime polynomial $g(X)$ of degree at most $log(n+1) - 1$, and we can identify it in polynomial time.

Remark 1: [7] The worst-case complexity of computing $g(X)$ is $O\left(n^2 log(n)\right)$, where $n = Nh\Lambda log(N)$.

B. Fast algorithm for computing least degree coprime polynomial

We now present a fast algorithm for computing the least degree irreducible polynomial $g(X)$ that is coprime with $f(X)$. Note that 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. Using this fact, we give Algorithm 2 which computes a least degree irreducible polynomial $g(X)$ that is coprime with $f(X)$.

Algorithm 2: Fast algorithm for computing a least degree coprime polynomial to the given polynomial $f(X)$.

(1) Let $\mathcal{P} = \left\{ X^{2^i} + X : i = 1, 2, ..., \lceil log(N) \rceil \right\}$.
(2) **foreach** $p_i(X) \in \mathcal{P}$ **do**
  Calculate $r(X) = f(X) (\text{mod } p_i(X))$.
  if $r(X)$ *is non-zero* **then**
    Break.
  end
end
(3) Let $p_j(X)$ be the first polynomial 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. 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 j is coprime with $f(X)$. Find one such polynomial $g(X)$.

C. Justification for Algorithm 2

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 2, using the set of polynomials $\mathcal{P}$ upto the appropriate degree.

Lemma 2: For some field $\mathbb{F}$, let $f, g \in \mathbb{F}[X]$ be two polynomials relatively prime with each other. Let $p \in \mathbb{F}[X]$ such that $g|p$. Then g is also relatively prime with the polynomial $f(\text{mod } p)$.

Proof: As f and g are coprime with each other, we can obtain polynomials $a, b \in \mathbb{F}[X]$ such that $af+bg = 1$. 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 $p = hg$. Then,

$$\begin{aligned} 1 &= a(qp + r) + bg \\ &= a(qhg + r) + bg \\ &= ar + (aqh + b)g, \end{aligned}$$

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

D. Complexity of Algorithm 2

We now prove that our method for Step 4 of Algorithm 1 has less complexity than that of [7]. Towards that end, 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 $f(\text{mod } p)$ can be calculated using at most $O(M)$ bit additions.

Proof: Let $f = \sum_{i=0}^{M} f_i X^i$. We arrange the coefficients of f as follows.

$$\begin{array}{cccccccc} f_0 & f_1 & f_2 & ... & ... & ... & f_{n-1} \\ & f_n & f_{n+1} & ... & ... & ... & f_{2n-2} \\ & f_{2n-1} & f_{n+1} & ... & ... & ... & f_{3n-3} \\ & . & . & ... & ... & ... & . \\ & . & . & ... & ... & ... & . \\ & f_{an-a+1} & ... & f_M & 0 & ... & 0 \end{array} ,$$

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

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

Proposition 1: The complexity of Algorithm 2 is at most $O(N^2) + O(hN(log(N))^2\Lambda)$.

Proof: The worst-case for Algorithm 2 would be $j = \lceil log(N) \rceil$. By Lemma 3, computing $f(X)(\text{mod } p_i(X))$ for some $p_i(X) \in \mathcal{P}$ takes at most $O(h\Lambda N log(N))$ operations. As there are $\lceil log(N) \rceil$ such $p_i(X)$s, evaluating the remainders $r(X)$s costs $O(hN(log(N))^2\Lambda)$ operations at most. Let

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

be the polynomial of degree at most $2^{\lceil log(N) \rceil} = 2^j$.

Now, we have to determine the complexity in obtaining the polynomial of degree $\lceil log(N) \rceil$ 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 [14], for example) that for any two polynomials $p(X)$ and $q(X)$ (atleast one of them of degree w), the complexity of dividing $p(X)$ by $q(X)$ (or equivalently, calculating $p(X)(\text{mod } q(X))$) is $wlog(w)$. Thus, the complexity of dividing $\tilde{f}(X)$ by every possible irreducible polynomial of degree $j = \lceil log(N) \rceil$ is at most $\frac{2^j}{j} O\left(2^j log(2^j)\right) = O(N^2)$.

