Polynomial Time Algorithms for Network Information Flow

Peter Sanders*
MPI Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany
sanders@mpi-sb.mpg.de

Sebastian Egner
Philips Research Laboratories
Prof. Holstlaan 4
5656 AA Eindhoven
The Netherlands

Ludo Tolhuizen
Philips Research Laboratories
Prof. Holstlaan 4
5656 AA Eindhoven
The Netherlands

[sebastian.egner,ludo.tolhuizen]@philips.com

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 data 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 reencode the information they receive. In contrast, we present graphs where without coding the rate must be a factor $\Omega(\log|V|)$ smaller. However, so far no fast algorithms for constructing appropriate coding schemes were known. Our main result are polynomial time algorithms for constructing coding schemes for multicasting at the maximal data rate.

Categories and Subject Descriptors

E.4 [**Coding and Information Theory**]: Formal models of communication; C.2.2 [**Network Protocols**]: Routing protocols; F.2.2 [**Nonnumerical Algorithms and Problems**]: Routing and layout

General Terms

algorithms theory

Keywords

Coding, communication, derandomization, finite field, linear algebra, multicasting, network information theory, randomized algorithm

1. INTRODUCTION

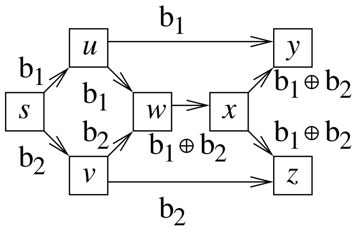

Figure 1: An example where coding helps [1].

In this paper, we study the problem of multicasting: Consider a directed 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 transport symbols of some alphabet.

If there is only one sink $t \in T$, we have the well known max-flow problem and the maximum data rate corresponds to the maximum flow from s to t which equals the minimum cut separating s from t. Maximum flows can be found in polynomial time. Furthermore, a flow of magnitude h can be decomposed into h edge disjoint paths so that multicasting simply has to send one input symbol along each of these paths.

Things get more complicated for multiple sinks. For example, consider the graph in Figure 1 [1]. There are flows of magnitude two from s to each sink in $T = \{y, z\}$. The reader can easly verify that there is no way to assign two 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 our 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* networks, i.e., the upper bound on the obtainable data rate imposed by the minimum cut separating s from some sink $t \in T$ can be achieved using coding [1, 15]. This area of *network information flow* is conceptually interesting because it brings together the seemingly unrelated concepts of coding and network flows.

As we have seen in the example from Figure 1, the

*Partially supported by DFG grant SA 933/1-1

data rate can be smaller if coding is not allowed, i.e., nodes can only forward (copies of) symbols they receive. In Section 4 we give simple examples where the rate achievable without coding must be a factor $\Omega(\log |V|)$ smaller. In addition, maximizing multicast data rate without coding is at least as hard as the minimum Steiner tree problem [4, 12], i.e., even for undirected graphs, $1 + \epsilon$ approximation is hard for some $\epsilon > 0.01$ [5] and after intensive research, no algorithm achieving an approximation factor better than 1.55 is known [17].

Our main result is that although coding allows *higher* data rates, finding optimal multicast coding schemes is possible in *polynomial* time.[1]

1.1 Overview

We continue the introduction with basic notation in Section 1.2 and a review of related work in Section 1.3. As our main result, Section 2 develops polynomial time algorithms for unit capacity directed acyclic graphs. Section 3 gives adaptations for capacitated and cyclic graphs that achieve almost optimal rate. The logarithmic gap to multicas ting without coding is demonstrated in Section 4. Section 5 gives an alternative algorithm for multicasting with coding that trades faster coding scheme construction for higher requirements for the size of the finite field used. This algorithm also illustrates relations between different approaches to reasoning about multicasting with coding. Section 6 discusses some open questions. Appendix A summarizes our notation.

