Polynomial Time Algorithms for Multicast Network Code Construction

Sidharth Jaggi, *Student Member, IEEE*, Peter Sanders, Philip A. Chou, *Fellow, IEEE*, Michelle Effros, *Senior Member, IEEE*, Sebastian Egner, *Senior Member, IEEE*, Kamal Jain, and Ludo M. G. M. Tolhuizen, *Senior Member, IEEE*

Abstract—**The famous max-flow min-cut theorem states that a source node s can send information through a network (V, E) to a sink node t at a rate determined by the min-cut separating s and t. Recently, it has been shown that this rate can also be achieved for multicasting to several sinks provided that the intermediate nodes are allowed to re-encode the information they receive. We demonstrate examples of networks where the achievable rates obtained by coding at intermediate nodes are arbitrarily larger than if coding is not allowed. We give deterministic polynomial time algorithms and even faster randomized algorithms for designing linear codes for directed acyclic graphs with edges of unit capacity. We extend these algorithms to integer capacities and to codes that are tolerant to edge failures.**

Index Terms—**Communication networks, efficient algorithms, linear coding, multicasting rate maximization.**

I. Introduction

In this paper, we study the problem of multicasting: Consider a directed acyclic graph $G = (V, E)$, a *source* node $s \in V$, and a set of *sink* nodes $T \subseteq V$. The task is to send the same information from the source to all sinks at maximum data rate (bandwidth). Edges can reliably transport a single symbol of some alphabet per channel use. Typically, this symbol will be a vector of m bits viewed as an element of the finite field $\mathbb{F}$ with 2^m elements, with a single channel use being defined as the block transmission of an element of $\mathbb{F}$.

If there is only one sink $t \in T$, we have the well-known max-flow problem. The maximum data rate corresponds to the magnitude h of the maximum flow from s to t, which equals the capacity across the minimum cut separating s from t. Maximum flows can be found in polynomial time. (See, for example, [2] and [7].) Furthermore, a flow of magnitude h symbols per unit time can be decomposed into h edge disjoint paths so that multicasting can simply take place by sending one input symbol per unit time along each of these paths.

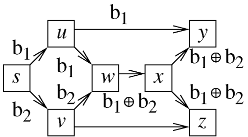

Fig. 1. An example where coding helps (see [1]).

The situation is more complicated for multiple sinks. For example, consider the graph in Fig. 1 [1]. There are flows of magnitude two from s to each sink in $T = \{y, z\}$. Yet there is no way to assign input symbols to flow paths such that each sink gets both symbols. Ahlswede *et al.* [1] have shown that coding within the network can solve this problem. In their example, assume we want to multicast the bits b_1 and b_2. Node w forwards the exclusive-or $b_1 \oplus b_2$ of the bits it receives. Now, sink y can find b_2 by computing $b_1 \oplus (b_1 \oplus b_2)$ and sink z can get b_1 from $b_2 \oplus (b_1 \oplus b_2)$. It turns out that this works for *all* multicast networks, i.e., the upper bound on the obtainable data rate imposed by the smallest maximum flow h from s to some sink $t \in T$ can be achieved using coding [1], [19]. This area of *network coding* is conceptually interesting because it brings together the seemingly unrelated concepts of coding and network flows.

A classical result of Edmonds [9] shows that network coding does not increase the achievable rate in the special case where one node is the source and all other nodes in the network are sinks ($V = \{s\} \cup T$). However, in networks that include nodes that are neither sources nor sinks, the rate achievable with coding can far exceed the rate achievable without coding (i.e., the rate achievable when nodes can only replicate and forward received symbols). In Section II, we give simple examples where the multicast rate achievable without coding must be a factor $\Omega(\log|V|)$ smaller than that achievable with coding. When coding is not allowed, even calculating the capacity is a computationally expensive problem: Maximizing the multicast rate without coding is at least as hard as the minimum directed Steiner tree problem [4], [15]. This implies that it is NP-hard to even approximate the maximum rate. Our main result is that although coding allows *higher* data rates than routing, finding optimal multicast coding schemes is possible in *polynomial* time.

Manuscript received July 18, 2003; revised November 30, 2004. Most of this work was performed when S. Jaggi was a summer intern at Microsoft Research, Redmond, WA, and when P. Sanders was at MPI Informatik. The work of S. Jaggi and M. Effros was supported in part by a Microsoft Fellowship, the National Science Foundation under Grant CCR-0220039, and a grant from the Lee Center for Advanced Networking at Caltech. The work of P. Sanders was supported in part by DFGunder Grant SA 933/1-1. The material in this paper was presented in part at the IEEE International Symposium on Information Theory, Yokohama, Japan, June/July 2003 and at the SPAA 2003, San Diego, CA, June 2003.

S. Jaggi and M. Effros are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: jaggi@caltech.edu; effros@caltech.edu).

P. Sanders is with the Fakultät für Informatik, Universität Karlsruhe, 76128 Karlsruhe, Germany (e-mail: sanders@ira.uka.de).

P. A. Chou and K. Jain are with Microsoft Research, Redmond, WA 98052-6399 USA (e-mail: pachou@microsoft.com; kamalj@microsoft.com).

S. Egner and L. M. G. M. Tolhuizen are with the Philips Research Laboratories, 5656 AA Eindhoven, The Netherlands (e-mail: sebastian.egner@philips.com; ludo.tolhuizen@philips.com).

Communicated by G. Sasaki, Associate Editor for Communication Networks.
Digital Object Identifier 10.1109/TIT.2005.847712

A. Overview

We continue the introduction with a short review of related work in Section I-B. Section II establishes that there can be large gaps between the multicast rates obtainable with and without coding. As our main results, Sections III and IV develop polynomial time algorithms for the centralized design of network multicast codes on unit capacity directed acyclic graphs. We then generalize our results to non-unit capacity edges and to centralized and distributed designs of robust codes that are tolerant to edge failures. We end with a discussion of the obtained results and possible future work. The Appendix summarizes the notation used in the paper, most of which is introduced in Section III.

B. Related Work

Ahlswede *et al.* [1] show that the source can multicast information to all sinks at a rate approaching h as the alphabet size approaches infinity. They also give the simple example from Fig. 1, which shows that without coding this rate is not always achievable.

Li *et al.* [19] show that linear coding can be used for multicasting with rate h and finite alphabet size. Our algorithms can be viewed as fast implementations of the approach by Li *et al.* The main difference is that Li *et al.* have to check a number of edge sets that is exponential in $|E|$ to verify that the coding coefficients chosen for a particular edge are correct. We reduce this to a single edge set per sink node by making explicit use of precomputed flows to each sink.

Koetter and Médard [16], [17] give an elegant algebraic characterization of the linear coding schemes that achieve the max-flow min-cut bound. They show that finite fields of size $\mathcal{O}(|T| \cdot h)$ are sufficient and give a polynomial time algorithm to verify a given linear network coding scheme. However, their algorithm for constructing coding schemes involves checking a multivariate polynomial identity with an exponential number of coefficients.

Ho *et al.* [12] present a polynomial expected time construction to the same problem, using a randomized approach. They give a tight lower bound on the probability that independent, random linear code design at every node achieves the max-flow min-cut bound. It turns out that the probability approaches one as $|T|/|\mathbb{F}|$ tends to zero, where $|\mathbb{F}|$ denotes the size of the finite field. They further note that this algorithm can be implemented in a distributed fashion, with a corresponding expected runtime which is logarithmic in $|\mathbb{F}|$. In another paper, Ho *et al.* [11] use algebraic techniques to bound the size of the finite field required by $|\mathbb{F}| \geq |T|$. In contrast, we use a centralized design of linear codes with field size $|\mathbb{F}| = \mathcal{O}(|T|)$ and construct a code scheme guaranteed to achieve the max-flow min-cut bound. Earlier versions of this algorithm were presented in [14], [22]. In our results on robust network codes, we also examine both centralized and distributed random design for codes with arbitrarily low probability of error.

Rasala-Lehman and Lehman [18] give a natural classification scheme for a large class of linear network coding problems. In this classification, a problem is either NP-hard or can be reduced to multicasting. This further underlines that a polynomial time algorithm for multicasting is a central result. They also obtain lower bounds on the minimum alphabet size required to do network coding, and show that finding the smallest alphabet size is NP-hard.