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 $\lceil log(N) \rceil + 1$) is at most $O(N^2) + O(hN(log(N))^2\Lambda)$. ∎

Remark 2: Note that the worst-case complexity of Algorithm 2 is lesser than the worst-case complexity of finding the coprime polynomial $g(X)$ according to [7] (which assumes a direct test for coprimeness of $f(X)$ and the candidate polynomials), indicated in Remark 1. Even if we test for coprimeness only for polynomials upto degree $\lceil log(N) \rceil$, the algorithm of [7] would still have a worst-case complexity of $O\left(N^2 h\Lambda log(n)\right)$, where $n = Nh\Lambda log(N)$.

IV. 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 α is said to be a α *network-error correcting code*. There have been different approaches to network-error correction [9]–[12]. 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 α 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

$$k \leq h - 2\alpha.$$

This bound was later refined [11] to accommodate for the different mincuts and different error capabilities at different sinks as

$$k \leq h_{_T} - 2\alpha_{_T},$$

where sink $T \in \mathcal{T}$ can correct any network-error with errors in at most $\alpha_{_T}$ edges.

Algorithm 3 is a brief version of the algorithm given in [12] for constructing an α network-error correcting code for a given single source, acyclic network that meets the network Singleton bound. The construction of [12] is based on the network code construction algorithm of [4]. The algorithm constructs a network code such that all network-errors in upto 2α 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 α network-error correcting code.

It is shown in [12] that Algorithm 3 results in a network code which is a α network-error correcting code meeting the network Singleton bound, as long as the field size

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

B. Network Error Correction using Small Fields - Algorithm

Algorithm 4, shown in the next page, 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).

C. Justification for Step 5 of Algorithm 4

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 of Algorithm 4.

Algorithm 3: 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 edge set of size 2α. Add an imaginary source s' and draw $k = h - 2\alpha$ edges from s' to s.
(2) **foreach** $F \in \mathcal{F}$ **do**
  (i) 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 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) Use the algorithm from [4] using the identified edge disjoint paths such that it ultimately gives a network code with the following property. Let B_T^F be the $\left(k + m_{_T}^F\right) \times (k + 2\alpha)$ matrix, the columns of which are the h length global encoding vectors (representing the linear combination of the k input symbols and 2α 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 α network-error correcting and meets the network Singleton bound.
end

However, the following lemma shows that both are equivalent.

Lemma 4: 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: If part: If g is relatively prime with the product of all the polynomials in $\mathcal{U}$, then there exist polynomials $a, b \in \mathbb{F}[X]$ such that

$$a \left(\prod_{i=1}^{n} f_i \right) + bg = 1. \tag{2}$$

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

Algorithm 4: Network-error correcting codes under small field sizes

(1) With $q = 2^{\lceil log(N|\mathcal{F}|)\rceil+1}$, run Algorithm 3 to find an α network-error correcting code meeting the network Singleton bound. Let the encoding coefficients for X_i be α_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 $\left(k + m_T^F\right) \times \left(k + m_T^F\right)$ 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 (1).
  end
end
(4) Calculate the polynomial

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

(5) Using Algorithm 2 with the set $\mathcal{P} = \left\{X^{2^i} + X : i = 1, 2, ..., \lceil log(N|\mathcal{F}|)\rceil\right\}$, find an irreducible polynomial of least degree, $g(X)$, which is coprime with $f(X)$.
(6) Let $X_i = p_i(X) (\text{mod } g(X))$. Thus, each X_i can be viewed as an element in $\frac{\mathbb{F}[X]}{(g(X))}$. Because of the fact that $f_T^F(X) (\text{mod } g(X)) \neq 0$, 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.

polynomials a_j and b_j such that, $a_j f_j + b_j g = 1$. In particular,

$$a_1 f_1 + b_1 g = 1, \tag{3}$$
$$a_2 f_2 + b_2 g = 1. \tag{4}$$

Using (4) in (3),

$$\begin{aligned} 1 &= a_1 f_1 (a_2 f_2 + b_2 g) + b_1 g \\ &= (a_1 a_2) f_1 f_2 + (a_1 f_1 b_2 + b_1) g. \end{aligned}$$

Thus, g is relatively prime with $f_1 f_2$. Continuing with the same argument, it is clear that g is relatively prime with $\prod_{i=1}^n f_i$. ■

D. Complexity of Algorithm 4

The complexity of Algorithm 4 is given by Table I, along with the references and reasoning for the mentioned complexities for every step of the algorithm.