1.2 Basic Notation

Consider an acyclic, unit capacity network $G = (V, E)$ where parallel edges are allowed. Node $s \in V$ is the *source* node, $T \subseteq V$ the set of *sink* nodes, h is the size of the smallest min-cut separating s from any $t \in T$. We use the notation $\Gamma^-(v)$ and $\Gamma^+(v)$ for the set of edges reaching and leaving node v respectively; $\mathrm{start}(e)$ denotes the node at which edge e starts.

We use linear coding over a finite field $\mathbb{F}$. It will turn out that $|\mathbb{F}| = \mathcal{O}(|T|)$ is sufficient. The source node s gets h input symbols from $\mathbb{F}$. The symbol $y(e) \in \mathbb{F}$ carried by an edge e is a linear combination of the symbols carried by the edges entering $\mathrm{start}(e)$. The *local coding vector* $m_e : \Gamma^-(\mathrm{start}(e)) \to \mathbb{F}$ determines the coefficients of this linear combination, i.e.,

$$y(e) = \sum_{e' \in \Gamma^-(\mathrm{start}(e))} m_e(e') y(e') \ .$$

This formula also holds for edges leaving the source node if we introduce a dummy node s' and h parallel input edges $e_1, \ldots, e_k$ connecting s' and s such that $y(e_i)$ is the i-th input symbol. Our task is to determine the coefficients $m_e(e')$ such that all sinks can reconstruct the original information from the symbols reaching them. Note that this definition assumes that only a single batch of h symbols are sent. Nodes wait until all incoming data has arrived. In Section 3.1 we will see that a multicasting scheme of this type can easily be adapted to a pipelined scheme where h symbols per timestep are communicated.

The symbol $\delta_{u,v}$ stands for 1 if $u = v$ and 0 otherwise.

1.3 Previous Work

Ahlswede et al. [1] have shown that the source can multicast information at a rate approaching h to all the sinks as the symbol size approaches infinity. They give a simple example that shows that without coding this rate is not always achievable. Li et al. [15] show that linear coding can be used for multicasting with rate h and finite symbol 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 an exponential number of edge sets 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 Medard [14] give an elegant algebraic characterization of linear coding schemes that achieve the maximal data rate. 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. They also give an algorithm for constructing coding schemes. However, this algorithm involves checking a polynomial identity $F(\mathbf{x}) = 0$ for a multivariate polynomial F with an exponential number of coefficients. Jaggi, Chou, and Jain [11] have independently and concurrently developed an algorithm similar to our deterministic algorithm from Section 2 that runs in time $\mathcal{O}\big(|E|(|T|^4 + h^3)\big)$ plus the time for $|T|$ maximum flow computations.

There is a number of results for multicasting without coding in undirected networks. The single source problem then amounts to (fractionally) packing Steiner trees connecting $\{s\} \cup T$. The fractional Steiner tree packing problem is equivalent to the minimum weight Steiner tree problem (MWST) via duality of the corresponding linear programs [4, 12]. In particular, approximation algorithms for MWST yield approximation algorithms for packing Steiner trees with the same approximation ratio using the ellipsoid method. Without coding, multiple multicast requests can be decomposed into a corresponding number of Steiner tree packing problems using linear programming. Using randomized rounding one obtains approximate integral solutions to multiple multicast requests [18, 4]. The approximation quality is a constant factor plus a logarithmic additive term. Jansen-Zhang and Baltz-Srivastav [13, 3] avoid using the ellipsoid method at the cost of a slightly worse approximation ratio. Using this approach, running times as low as $|E|(|E| + k\mathrm{polylog}(k|E|))$ can be achieved where k is the number of unit data rate multicast requests.

2. POLYNOMIAL TIME CODING

This section is concerned with establishing the following result.

THEOREM 1. *Consider an acyclic unit capacity multigraph (V, E) and let h denote the minimum cut size between the source s and any sink $t \in T$. The* Linear

[1]Independently, Jaggi, Chou, and Jain [11] have obtained a similar result.