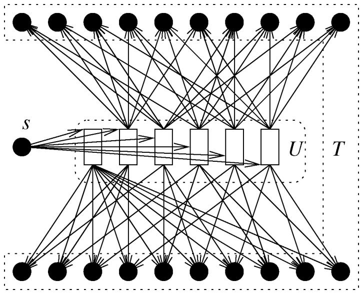

Fig. 2. An example where three symbols per time step can be delivered. Without coding, the best we can do is to send three symbols over every two time steps.

II. The Gap to Multicasting Without Coding

The following family of three-layer graphs gives examples where coding greatly increases the achievable rate: $\mathcal{G}_{a,h} = (V, E)$ with vertices $V = \{s\} \cup U \cup T$ where $U = \{1, \ldots, a\}, T = \{t_W \mid W \subseteq U, |W| = h\}$, and edges

$$E = \{(s, u) \mid u \in U\} \cup \{(u, t_W) \mid t_W \in T, u \in W\}.$$

That is, the source s constitutes the first layer, the a nodes in U constitute the second layer, and the $\binom{a}{h}$ nodes described by T constitute the third; each node in T is connected by unit capacity links to a distinct h-element subset W of U. Figs. 2 and 3 show $\mathcal{G}_{6,3}$ and $\mathcal{G}_{4,2}$, respectively.

Lemma 1: For any $a \geq h$, rate h is achievable on network $\mathcal{G}_{a,h}$.

Proof: A q-ary, $q \geq a$, maximum distance separable (MDS) code [20] is used to achieve this rate. The code has q^h codewords of block length a. The source maps the h input symbols to a unique codeword from the given codebook, sending each symbol of that codeword to a distinct node in U. The intermediate nodes do not code at all. □

Note: While in this example encoding operations only need to be carried out at the source node, in general, coding only at the source is not sufficient to guarantee capacity-achieving codes. Fig. 1 gives an example of a network where an interior node needs to perform a coding operation.

Theorem 2: There are unit capacity, directed, acyclic networks where multicasting with coding allows a factor $\Omega(\log |V|)$ larger rate than multicasting without coding.

Proof: Consider the network $\mathcal{G}_{2h,h}$. As stated before, the rate with coding is h. Without coding, the rate is less than 2. To see this, suppose that the source attempts to send $2n$ symbols $b_1, \ldots, b_{2n}$ to each of the sinks in T using n consecutive uncoded transmissions. Since each edge has unit capacity and $a = 2h$, the source can send at most $2hn$ symbols in total to the intermediate nodes. Thus, if U_i is the subset of intermediate nodes receiving b_i, then $\sum_{i=1}^{2n} |U_i| \leq 2hn$. This implies that there is an i for which $|U_i| \leq h$. Since $|U| = 2h$ and $|U \setminus U_i| \geq h$, there is at least one node $t_W \in T$ that receives all of its information from $W \subset U \setminus U_i$. Sink t_W fails to get symbol b_i. Finally, $|V|$ is strictly greater than the number of sink nodes, which equals $|T| = \binom{2h}{h}$, and we can bound $\binom{2h}{h}$ as

$$2^{2h} = \sum_{i=0}^{2h} \binom{2h}{i} \leq (2h+1)\binom{2h}{h}$$

thus, $|V| > |T| \geq 2^{2h}/(2h+1)$. □

III. Polynomial Time Coding

We now describe a polynomial time algorithm for centralized design of optimal network multicast codes. The codes are linear with symbols from a finite field $\mathbb{F}$. In practice, we will use a field of size $|\mathbb{F}| = 2^m$ so that the edges actually carry bits. Coding is done by forming linear combinations of the field elements reaching a node.

Since the detailed description of this key algorithm requires a lot of notation describing graphs, flows, symbols, and their interrelations, we begin with an informal outline that describes in words the underlying principles.

Our algorithm consists of two stages. In the first stage, a flow algorithm is run to find, for each sink $t \in T$, a set f^t of h edge-disjoint paths from the source s to t. Only the edges in the union of these flows are considered in the second stage of the algorithm.

The second stage is a greedy algorithm that visits each edge in turn and designs the linear coding employed for that edge. The order for visiting the edges is chosen so that the encoding for edge e is designed after the encodings for all edges leading to e. The goal in designing the encoding for $e = (v, w)$ is to choose a linear combination of the inputs to node v that ensures that all sinks that lie downstream from e obtain h linearly independent combinations of the original source symbols $b_1, \ldots, b_h$. For each sink t, the algorithm maintains a set $C_t \subset E$ and an $h \times h$ matrix B_t. The set C_t describes the most recently processed edge in each of the h edge-disjoint paths in f^t. The h columns of B_t correspond to the h edges in C_t, and the column for edge $c \in C_t$ describes the linear combination of $b_1, \ldots, b_h$ that traverses edge c. That is, if c carries $b_c(1)b_1 + \cdots + b_c(h)b_h$, then the corresponding column is $[b_c(1), \ldots, b_c(h)]^T$. The algorithm maintains the invariant that B_t is at every step invertible, thereby ensuring that the copy of $b_1, \ldots, b_h$ intended for sink t remains retrievable with every new code choice.

Theorem 3 summarizes the properties of the resulting algorithm. A formal algorithm description follows. (Recall that the notation is summarized in the Appendix.)

Theorem 3: Consider a unit capacity, directed, acyclic multigraph (V, E), and let h denote the minimum cut between the source s and any sink $t \in T$. The linear information flow (LIF) algorithms construct linear multicast codes over a finite field $\mathbb{F}$. In particular, the randomized LIF (RLIF) algorithm has expected running time $\mathcal{O}(|E| \cdot |T| \cdot h^2)$. Any finite field of size $|\mathbb{F}| \geq 2|T|$ can be used[1] to represent symbols sent along edges. The deterministic LIF (DLIF) algorithm has running time $\mathcal{O}(|E| \cdot |T| \cdot h \cdot (h + |T|))$. Any finite field of size $|\mathbb{F}| \geq |T|$ can be used to represent symbols sent along edges. The linear codes resulting from either of the LIF algorithms have the following properties.

- The source gets h information symbols as its input.
- A node v needs time $\mathcal{O}(\min(|\Gamma_I(v)|, |T|))$ to compute the symbol to be sent along a leaving edge, where $\Gamma_I(v)$ denotes the set of edges feeding into v. The source needs time $\mathcal{O}(h)$ for each edge.
- Each sink can reconstruct all h information symbols in time $\mathcal{O}(h^2)$.

To describe the algorithm, we need the following notation. $\Gamma_O(v)$ denotes the set of edges leaving node v; $\mathrm{start}(e)$ denotes the node at which edge e starts. For each edge e we define the $|\Gamma_I(\mathrm{start}(e))|$-length *local coding vector*

$$m_e : \Gamma_I(\mathrm{start}(e)) \rightarrow \mathbb{F}^{|\Gamma_I(\mathrm{start}(e))|}$$

as the vector which determines the linear combination of the symbols on the edges in $\Gamma_I(\mathrm{start}(e))$ to produce the symbol on edge e. That is, if $y(e)$ is the symbol carried by edge e, we have

$$y(e) = \sum_{p \in \Gamma_I(\mathrm{start}(e))} m_e(p) y(p).$$

Our task is to determine the coefficients $m_e(p)$ such that all sinks can reconstruct the original information from the symbols reaching them. We introduce h parallel edges $e_1, \ldots, e_h$ from some new node s' to s; these edges carry the input symbols for the source s.