The only complexity calculation of Table I which remains to be explained is the complexity involved in identifying and calculating the non-zero minor of the matrix B_T^F. There are $\begin{pmatrix} h \\ k + m_T^F \end{pmatrix}$ 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 upto h, clearly the function to be maximized is of the form $f(m) = \begin{pmatrix} h \\ m \end{pmatrix} m^3$, 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 $0 \leq m \leq n$. The function

$$f(m) = \begin{pmatrix} n \\ m \end{pmatrix} m^3$$

is maximized at

$$m = \begin{cases} \left(\lceil \frac{n}{2} \rceil + 1\right) & \text{if } n \geq 2 \\ 1 & \text{if } n = 1. \end{cases}$$

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

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

where $\tilde{g}(k) = \left(2k^3 + k^2(2-n) + k(1-2n) - n\right)$. 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 $0 \leq k \leq \lceil \frac{n}{2} \rceil$.
- $\tilde{g}(k) > 0$ for all integers $\lceil \frac{n}{2} \rceil + 1 \leq k \leq n-1$.

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

$$\tilde{g}(k) = 2k^3 + k^2(2 - 2k + 2i) + k(1 - 4k + 4i) - 2k + 2i$$
$$\tilde{g}(k) = k^2(-2 + 2i) + k(4i - 1) + 2i. \tag{5}$$

For $-\frac{n}{2} \leq i \leq 0$, it is clear from (5) 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 $-\left(\frac{n+1}{2}\right) \leq i \leq \left(\frac{n-3}{2}\right)$. Then,

$$\begin{aligned} \tilde{g}(k) = 2k^3 \ &+ k^2(2 - 2k + 2i + 1) \\ &+ k(1 - 4k + 4i + 2) - 2k + 2i + 1 \\ \tilde{g}(k) = \ & k^2(-1 + 2i) + k(1 + 4i) + 2i + 1. \end{aligned} \tag{6}$$

Now, for $i = 0$, $k = \left(\frac{n+1}{2}\right) \geq 2$ (as $n \geq 2$ and is odd). Hence, $\tilde{g}(k) = -k^2 + k + 1 < 0$ for $i = 0$. If $-\left(\frac{n+1}{2}\right) \leq i < 0$, then by (6), it is clear that $\tilde{g}(k) < 0$. Thus for all $-\left(\frac{n+1}{2}\right) \leq i \leq 0$, $\tilde{g}(k) < 0$.

For $1 \leq i \leq \left(\frac{n-3}{2}\right)$, again by (6), it is clear that $\tilde{g}(k) > 0$, and thus the theorem holds for odd values of n. This completes the proof. ■

TABLE I
COMPLEXITY CALCULATIONS FOR ALGORITHM 4

Step(s)	Complexity	Reasoning
Algorithm 3	$A := O\left(\lvert\mathcal{F}\rvert Nh\left(\lvert\mathcal{E}\rvert\lvert\mathcal{F}\rvert N + \lvert\mathcal{E}\rvert + h + 2\alpha\right)\right).$	[12]
Identifying non-zero minor of matrix B_T^F	$B := O\left(\binom{h}{m} m^3\right)$, 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 from a $(k + m_T^F)$ square submatrix	$C := O\left(h^4 \Lambda log(N\lvert\mathcal{F}\rvert)\right) + O\left(\left(h\Lambda log\left(N\lvert\mathcal{F}\rvert\right)\right)^3\right)$	[7]
Calculating $f(X) = \prod_{F\in\mathcal{F}} \prod_{T\in\mathcal{T}} f_T^F(X)$.	$D := O\left(alog(a)\right)$, where $a = Nh\lvert\mathcal{F}\rvert\Lambda log\left(N\lvert\mathcal{F}\rvert\right)$	[14]
Computing the coprime polynomial $g(X)$	$E := O\left(N^2\lvert\mathcal{F}\rvert^2\right) + O(Nh\lvert\mathcal{F}\rvert\Lambda log\left(N\lvert\mathcal{F}\rvert\right)^2)$.	Proposition 1
Total complexity	$A + N\lvert\mathcal{F}\rvert(B + C + D) + E$	