Information Flow algorithm constructs linear codes for all nodes in $\mathcal{O}(|E| \cdot |T| \cdot h^2)$ expected time in a randomized variant and in $\mathcal{O}(|E| \cdot |T| \cdot (h^2 + |T|^2))$ worst case time in a deterministic variant. These codes have the following properties:*

- *Any finite field of size $|\mathbb{F}| \geq 2 \cdot |T|$ can be used to represent symbols sent along edges. For the deterministic case, $|\mathbb{F}| \geq |T|$ is sufficient.*
- *The source gets h information symbols as input.*
- *A node needs time $\mathcal{O}(\min(|T|, |\Gamma^-(\mathrm{start}(e))|))$ to compute the symbol to be sent along edge e. The source needs time $\mathcal{O}(h)$ for each edge.*
- *Each sink can reconstruct all h information symbols in time $\mathcal{O}(h^2)$.*

Building on the notation given in Section 1.2 we explain the effects of linear coding in terms of linear algebra. Since linear coding is used, the information carried by an edge e is a linear combination of the input symbols of s. We can characterize the effect of all the local coding vectors on edge e independently of a concrete input using *global coding vectors* $\mathbf{b}(e) \in \mathbb{F}^h$. The input edges of s have the unit vectors $\mathbf{b}(e_i) = [0^{i-1}, 1, 0^{h-i}]$ and the other global coding vectors are inductively defined by $\mathbf{b}(e) = \sum_{p \in \Gamma^-(\mathrm{start}(e))} m_e(p)\mathbf{b}(p)$ for $e \in E$. These 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 $\{\mathbf{b}(p) : p \in \Gamma^-(t)\}$ span the vector space $\mathbb{F}^h$. Reconstructing the original information can then be achieved by solving a linear system of equations of polynomial size. The intuition is that a linear code mixes the information received from different edges but does not loose essential information as long as there is a bijective mapping between the input and the data reaching the sink.

The challenge is now to find the local coding vectors efficiently, possibly using a small finite field that allows fast arithmetics. 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 i-th input symbol along the i-th edge disjoint path. If an edge e is 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 determined. The algorithm defines the coefficients of m_e for one edge $e \in \Gamma^+(u)$ after the other. There might be multiple flow paths to different sinks through edge e. Let $T(e)$ denote the set of sinks using e in some

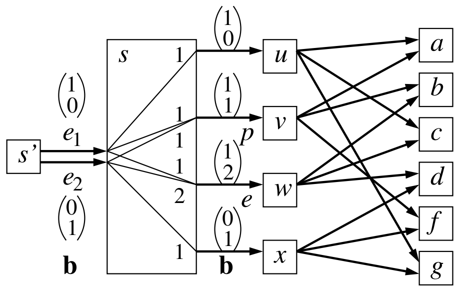

Figure 2: An example for multicasting with linear coding from s to $T = \{a, b, c, d, f, g\}$. We have $h = 2$. Assume that all the flows are decomposed into a topmost path and a bottommost path. The thin lines within s give nonzero coefficients for local coding vectors. The b vectors give the resulting global coding vectors. Let us assume that $\mathbb{F} = \mathrm{GF}(3)$ and that the edges leaving s are considered from top to bottom. Then the only feasible linear combinations for 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 $\Gamma^-(b) = \{(v, b), (w, b)\}$, $\mathrm{start}(e) = s$, $T(e) = \{b, c, d\}$, $P(e) = \{e_1, e_2\}$, $f^d_{\leftarrow}(e) = e_1$, and $f^c_{\leftarrow}(e) = e_2$. Before m_e is fixed, $C_b = \{p, e_2\}$ and correspondingly $B_b = \{\binom{1}{1}, \binom{0}{1}\}$.

flow f^t and let $P(e) = \{f^t_{\leftarrow}(e) : t \in T(e)\}$ denote the set of predecessors edges of e in some flow path. 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. [15] verifies that the global coding vector $\mathbf{b}(e)$ is linearly independent of an exponential number of sets of other global coding vectors. Our algorithm can dramatically 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 of h edges C_t such that the global coding vectors $B_t := \{\mathbf{b}(c) : c \in C_t\}$ are a basis of $\mathbb{F}^h$, i.e., the original input can be reconstructed from the information carried by the edges in C_t. 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, $C_t \subseteq \Gamma^-(t)$ and the invariant ensures that sink t gets all the information.