We can characterize the effect of all the local coding vectors on edge e independently of a concrete input using *global coding vectors* $\boldsymbol{b}(e) \in \mathbb{F}^h$. The h-length vector $\boldsymbol{b}(e)$ represents the linear combination of the input symbols that generate $y(e)$. Thus, $\boldsymbol{b}(e_i) = [0^{i-1}, 1, 0^{h-i}]$ (an h-length vector with a 1 in the ith location) and

$$\boldsymbol{b}(e) = \sum_{p \in \Gamma_I(\mathrm{start}(e))} m_e(p) \boldsymbol{b}(p), \qquad \text{for } e \in E.$$

The $\boldsymbol{b}(e)$ vectors are well defined because the network is acyclic. Using elementary linear algebra, it can be seen that a linear coding scheme can be used for multicasting from s to T if and only if for all $t \in T$, the vectors $\{\boldsymbol{b}(p) : p \in \Gamma_I(t)\}$ span the vector space $\mathbb{F}^h$. Reconstructing the original information can then be achieved by solving a linear system of equations over h variables. The intuition is that a linear code mixes the information received from different edges but it does not lose essential information as long as there is a bijective mapping between the input and the data reaching the sink.

The challenge now is to find the local coding vectors efficiently, ideally using a small finite field that allows fast arithmetic. Our algorithm achieves this goal by making explicit use of a maximum flow algorithm. Initially, it computes s-t flows f^t of magnitude h for each $t \in T$ and decomposes these flows into h edge disjoint paths from s to t. If there were only a single sink node, our task would be simple now. We could route the ith input symbol along the ith edge disjoint path. If an edge e is

[1] A simple upper bound, not necessarily tight, for the failure probability of a single stage of the RLIF algorithm will be shown to be $|T|/|\mathbb{F}|$. We choose $|\mathbb{F}|/|T|$ as a constant greater than 1, thus making the expected number of trials independent of $|T|$. For convenience, we choose $|\mathbb{F}|/|T| \geq 2$.

on some flow path W from s to t, let $f^t_{\leftarrow}(e)$ denote the predecessor edge of edge e on path W. In our single-sink example, we could choose a nonzero coefficient for $m_e(f^t_{\leftarrow}(e))$ and zero for all other coefficients.

With multiple sinks, our approach is to superimpose multiple s-t flows. The algorithm steps through the nodes $u \in V$ in topological order. This ensures that the global coding vectors of all edges reaching u are known when the local coding vectors of the edges leaving u are being determined. The algorithm computes the coefficients of m_e for edges in $e \in \Gamma_O(u)$, one edge at a time. There might be multiple flow paths to different sinks through edge e. Let $T(e)$ denote the set of sinks using e in some flow f^t and let $P(e) = \{f^t_{\leftarrow}(e) : t \in T(e)\}$ denote the set of predecessor edges of e in the corresponding flow paths. Nonzero coefficients for m_e are only chosen for edges in $P(e)$. To ensure that all sinks can reconstruct the input, the algorithm of Li *et al.* [19] verifies that the global coding vector $\boldsymbol{b}(e)$ is linearly independent of an exponential number of sets of other global coding vectors. Our algorithm can simplify this task by exploiting the flows. It turns out that only $\mathcal{O}(|T|)$ edge sets need to be checked for each $e \in E$.

We maintain the invariant that for each sink $t \in T$ there is a set C_t of h edges such that the set of global coding vectors B_t, defined as $B_t = \{\boldsymbol{b}(c) : c \in C_t\}$, forms a basis of $\mathbb{F}^h$, i.e., the original input can be reconstructed from the information carried by the edges in C_t. The set C_t contains one edge from each path in f^t, namely, the edge whose global coding vector was defined most recently. Thus, when the computation completes, $C_t \subseteq \Gamma_I(t)$, and the invariant ensures that sink t gets all the information.

We initially establish the invariant by assigning the artificial input edges $\{e_1, \ldots, e_h\}$ with $\boldsymbol{b}(e_i) = [0^{i-1}, 1, 0^{h-i}]$ to C_t. When the linear combination m_e for a new edge e has been defined, we replace $f^t_{\leftarrow}(e)$ by e in all the C_t with $t \in T(e)$. Hence, to maintain the invariant, it is only necessary to check for all $t \in T(e)$ whether B_t still spans $\mathbb{F}^h$. Fig. 3 gives an example for the algorithm and its notation.

It remains to explain how to find coefficients for m_e that maintain the invariant. We argue that random coefficients for edges in $P(e)$ do the job if $|\mathbb{F}| \geq 2 \cdot |T|$. Indeed, Lemma 4 below shows that for a fixed sink, the failure probability is only $1/|\mathbb{F}|$. Summing over all sinks, we see that the failure probability is at most $|T|/|\mathbb{F}| \leq 1/2$.

Lemma 4: For any $e \in E$ and $t \in T(e)$, assume that B_t, which contains $\boldsymbol{b}(f^t_{\leftarrow}(e))$, is a basis of $\mathbb{F}^h$. Then with probability $1/|\mathbb{F}|$, a random choice of the coefficients in m_e with support in $P(e)$ fails to fulfill the property that $(B_t \setminus \{\boldsymbol{b}(f^t_{\leftarrow}(e))\}) \cup \{\boldsymbol{b}(e)\}$ is a basis of $\mathbb{F}^h$, where $\boldsymbol{b}(e)$ is the corresponding global coding vector $\sum_{p \in P(e)} m_e(p)\boldsymbol{b}(p)$.

Proof: If we fix the coefficients $m_e(p)$ for $p \in P(e) \setminus \{f^t_{\leftarrow}(e)\}$ then there is exactly one choice of $m_e(f^t_{\leftarrow}(e))$ for which $\boldsymbol{b}(e)$ is linearly dependent on $B_t \setminus \{\boldsymbol{b}(f^t_{\leftarrow}(e))\}$. To see this, observe that since B_t is a basis of $\mathbb{F}^h$

$$\boldsymbol{b}(e) = \sum_{g \in P(e)} m_e(g)\boldsymbol{b}(g)$$

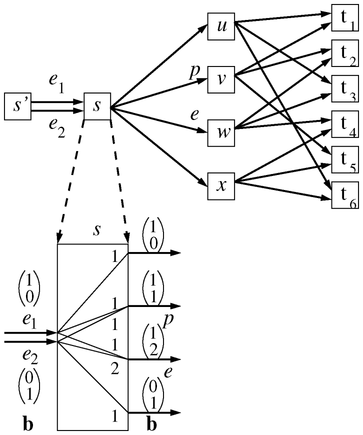

Fig. 3. An example for multicasting with linear coding from s to $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. We have $h = 2$. Assume that all the flows are decomposed into a topmost path and a bottommost path. The thin lines within s give nonzero coefficients for local coding vectors. The $\boldsymbol{b}$ vectors give the resulting global coding vectors. Let us assume that $\mathbb{F} = \mathrm{GF}(3)$ and that the edges leaving s are considered from top to bottom. If the global coding vectors for all other edges are fixed, then the only feasible linear combinations when we design m_e are $[m_e(e_1) = 1, m_e(e_2) = 2]$, or $[m_e(e_1) = 2, m_e(e_2) = 1]$. As further examples for our notation we have $\Gamma_I(t_2) = \{(v, t_2), (w, t_2)\}$, $\mathtt{start}(e) = s$, $T(e) = \{t_2, t_3, t_4\}$, $P(e) = \{e_1, e_2\}$, $f^{t_4}_{\leftarrow}(e) = e_1$, and $f^{t_3}_{\leftarrow}(e) = e_2$. Before m_e is fixed, $C_{t_2} = \{p, e_2\}$ and correspondingly $B_{t_2} = \{\binom{1}{1}, \binom{0}{1}\}$.

can be written in the form $(m_e(f^t_{\leftarrow}(e)) + \alpha)\boldsymbol{b}(f^t_{\leftarrow}(e)) + \boldsymbol{v}$ such that $\boldsymbol{v}$ and α only depend on the fixed coefficients and $\boldsymbol{v}$ is linearly dependent on $B_t \setminus \{\boldsymbol{b}(f^t_{\leftarrow}(e))\}$. Therefore, $\boldsymbol{b}(e)$ will be linearly dependent if and only if $m_e(f^t_{\leftarrow}(e)) = -\alpha$.