V. ILLUSTRATIVE EXAMPLES

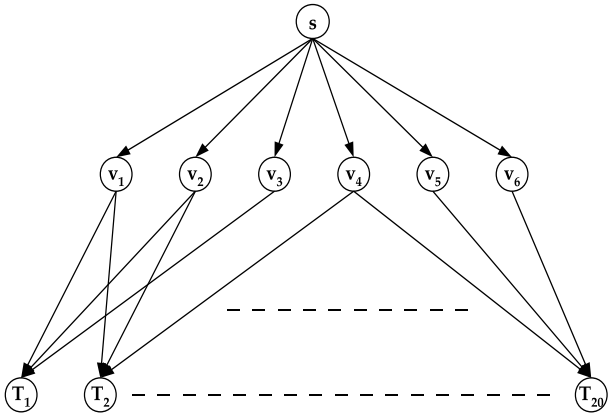

Fig. 1. ${}_6C_3$ network with 20 sinks

Example 1: Consider the $\binom{6}{3}$ network shown in Fig. 1. This network has 20 sinks, each of which has 3 incoming edges from some 3-combination of the 6 intermediate nodes, thus the mincut h being 3. Using the methods in [3]–[5], a 3-dimensional 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 β being a primitive element of $\mathbb{F}_{32}$.

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

$$\mathcal{B} = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ \beta^{6} \end{bmatrix}, \begin{bmatrix} 1 \\ \beta \\ \beta^{18} \end{bmatrix}, \begin{bmatrix} 1 \\ \beta^{18} \\ \beta^{5} \end{bmatrix} \right\}.$$

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 II illustrates the results obtained with the execution of Algorithm 1, with Algorithm 2 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 $\binom{6}{3}$ network, after the modulo operations of the individual coding coefficients using the polynomial $g(X)$, are also shown in Table II. It can be checked that both of these sets of vectors are valid network coding solutions for a 3-dimensional network code for the $\binom{6}{3}$ 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.

Example 2: Consider the network, with 18 edges, shown in Fig. 2. This network is from [15], in which a 1 network-error 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 $q > 2\binom{18}{2} = 306$. In Fig. 2, let the variable X_1 denote the encoding coefficient between edges $v_1 \to v_4$ and $v_4 \to v_6$. Similarly, the variable X_2 (X_3) denote the local encoding coefficients between $v_2 \to v_5$ ($v_6 \to v_7$) and

TABLE II
${}_6C_3$ NETWORK - ALGORITHM 1 (TOGETHER WITH ALGORITHM 2)

Algorithm parameter	**Global encoding vectors $\mathcal{A}$**		**Global encoding vectors $\mathcal{B}$**	
	Prim. poly. $b_1(X)$	**Prim. poly.** $b_2(X)$	**Prim. poly.** $b_1(X)$	**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 $f(X)(\text{mod } p_i(X))$ is non-zero	X^4+X	X^8+X	X^8+X	None of the form $X^{2^i}+X$, for $i \leq 4$
$f(X)(\text{mod } p(X))$	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
Resultant network code	$\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}$ $\begin{bmatrix}1\\\beta_4\\\beta_4^2\end{bmatrix}\begin{bmatrix}1\\\beta_4^2\\\beta_4\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}$ $\begin{bmatrix}1\\\beta_8\\\beta_8^4\end{bmatrix}\begin{bmatrix}1\\\beta_8^4\\\beta_8^2\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}$ $\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}$ $\begin{bmatrix}1\\\beta_8\\\beta_8^3\end{bmatrix}\begin{bmatrix}1\\\beta_8^3\\\beta_8^6\end{bmatrix}$	Not applicable

TABLE III
USING ALGORITHM 4 FOR THE NETWORK IN FIG. 2

Algorithm parameter	**Network code defined by $\mathcal{A}$**	**Network code defined by $\mathcal{B}$**
Degree of $f(X)$, the product of the 306 determinant polynomials	260	978
$p(X)$: First $p_i(X)$ for which $f(X)(\text{mod } p_i(X))$ is non-zero	X^8+X	X^4+X
$f(X)(\text{mod } p(X))$	$X^7+X^6+X^3+X^2$	X^3+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	$\{\beta_8^1, \beta_8^3, \beta_8^3\}$	$\{\beta_4, \beta_4, \beta_4\}$

$v_5 \to v_8$ ($v_7 \to v_9$).