Hence, there are exactly $|\mathbb{F}|^{|P(e)|-1}$ local coding vectors that violate the property for sink t. Since there are $|\mathbb{F}|^{|P(e)|}$ choices for local coding vectors, the probability that a random choice violates the property is

$$|\mathbb{F}|^{|P(e)|-1}/|\mathbb{F}|^{|P(e)|} = 1/|\mathbb{F}|. \qquad \square$$

Lemma 4 yields a simple randomized algorithm for finding a *single* local coding vector. However, the construction fails with probability at most $1/2$, which is not sufficient to quickly find *all* coding vectors using a small field. Given the knowledge of the flows encoded in the C_t's and the invariant, we convert the preceding algorithm into one with a constant expected number of trials followed by $|T(e)|$ independence tests. This suffices to find a feasible local coding vector.[2]

What we have said so far already yields a LIF algorithm, running in polynomial expected time. In what follows, we further refine the algorithm to obtain a fast and more concrete implementation (Fig. 4) and a deterministic way of choosing the linear combinations m_e.

A. Testing Linear Independence Quickly

The mathematical basis for our refinement of the LIF algorithm is the following lemma, which uses the idea that testing

[2] This modification converts a "Monte Carlo-type" randomized algorithm—one that can fail—into a "Las Vegas-type" algorithm—one that always returns a correct answer but whose execution time is a random variable.

```
Function LIF($V$, $E$, $s$, $T$)
    $h := \min_{t\in T} \min\{|C| : C$ is $s$-$t$ cut$\}$                    -- $= \min_{t\in T}$ |max flow from $s$ to $t$|
    insert a new source $s'$ into $V$                                    -- help to establish the invariant
    insert $h$ parallel edges $\{e_1,\ldots,e_h\}$ from $s'$ to $s$ into $E$

    let $f^t$ denote a set of $h$ edge disjoint paths from $s$ to $t$        -- the chosen flow from $s$ to $t$
    (* We use the notation $f^t_{\leftarrow}(e)$, $T(e)$, and $P(e)$ to access the flows. *)
    let $\mathbb{F}$ be a finite field of a size satisfying the conditions of Theorem 3

    forall $i$: $\mathbf{b}(e_i) := [0^{i-1}, 1, 0^{h-i}]$                     -- the $i$-th unit vector of $\mathbb{F}^h$
    forall $t \in T$ do
        $C_t := \{e_1,\ldots,e_h\}$                                         -- $t$ is supplied through $C_t$
        $B_t := \{\mathbf{b}(e_1),\ldots,\mathbf{b}(e_h)\}$                             -- the coding vectors span $\mathbb{F}^h$
        forall $c \in C_t$: $\mathbf{a}_t(c) := \mathbf{b}(c)$                                   -- inverse vectors

    foreach vertex $v \in V \setminus \{s'\}$ in topological order do
        forall outgoing edges $e$ of $v$ do
            (* Invariant: $\forall t \in T : |C_t| = h$ and $\forall c, c' \in C_t : \mathbf{b}(c)\cdot\mathbf{a}_t(c') = \delta_{c,c'}$ *)
            choose a linear combination $\mathbf{b}(e) = \sum_{p\in P(e)} m_e(p)\mathbf{b}(p)$ such that       -- (*)
                $\forall t \in T(e) : (B_t \setminus \{\mathbf{b}(f^t_{\leftarrow}(e))\}) \cup \{\mathbf{b}(e)\}$ is linearly independent
            forall $t \in T(e)$ do
                $C'_t := (C_t \setminus \{f^t_{\leftarrow}(e)\}) \cup \{e\}$                       -- advance the set of edges $C_t$,
                $B'_t := (B_t \setminus \{\mathbf{b}(f^t_{\leftarrow}(e))\}) \cup \{\mathbf{b}(e)\}$              -- update $B_t$ correspondingly, and
                $\mathbf{a}'_t(e) := (\mathbf{b}(e)\cdot\mathbf{a}_t(f^t_{\leftarrow}(e)))^{-1}\,\mathbf{a}_t(f^t_{\leftarrow}(e))$            -- update $\mathbf{a}_t$ correspondingly
                forall $c \in C_t \setminus \{f^t_{\leftarrow}(e)\}$: $\mathbf{a}'_t(c) := \mathbf{a}_t(c) - (\mathbf{b}(e)\cdot\mathbf{a}_t(c))\mathbf{a}'_t(e)$
                $(C_t, B_t, \mathbf{a}_t) := (C'_t, B'_t, \mathbf{a}'_t)$

    return $(h, \{m_e : e \in E\}, \{(C_t, \mathbf{a}_t) : t \in T\}, \mathbb{F})$.
```

Fig. 4. LIF code design with fast testing of linear independence. Given a network (V, E), a source s and a set of sinks T, the algorithm constructs linear codes for intermediate nodes such that the rate from s to T is maximal.

whether a vector is linearly dependent on an $h-1$-dimensional subspace can be done by testing the dot-product of the vector with the vector representing the orthogonal complement of the subspace. Thus, testing linear dependence on an $h-1$-dimensional subspace can be reduced to a single scalar product of time complexity $\mathcal{O}(h)$. Here and in the sequel, $\delta_{a,b} = 1$ if $a = b$ and 0 otherwise.

Lemma 5: Consider a basis B of $\mathbb{F}^h$ and vectors $\boldsymbol{b} \in B$, $\boldsymbol{a} \in \mathbb{F}^h$ such that $\forall \boldsymbol{b}' \in B : \boldsymbol{b}' \cdot \boldsymbol{a} = \delta_{\boldsymbol{b},\boldsymbol{b}'}$. Then, any vector $\boldsymbol{x} \in \mathbb{F}^h$ is linearly dependent on $B \setminus \{\boldsymbol{b}\}$ if and only if $\boldsymbol{x} \cdot \boldsymbol{a} = 0$.

Proof: Let $\boldsymbol{x} = \sum_{\boldsymbol{b}'\in B} x(\boldsymbol{b}')\boldsymbol{b}'$ be the unique representation of $\boldsymbol{x}$ in the basis B. We get

$$\boldsymbol{x}\cdot\boldsymbol{a} = \sum_{\boldsymbol{b}'\in B} x(\boldsymbol{b}')\boldsymbol{b}'\cdot\boldsymbol{a} = \sum_{\boldsymbol{b}'\in B} x(\boldsymbol{b}')\delta_{\boldsymbol{b},\boldsymbol{b}'} = x(\boldsymbol{b}).$$

Now, $\boldsymbol{x}$ is linearly dependent on $B\setminus\{\boldsymbol{b}\}$ if and only if $x(\boldsymbol{b}) = 0$, i.e., if and only if $\boldsymbol{x}\cdot\boldsymbol{a} = 0$. □

The LIF algorithm given in Fig. 4 maintains vectors $\boldsymbol{a}_t(c)$ for each sink t and edge $c \in C_t$ that can be used to test linear dependence on $B_t \setminus \{\boldsymbol{b}(c)\}$. The invariant now becomes

$$\forall t \in T : |C_t| = h \text{ and } \forall c, c' \in C_t : \boldsymbol{b}(c)\cdot\boldsymbol{a}_t(c') = \delta_{c,c'}. \quad (1)$$

This invariant implies both the linear independence of B_t and the desired property of $\boldsymbol{a}_t$.

The algorithm in Fig. 4 implements the outline of the LIF algorithm given above. To prove correctness we have to verify the loop invariant.

Lemma 6: The loop invariant (1) holds for $(C_t, B_t, \boldsymbol{a}_t)$.

Proof: Proof by induction. Before the loop over the vertices, the loop invariant (1) is trivially satisfied. Now assume as the inductive hypothesis that loop invariant (1) holds for $(C_t, B_t, \boldsymbol{a}_t)$. We show that it holds for $(C'_t, B'_t, \boldsymbol{a}'_t)$.

In C_t, we replace edge $f^t_{\leftarrow}(e)$ by edge e, hence, the size of C'_t is the same as the size of C_t. According to the algorithm, $\boldsymbol{b}(e)$ is chosen such that $(B_t \setminus \{\boldsymbol{b}(f^t_{\leftarrow}(e))\}) \cup \{\boldsymbol{b}(e)\}$ is linearly independent. Hence, $\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(f^t_{\leftarrow}(e)) \neq 0$ by Lemma 5, and $\boldsymbol{a}'_t(e)$ is well defined. Finally, we verify $\boldsymbol{b}(c)\cdot\boldsymbol{a}'_t(c') = \delta_{c,c'}$ for all $c, c' \in C'_t$ by a short calculation

$$\begin{aligned}
\boldsymbol{b}(e)\cdot\boldsymbol{a}'_t(e) &= \boldsymbol{b}(e)\cdot\left(\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(f^t_{\leftarrow}(e))\right)^{-1}\boldsymbol{a}_t(f^t_{\leftarrow}(e)) = 1\\
\boldsymbol{b}(e)\cdot\boldsymbol{a}'_t(c) &= \boldsymbol{b}(e)\cdot\boldsymbol{a}_t(c) - (\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(c))(\boldsymbol{b}(e)\cdot\boldsymbol{a}'_t(e))\\
&= 0, \qquad \text{for } c \neq e\\
\boldsymbol{b}(c)\cdot\boldsymbol{a}'_t(e) &= \left(\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(f^t_{\leftarrow}(e))\right)^{-1}(\boldsymbol{b}(c)\cdot\boldsymbol{a}_t(f^t_{\leftarrow}(e)))\\
&= 0, \qquad \text{for } c \neq e\\
\boldsymbol{b}(c)\cdot\boldsymbol{a}'_t(c') &= \boldsymbol{b}(c)\cdot(\boldsymbol{a}_t(c') - (\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(c'))\boldsymbol{a}'_t(e))\\
&= \boldsymbol{b}(c)\cdot\boldsymbol{a}_t(c') - (\boldsymbol{b}(e)\cdot\boldsymbol{a}_t(c'))(\boldsymbol{b}(c)\cdot\boldsymbol{a}'_t(e))\\
&= \boldsymbol{b}(c)\cdot\boldsymbol{a}_t(c') = \delta_{c,c'}, \qquad \text{for } c, c' \neq e.
\end{aligned}$$
□

Remark: If the vectors in B_t are arranged as the rows of a matrix $\boldsymbol{B}_t$ and the columns $\boldsymbol{a}_t$ are correspondingly arranged as a matrix $\boldsymbol{A}_t$, then the invariant is equivalent to $\boldsymbol{A}_t = \boldsymbol{B}_t^{-1}$. This relation is also useful as it leads to a low-complexity decoding algorithm, as will be explained at the end of this section. In this notation, the method of updating the inverse vectors $\boldsymbol{a}_t$ in the LIF algorithm is a special case of the Sherman–Morrison formula [21, Sec.2.7].

What we have said so far suffices to establish the complexity of the randomized variant of LIF:

Lemma 7: If line **(*)** in Fig. 4 is implemented by choosing random m_e with support in $P(e)$ until the condition "$\forall t \in T(e) : (B_t \setminus \{\boldsymbol{b}(f^t_{\leftarrow}(e))\}) \cup \{\boldsymbol{b}(e)\}$ is linearly independent" is satisfied, then the algorithm can be implemented to run in expected time $\mathcal{O}(|E|\cdot|T|\cdot h^2)$ and the returned information allows decoding in time $\mathcal{O}(h^2)$ at each sink.

Proof: Using a single graph traversal, we can find a flow-augmenting path from s to $t \in T$ in time $\mathcal{O}(|E|)$ [2]. We apply this routine cycling through the sinks until, for some sink, no augmenting paths are left. We can find h augmenting paths for all sinks in time $\mathcal{O}(|T| \cdot |E| \cdot h)$.[3]

The algorithm works correctly over any finite field of size $|\mathbb{F}| \geq 2|T|$. In order to perform finite-field operations efficiently, we can create a lookup table of $|\mathbb{F}|$ entries for successors of elements (Conway's "Zech-logarithm" [8], [13]). Using this table, any arithmetic operation in $\mathbb{F}$ can be computed in constant time.[4]

Initializing C_t, B_t, and $\boldsymbol{a}_t(c)$ takes time $\mathcal{O}\big(|T| \cdot h^2\big)$. The two main loops collectively iterate over all edges so that there is a total number of $|E|$ iterations. Computing $P(e)$ takes time $\mathcal{O}(|T|)$ if the flows f^t maintain pointers to the predecessors of edges in the path decomposition of f^t.

Finding a random local coding vector m_e takes time $\mathcal{O}(|P(e)|) = \mathcal{O}(|T|)$. Computing $\boldsymbol{b}(e)$ and testing linear independence using the vectors $\boldsymbol{a}_t(c)$ takes time $\mathcal{O}(h \cdot |T|)$. Since the success probability is constant, the expected cost for finding a linearly independent m_e is $\mathcal{O}(1) \cdot \mathcal{O}(h \cdot |T|) = \mathcal{O}(h \cdot |T|)$. Computing $\boldsymbol{a}'_t(c)$ for all t and all $c \in C_t$ takes time $\mathcal{O}\big(|T| \cdot h^2\big)$.

Combining all the parts, we get the claimed expected time bound of $\mathcal{O}\big(|E| \cdot |T| \cdot h^2\big)$. Sink $t \in T$ can reconstruct the vector of input symbols $\boldsymbol{x}$ at s by computing $\boldsymbol{x} = \sum_{c \in C_t} \boldsymbol{a}_t(c) y(c)$, where $y(c)$ denotes the symbol received over edge $c \in C_t$. This decoding algorithm works since $\sum_{c \in C_t} \boldsymbol{a}_t(c) y(c)$ can be written as a matrix product between $\boldsymbol{A}_t$ and the vector $y(c)$. But by our invariant $\boldsymbol{A}_t = \boldsymbol{B}_t^{-1}$, and as shown earlier $y(c)$ equals $\boldsymbol{B}_t \boldsymbol{x}$. This decoding operation takes time $\mathcal{O}\big(h^2\big)$. □

B. Deterministic Implementation

We now explain how the LIF algorithm in Fig. 4 can be implemented deterministically. We develop a deterministic method for choosing the local coding vectors in step (*) using the following lemma which is formulated as a pure linear algebra problem without using graph-theoretic concepts.

Lemma 8: Let $n \leq |\mathbb{F}|$. Consider pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{F}^h \times \mathbb{F}^h$ with $\boldsymbol{x}_i \cdot \boldsymbol{y}_i \neq 0$ for $1 \leq i \leq n$. There exists a linear combination $\boldsymbol{u}$ of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ such that $\boldsymbol{u} \cdot \boldsymbol{y}_i \neq 0$ for $1 \leq i \leq n$. Such a vector $\boldsymbol{u}$ can be found in time $\mathcal{O}\big(n^2 h\big)$.

Proof: We inductively construct $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$ such that for all i, j with $1 \leq j \leq i \leq n$, we have that $\boldsymbol{u}_i \cdot \boldsymbol{y}_j \neq 0$. We choose $\boldsymbol{u}_1 = \boldsymbol{x}_1$. Now, let $1 \leq i \leq n-1$.

If $\boldsymbol{u}_i \cdot \boldsymbol{y}_{i+1} \neq 0$, we set $\boldsymbol{u}_{i+1} := \boldsymbol{u}_i$. Otherwise, note that for each $\alpha \in \mathbb{F}$, $(\alpha \boldsymbol{u}_i + \boldsymbol{x}_{i+1}) \cdot \boldsymbol{y}_{i+1} \neq 0$. For each j, $1 \leq j \leq i$, we have that $(\alpha \boldsymbol{u}_i + \boldsymbol{x}_{i+1}) \cdot \boldsymbol{y}_j = 0$ if and only if

$$\alpha = \alpha_j := -(\boldsymbol{x}_{i+1} \cdot \boldsymbol{y}_j)/(\boldsymbol{u}_i \cdot \boldsymbol{y}_j).$$

Now, let H be a set of n distinct field elements. As $|H| > i$, the set $H' := H \setminus \{\alpha_j \mid 1 \leq j \leq i\}$ is nonempty. We choose some element $\alpha \in H'$ and define $\boldsymbol{u}_{i+1} := \alpha \boldsymbol{u}_i + \boldsymbol{x}_{i+1}$. By construction, $\boldsymbol{u}_{i+1} \cdot \boldsymbol{y}_j \neq 0$ for $1 \leq j \leq i+1$.