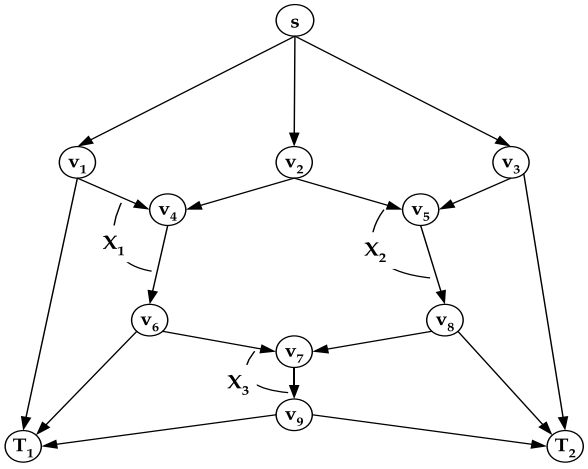

Fig. 2. Example network for network-error correction

Let $q = 2^9$. Let $\mathcal{A} = \{\beta, \beta^{130}, \beta^{130}\}$ and $\mathcal{B} = \{\beta^{132}, \beta^{391}, \beta^{391}\}$, where β 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 3 for the network of Fig. 2 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 not more than single edge errors occur in the network. Table III gives the results of running Algorithm 4 for this network starting from these two codes, with β_4 and β_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. As in Example 1, the initial choice of the sets $\mathcal{A}$ and $\mathcal{B}$ for $\{X_1, X_2, X_3\}$ results in the final network code being over different field sizes.

With $\mathcal{B}$, the resultant network-error correcting code is over $\mathbb{F}_4$, exactly the one reported in [15] by brute force construction.

VI. CONCLUSION

A new and faster method of computing a coprime polynomial to a given polynomial has been presented. thereby improving the performance of the EF algorithm, which is applicable to scalar and vector network coding. Based on the EF algorithm, a method has been presented which can obtain network-error correcting codes using small fields that meet the network Singleton bound. This technique can be adapted to obtain network-error correcting codes meeting the refined

Singleton bound, or for linear deterministic networks which permit solutions similar to those obtained in [7].

As in the original paper [7], questions remain open about the achievability of a code using the minimal field size. As illustrated by the examples in Section V, factors such as 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.

ACKNOWLEDGMENT

The authors would like to thank Raman Sankaran of the CSA Dept., IISc, for useful discussions regarding the complexity calculations of the algorithms presented in this paper. This work was supported partly by the DRDO-IISc program on Advanced Research in Mathematical Engineering through a research grant and partly by the INAE Chair Professorship grant to B. S. Rajan.

REFERENCES

[1] 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 Trans. Inf. 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. Ebrahimi and C. Fragouli, "Vector Network Coding Algorithms", IEEE ISIT, 2010, Austin, Texas, USA, June 13-18.

[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", Comm. in Inform. 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 for network error Correction", ITW on Information Theory for Wireless Networks, July 1-6, 2007, Bergen, Norway, pp. 1-5.

[12] R. Matsumoto, "Construction Algorithm for Network Error-Correcting Codes Attaining the Singleton Bound", IEICE Trans. Fundamentals, Vol. E90-A, No. 9, September 2007, pp. 1729-1735.

[13] 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.

[14] A. Borodin and I. Munro, "The computational complexity of algebraic and numeric problems", American Elsevier Pub. Co., 1975.

[15] S. Yang and R.W. Yeung, "Construction of Linear Network Codes that achieve a Refined Singleton Bound", ISIT2007, Nice, France, June 24-29, 2007, pp. 1576-1580.