Computation of an inner product takes times $\mathcal{O}(h)$. Each α_j can be computed in time $\mathcal{O}(h)$. Since $|H| < n$, H can be represented in such a way that initialization and finding an element can be done in time $\mathcal{O}(n)$ and removing an element can be done in constant time. Hence, finding a single coefficient takes time $\mathcal{O}(nh)$ and finding $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$ takes time $\mathcal{O}\big(n^2 h\big)$. □

Lemma 8 can be used to find the linear combination m_e in the LIF algorithm: Apply Lemma 8 to

$$\big\{(\boldsymbol{b}(f^t_{\leftarrow}(e)), \boldsymbol{a}_t(f^t_{\leftarrow}(e))) : t \in T(e)\big\}$$

i.e., let $\boldsymbol{b}(e) = \boldsymbol{u}$ denote a vector with $\boldsymbol{b}(e) \cdot \boldsymbol{a}_t(f^t_{\leftarrow}(e)) \neq 0$.

The deterministic part of Theorem 3 can now be proven analogously to the proof of Lemma 7. We just replace the expected time $\mathcal{O}(|T| \cdot h)$ needed by the randomized routine for choosing m_e by the time $\mathcal{O}\big(|T(e)|^2 h\big) = \mathcal{O}\big(|T|^2 h\big)$ needed to apply Lemma 8. We obtain a total execution time of

$$\mathcal{O}\big(|E|(|T| \cdot h^2 + |T|^2 h)\big) = \mathcal{O}(|E| \cdot |T| \cdot h(h + |T|)).$$

Note that the restriction $|\mathbb{F}| > n$ in Lemma 8 is the reason that $|\mathbb{F}| > |T|$ in Theorem 3.

IV. Faster Construction

We now outline an alternative algorithm for constructing linear network coding schemes. The algorithm is faster at the cost of using larger finite fields and hence possibly more expensive coding and decoding. Perhaps more importantly, this approach illustrates interesting connections between previous results and the present paper.

Theorem 9: Linear network coding schemes using finite fields of size $|\mathbb{F}| \geq 2 \cdot |E| \cdot |T|$ and achieving rate h can be found in expected time $\mathcal{O}(|E| \cdot |T| \cdot h + |T| \cdot \mathcal{M}(h))$, where $\mathcal{M}(h) = \mathcal{O}\big(h^{2.376}\big)$ denotes the time required for performing $h \times h$ matrix multiplications.

Proof: (Outline) First find flows f^t decomposed into paths as before (time $\mathcal{O}(|E| \cdot |T| \cdot h)$). Then pick independent random local coding vectors m_e for all edges simultaneously. Compute all global coding vectors $\boldsymbol{b}(e)$ (time $\mathcal{O}\big(\sum_{e \in E} |P(e)| \cdot h\big) = \mathcal{O}(|E| \cdot |T| \cdot h)$). For each sink $t \in T$, let B_t denote the set of global coding vectors corresponding to edges ending a path in f^t. Check whether all the B_t span $\mathbb{F}^h$ (time $\mathcal{O}(|T| \cdot \mathcal{M}(h))$ using matrix inversion based on fast matrix multiplication [6]). If any of the tests fails, repeat.

Using Lemma 4 and the analysis of the algorithm presented in Fig. 4 in Section III it can be seen that the success probability is at least $1/2$: The algorithm of Fig. 4 would perform $\sum_{e \in E} |T(e)|$ independence tests, each of which would go wrong with probability $1/|\mathbb{F}|$. Hence, if we omit the tests, the failure probability is at most $\sum_{e \in E} |T(e)|/|\mathbb{F}| \leq 1/2$. The expected number of repetitions will be constant. □

This algorithm is quite similar to the one by Ahlswede *et al.* [1], who choose arbitrary (possibly nonlinear) functions over a fixed block length for the local encoding operations. The main difference is that we choose random linear local coding functions, which leads to a practical design of low-complexity codes.

[3] We can also use Dinitz' algorithm [7] to find many paths in time $\mathcal{O}(|E|)$. For large h this also yields improved asymptotic time bounds for the flow computation part [10].

[4] If the table is considered too large, one can resort to the polynomial representation of field elements. In this case, no table is needed at the cost of additional factors in running time that are polylogarithmic in $|T|$.

Even the analysis of Ahlswede *et al.* could be adapted. Their analysis goes through if the arbitrary random functions are replaced by a random choice from a universal family of hash functions.[5] It is well known that random linear mappings form such a universal family. Exploiting the special structure of linear functions, this idea can be further developed into a polynomial time randomized algorithm achieving rate $(1-\epsilon)h$ for any constant $\epsilon > 0$. However, the analysis given here yields stronger results (rate h, exponentially smaller finite fields) because we can reduce the exponential number of cuts considered in [1] to a polynomial number of edge sets that need to carry all the information.

Another interesting observation is that Koetter and Médard [16], [17] arrive at a similar requirement for $|\mathbb{F}|$ as Theorem 9 using quite different algebraic arguments.

V. Handling Integer Edge Capacities

We now generalize from acyclic graphs with unit edge capacities to arbitrary integer edge capacities. We can compute h in a time polynomial in $|V|$ and $|E|$ and subsequently replace each capacitated edge e by $\min(h, C(e))$ parallel edges of unit capacity. Section III immediately yields algorithms with running times polynomial in $|V|$, $|E|$, and h.

In the unit capacity case, since each unit capacity link has to be defined separately, h can only grow linearly in the number of bits needed to define the network. However, if the edge capacities are large integers, h can be exponential in the input size of the graph. From a complexity point of view, this is not satisfactory. Hence, the question arises how to handle graphs with very large h. Again, Section III (almost) suffices to solve this problem:

Theorem 10: Let h denote the maximum flow in a network $\mathcal{G} = (V, E)$ with edge capacities $C : E \rightarrow \mathbb{N}$. For any $\epsilon > 0$ such that $\epsilon h \in \mathbb{N}$,[6] linear network coding schemes can be found in time polynomial in $|V|$, $|E|$, and $1/\epsilon$ so that $(1-\epsilon)h$ symbols per time step can be communicated.

Proof: In a preprocessing step, find the maximum flow f^t from s to each sink $t \in T$. Let $f^t(e)$ denote the number of symbols transported per time unit by flow f^t over edge e. Reduce $C(e)$ to $\max_{t \in T} f^t(e)$. Note that no edge capacity exceeds h now. Let $w \leq |E|$ denote the maximum number of edge disjoint paths needed to realize any of the flows f^t. Now build a unit capacity network $\mathcal{G}' = (V, E')$, where each edge $e \in E$ corresponds to $C'(e) := \lfloor w \cdot C(e)/h\epsilon \rfloor$ parallel unit capacity edges in E'. Then find a multicast coding scheme for $\mathcal{G}'$. This is possible in polynomial time since there are at most $w \cdot |E|/\epsilon$ edges in the unit capacity problem.

For coding and decoding in the capacitated instance, edge e is split into $C'(e)$ edges each with capacity $h\epsilon/w$. Each such edge transmits $h\epsilon$ symbols every w time steps using the encoding prescribed for the corresponding unit capacity edge. Thus, on each s—t path in any flow f^t, the unused capacity is at most $h\epsilon/w$ (due to the rounding-off caused by the $\lfloor \cdot \rfloor$ function) or $h\epsilon$ on all paths on one flow. Hence, the total used capacity is at least $h - h\epsilon = (1-\epsilon)h$. □

VI. Tolerating Edge Failures

From a practical point of view, the importance of network coding may come as much from an ability to increase the robustness of network communication as from an ability to increase throughput. In this section, we address the problem of constructing robust network codes. There are many possible models of robust network coding (e.g., [3]). We consider a model similar to that of Koetter and Médard [16], [17]. Koetter and Médard define a link failure pattern, in essence, as a subset $F \subseteq E$ such that every edge $e \in F$ fails sometime after code design. A failed edge transmits only the zero symbol, and such zero symbols are processed identically to zero symbols transmitted by functioning edges. (In practice, a node connected to a failed edge would detect the failure and ignore the symbols from that edge in forming its linear combinations. The coefficients for the functioning edges would not change as a result of the failure.) Koetter and Médard demonstrate that for any set $\mathcal{F}$ of failure patterns F not reducing the network capacity below some desired transmission rate $k \leq h$, there exists a single linear code where the source and interior node encoding schemes remain unchanged for all $F \in \mathcal{F}$, but each sink node can decode all k symbols if it knows the failure pattern, provided the field size exceeds $k|T||\mathcal{F}|$. Knowledge of the failure pattern allows the sink to compute the appropriate decoding matrix.[7] In [5], [12] it is shown that the appropriate decoding matrix can be directly communicated to each sink node by transmitting over each functioning edge e the global coding vector for edge e computed under failure pattern F. This is a one-time transmission, and as such has low overhead. In this paper, we improve on the field size bound slightly and provide complexity bounds. The following theorem generalizes Theorem 3.

Theorem 11: Let k be a desired source transmission rate, and let $\mathcal{F}$ be a set of edge failure patterns that do not reduce the network capacity below k. A robust linear network code achieving rate k under every edge failure pattern in $\mathcal{F}$ using a finite field of size $|\mathbb{F}| \geq 2|T||\mathcal{F}|$ can be found in expected time $\mathcal{O}\big(|\mathcal{F}| \cdot |E| \cdot (|T| \cdot k^2 + \min\{I, |T||\mathcal{F}|\} \cdot k)\big)$, where I denotes the maximum in-degree of a node.

Proof: For each failure pattern $F \in \mathcal{F}$, we find a flow of magnitude k from s to each sink $t \in T$. We reduce the graph by considering only the edges that occur in a flow for at least one failure pattern. For each failure pattern, an edge may be on at most $|T|$ paths from s to a sink; therefore, in total, an edge may be on at most $|T||\mathcal{F}|$ paths from s to a sink. Alternately, the number of paths from s to a sink passing through an edge equals at most I. As a result, the symbol on an edge may be a linear combination of as many as $\min\{I, |T||\mathcal{F}|\}$ symbols. We employ the algorithm of Fig. 5, which in essence runs the RLIF algorithm in parallel for each failure pattern. (A version based on the DLIF algorithm could also be constructed with

[5] A family $\mathcal{H} \subseteq A^B$ of functions from B to A is called *universal* if $\forall x, y \in B : \boldsymbol{Pr}[f(x) = f(y)] = 1/|A|$ for randomly chosen $f \in \mathcal{H}$.

[6] The requirement $\epsilon h \in \mathbb{N}$ avoids trivial rounding issues. By appropriately choosing our unit of time, we are quite flexible in choosing ϵ.

[7] Note the similarity with error-correcting codes: with an $[n, k]$ MDS code, $n - k$ errors can be corrected if their locations are known (erasure decoding); without this knowledge, only $\lfloor (n-k)/2 \rfloor$ errors can be corrected.

```
Function LIF($V$, $E$, $s$, $T$, $k$, $\mathcal{F}$)        -- Assume $E$ and each $F \in \mathcal{F}$ are in topological order
    (* Initialization. *)
    Introduce artificial node $s'$ and edges $e_1, \ldots, e_k$ from $s'$ to $s$        -- $\mathcal{O}(k)$
    foreach failure pattern $F \in \mathcal{F}$ do
        Precompute reduced graph $(V, E \setminus F)$        -- $\mathcal{O}(|E|)$
        foreach artificial edge $e_i$ do
            Initialize global coding vector $\mathbf{b}^F(e_i) = [0^{i-1}, 1, 0^{k-i}]$;        -- $\mathcal{O}(k)$
        foreach sink $t \in T$ do
            Initialize basis $B_t^F = \{\mathbf{b}^F(e_i)\}_{i=1}^{k}$        -- $\mathcal{O}(k^2)$
            Establish flow $f^{t,F}$ of strength $k$ from $s$ to $t$ in $(V, E \setminus F)$        -- $\mathcal{O}(|E|k)$

    (* Main loop. *)
    foreach edge $e$ in topological order do
        Choose local encoding vector $m_e$ with support in $P(e)$ at random        -- $\mathcal{O}(I)$
        foreach failure pattern $F \in \mathcal{F}$ do
            if $e \in F$ then        -- $\mathcal{O}(1)$
                Set $\mathbf{b}^F(e) = 0$        -- $\mathcal{O}(k)$
            else set $\mathbf{b}^F(e) = \sum_{e' \in \Gamma_I(\text{start}(e))} m_e(e')\mathbf{b}^F(e')$        -- $\mathcal{O}((\min\{I, |T||\mathbb{F}|\})k)$
            foreach sink $t \in T$ do
                if $e \in f^{t,F}$ then        -- $\mathcal{O}(1)$
                    Set $\hat{B}_t^F = (B_t^F \setminus \{\mathbf{b}^F(f_{\leftarrow}^{t,F}(e))\}) \cup \{\mathbf{b}^F(e)\}$        -- $\mathcal{O}(k^2)$
                    if $\hat{B}_t^F$ fails to be linearly independent then        -- $\mathcal{O}(\mathcal{M}(k))$
                        Go to Choose
        foreach failure pattern $F \in \mathcal{F}$ do
            foreach sink $t \in T$ do
                $B_t^F = \hat{B}_t^F$        -- $\mathcal{O}(k^2)$

    return $(\{m_e : e \in E\})$.
```

Fig. 5. Robust linear information flow. Given a network (V, E), a source s, a set of sinks T, a transmission rate k, and a set of failure patterns $\mathcal{F}$ not reducing the capacity below k, the algorithm constructs linear codes for intermediate nodes robust to all failure patterns in $\mathcal{F}$.

corresponding results.) The algorithm maintains the invariant that the sets of vectors B_t^F are linearly independent for each failure pattern $F \in \mathcal{F}$ and sink $t \in T$. Hence, every sink node can decode under every failure pattern. The linear independence test in the algorithm fails with probability $1/|\mathbb{F}|$ by Lemma 4. Hence, by the union bound, the linear independence test fails for some failure pattern $F \in \mathcal{F}$ and some sink $t \in T$ with probability at most $\delta = |T||\mathcal{F}|/|\mathbb{F}|$. Therefore, the expected number of times a local encoding vector is chosen for each edge e is at most $\sum_{i=1}^{\infty} i\delta^{i-1}(1-\delta) = 1/(1-\delta)$, which in turn is at most 2 if $|\mathbb{F}| \geq 2|T||\mathcal{F}|$. The complexity of the initialization is

$$\mathcal{O}\big(k + |\mathcal{F}|(|E| + k^2 + |T|(k^2 + |E|k))\big)$$

while the complexity of the main loop is

$$\mathcal{O}\big(|E|(I + |\mathcal{F}|(Ik + |T|(k^2 + \mathcal{M}(k))) + |\mathcal{F}||T|k^2)\big).$$

Combining terms, the overall complexity is

$$\mathcal{O}(|\mathcal{F}||E|(|T|\mathcal{M}(k) + Ik)).$$

Using the fast linear independence test techniques of Section III-B, the factor $\mathcal{M}(k)$ can be replaced by k^2. □

We can avoid most of the complexity if we are only interested in network codes that are robust with high probability (rather than with certainty), as the following theorem shows.

Theorem 12: Let k be a desired source transmission rate, and let $\mathcal{F}$ be a set of edge failure patterns not reducing the network capacity below k. A linear network code whose local encoding vector coefficients are generated at random independently and uniformly over a finite field $\mathbb{F}$ will tolerate all edge failure patterns in $\mathcal{F}$ (i.e., will achieve rate k) with probability at least $1-\delta$ if $|\mathbb{F}| \geq |E||T||\mathcal{F}|/\delta$, and will tolerate any particular failure pattern in $\mathcal{F}$ (and hence will tolerate a random failure pattern drawn from $\mathcal{F}$) with probability at least $1-\delta$ if $|\mathbb{F}| \geq |E||T|/\delta$.

Proof: First pick independent random local coding vectors for all edges in the graph simultaneously. Then pick a failure pattern in $\mathcal{F}$. For this failure pattern, compute the global coding vectors for all edges in the graph, find a flow of magnitude k from the source to each sink in T, and test that the global coding vectors for the k edges in the flow in any cut to any sink are linearly independent. This test fails with probability at most $|T|/|\mathbb{F}|$ by the proof of Theorem 9. But for each sink to be able to decode the message, one needs to consider linear independence only on at most $|E|$ such cuts. By the union bound, the probability that the independence test fails for any of $|T|$ sinks in any of the $|E|$ cuts in any of the $|\mathcal{F}|$ failure patterns is at most δ if $|\mathbb{F}| \geq |E||T||\mathcal{F}|/\delta$. □

As pointed out in [5], [12], the local coding vectors can be chosen in a distributed manner and knowledge of the global coding vectors can be passed downstream with asymptotically negligible rate overhead.

VII. Discussion

In this paper, we present polynomial time algorithms for the design of maximum rate linear multicast network codes by combining techniques from linear algebra, network flows, and (de)randomization. The existence of such an algorithm is remarkable because the maximal rate without coding can be much smaller and finding the routing solution that achieves that maximum is NP-hard. The resulting codes operate over finite fields that are much smaller than those of previous constructions. We also obtain results for fault-tolerant multicast network coding.

Linear network codes are designed to work over finite fields of size 2^m. Any symbol from such a field can be represented as an m-bit binary string. Rasala-Lehman and Lehman [18] show that there exist networks with $|T|$ nodes for which the minimum required alphabet size for any capacity-achieving network multicast code is $\Theta(\sqrt{|T|})$. Hence, in general, finite fields of arbitrarily large sizes are required for network coding, and for these worst case networks, the symbol size of our linear multicast codes (measured in bits) is at most twice the minimum required symbol size. Rasala-Lehman and Lehman [18] also show that finding the minimal alphabet size for network coding on a given graph is NP-hard.

Many interesting problems still remain open. For example, from a complexity point of view it would be interesting to replace the approximation scheme for capacitated edges in Section V by a fully polynomial time[8] exact algorithm perhaps using some kind of "scaling" approach.

Perhaps the most challenging open questions involve the more general network coding problem where multiple senders send different messages to multiple sets of receivers. Although no further tractable problem classes exist within the classification scheme by Rasala-Lehman and Lehman [18], it might still be possible to find polynomial time approximation algorithms for NP-hard cases that outperform the best tractable algorithms without coding.

Appendix
Notation

$[0^{i-1}, 1, 0^{h-i}]$:	a length-h vector with a 1 in the ith location and 0 otherwise.
$\boldsymbol{a}_t(c)$:	a vector with the property that $\boldsymbol{x} \cdot \boldsymbol{a}_t(c) \neq 0$ if and only if $\boldsymbol{x}$ is linearly independent of $B_t \setminus \{\boldsymbol{b}(c)\}$ for some $c \in C_t$.
$\boldsymbol{b}(e) \in \mathbb{F}^h$:	global coding vector for edge $e \in E$.
B_t:	the set of global coding vectors for sink t, $B_t = \{\boldsymbol{b}(c) : c \in C_t\}$.
C_t:	h edges on edge-disjoint paths from s to t.
$C(e)$:	the capacity of edge $e \in E$.
$\delta_{u,v}$:	1 if $u = v$ and 0 otherwise.
E:	the set of edges.
$e \in E$:	an edge.
$e_1, \ldots, e_h$:	input edges connecting s' with s.
$\epsilon > 0$:	a small constant.
$\mathbb{F}$:	the finite field used
f^t:	a flow of magnitude h from s to t represented by h edge disjoint paths.
$f^t_{\leftarrow}(e)$:	predecessor edge of e on a path from s to $t \in T$.
$\mathcal{G} = (V, E)$:	the graph.
$\Gamma_I(v)$:	the set of edges entering node $v \in V$.
$\Gamma_O(v)$:	the set of edges leaving node $v \in V$.
h:	the smallest maximum flow from s to some sink $t \in T$.
m:	the block length of the linear codes.
m_e:	the local coding vector for edge $e \in E$, i.e., $m_e(e')$ is the coefficient multiplied with $y(e')$ to contribute to $y(e)$.
$P(e)$:	the predecessor edges of e in some flow path $\{f^t_{\leftarrow}(e) : t \in T(e)\}$.
s:	source node.
s':	dummy source node.
$\text{start}(e)$:	the node where edge $e \in E$ departs.
$T \subseteq V$:	the set of sink nodes.
$T(e) \subseteq T$:	the sinks supplied through edge $e \in E$, i.e., $T(e) = \{t \in T : e \in f^t\}$.
$t \in T$:	a sink node.
V:	the set of nodes.
$v \in V$:	a node.
$y(e)$:	the symbol carried by edge $e \in E$.

[8] A *fully polynomial* time algorithm runs in time polynomial in the number of bits needed to encode the input.

Acknowledgment

The authors would like to thank Rudolf Ahlswede, Ning Cai, Tracy Ho, Irit Katriel, Joerg Kliewer, Piotr Krysta, Anand Srivastav, Mandyam Aji Srinivas, and Berthold Vöcking for fruitful discussions, and the anonymous reviewers for many valuable comments concerning the presentation.

References

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

[2] R. K. Ahuja, R. L. Magnanti, and J. B. Orlin, *Network Flows*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[3] N. Cai and R. W. Yeung, "Network coding and error correction," in *Proc. Information Theory Workshop (ITW)*, Bangalore, India, Oct. 2002, pp. 119–122.

[4] B. Carr and S. Vempala, "Randomized meta-rounding," in *Proc. 32nd ACM Symp. Theory of Computing (STOC)*, Portland, OR, May 2000, pp. 58–62.

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

[6] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *J. Symb. Comput.*, vol. 9, pp. 251–280, 1990.

[7] E. A. Dinic, "Algorithm for solution of a problem of maximum flow," *Sov. Math.—Dokl.*, vol. 11, pp. 1277–1280, 1970.

[8] J. Dumas, T. Gautier, and C. Pernet, "Finite field linear algebra subroutines," in *Proc. Int. Symp. Symbolic and Algebraic Computation(ISSAC)*, Lille, France, Jul. 2002, pp. 63–74.

[9] J. Edmonds, "Minimum partition of a matroid into independent sets," *J. Res. Nat. Bur. Stand. Sect.*, vol. 869, pp. 67–72, 1965.

[10] S. Even and E. Tarjan, "Network flow and testing graph connectivity," *SIAM J. Comput.*, vol. 4, pp. 507–518, 1975.

[11] T. Ho, D. Karger, R. Koetter, and M. Médard, "Network coding from a network flow perspective," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Yokohama, Japan, Jun./Jul. 2003, p. 441.

[12] T. Ho, R. Koetter, M. Médard, D. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Yokohama, Japan, Jun./Jul. 2003, p. 442.

[13] K. Imamura, "A method for computing addition tables in $GF(p^n)$," *IEEE Trans. Inf. Theory*, vol. IT-26, no. 3, pp. 367–369, May 1980.

[14] S. Jaggi, P. A. Chou, and K. Jain, "Low complexity algebraic multicast network codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Yokohama, Japan, Jun./Jul. 2003, p. 368.

[15] K. Jain, M. Mahdian, and M. R. Salavatipour, "Packing Steiner trees," in *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Baltimore, MD, Jan. 2003.

[16] R. Koetter and M. Médard, "Beyond routing: An algebraic approach to network coding," in *Proc. 21st Annu. Joint Conf. IEEE Computer and Communications Societies (INFOCOMM)*, vol. 1, New York, Jun. 2002, pp. 122–130.

[17] ——, “An algebraic approach to network coding,” *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[18] A. Rasala-Lehman and E. Lehman, “Complexity classification of network information flow problems,” manuscript, Apr. 2003.

[19] S.-Y. R. Li, R. W. Yeung, and N. Cai, “Linear network coding,” *IEEE Trans. Inf. Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[20] F. J. McWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1977.

[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[22] P. Sanders, S. Egner, and L. Tolhuizen, “Polynomial time algorithms for network information flow,” in *Proc. 15th ACM Symp. Parallel Algorithms and Architectures (SPAA)*, San Diego, CA, Jun. 2003, pp. 286–294.