We first establish the invariant by assigning the artificial input edges $\{e_1, \ldots, e_h\}$ with $\mathbf{b}(e_i) = [0^{i-1}, 1, 0^{h-i}]$ to C_t. When the linear combination m_e for a new edge e has been defined, we have to exchange $f^t_{\leftarrow}(e)$ by e in all the C_t with $t \in T(e)$. Hence, to maintain the invariant, it suffices to check for all $t \in T(e)$ whether B_t still spans $\mathbb{F}^h$. Figure 2 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 $\mathbb{F} \geq 2 \cdot |T|$: Lemma 2 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 $|T|/|F| \leq 1/2$. This straight-forward construction yields a Monte-Carlo algorithm for finding a *single* local coding vector, i.e., the construction can fail. To find *all* coding vectors fast using a small field, this is not sufficient. Here the knowledge of the flows encoded in the C_ts and the invariant come in and allow us to convert the Monte Carlo Algorithm into a Las Vegas algorithm, i.e., a constant expected number of trials followed by $|T(e)|$ independence tests suffices to find a feasible local coding vector.

LEMMA 2. *For any $e \in E$ and $t \in T(e)$ assume that B_t is a basis of $\mathbb{F}^h$. Then with probability $1/|\mathbb{F}|$, a random local coding vector $m_e : P(e) \to \mathbb{F}$ fulfills the property that $(B_t \setminus \{\mathbf{b}(f_\leftarrow^t(e))\}) \cup \{\mathbf{b}(e)\}$ is a basis of $\mathbb{F}^h$ where $\mathbf{b}(e)$ is the corresponding global coding vector $\sum_{p \in P(e)} m_e(p)\mathbf{b}(p)$.*

PROOF. If we fix the coefficients $m_e(p)$ for $p \in P(e) \setminus \{f_\leftarrow^t(e)\}$ then there is exactly one choice of $m_e(f_\leftarrow^t(e))$ for which $\mathbf{b}(e)$ is linearly dependent of $B_t \setminus \{\mathbf{b}(f_\leftarrow^t(e))\}$, i.e., 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 $|\mathbb{F}|^{|P(e)|-1}/|\mathbb{F}|^{|P(e)|} = 1/|\mathbb{F}|$. ∎

What we have said so far already yields an algorithm running in polynomial expected time. In what follows, we further refine the algorithm to obtain a fast and more concrete implementation (Figure 3) and a deterministic way of choosing the linear combinations m_e.

2.1 Testing Linear Independence Fast

The mathematical basis for our refinement of the LIF algorithm is the following lemma which states that the problem of testing linear independence from an $h-1$ dimensional vector space can be reduced to a single scalar product.

LEMMA 3. *Consider a basis B of $\mathbb{F}^h$ and vectors $\mathbf{b} \in B$, $\mathbf{a} \in \mathbb{F}^h$ such that $\forall \mathbf{b}' \in B : \mathbf{b}' \cdot \mathbf{a} = \delta_{\mathbf{b},\mathbf{b}'}$. Then, any vector $\mathbf{x} \in \mathbb{F}^h$ is linearly independent of $B \setminus \{\mathbf{b}\}$ if and only if $\mathbf{x} \cdot \mathbf{a} \neq 0$.*

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

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

Now, $\mathbf{x}$ is linearly independent of $B \setminus \{\mathbf{b}\}$ if and only if $x(\mathbf{b}) \neq 0$, i.e, if and only if $\mathbf{x} \cdot \mathbf{a} = 0$. ∎

The LIF-algorithm given in Figure 3 maintains vectors $\mathbf{a}_t(c)$ for each sink t and edge $c \in C_t$ that can be used to test linear independence of $B_t \setminus \{\mathbf{b}(c)\}$. The invariant now becomes

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

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

The algorithm in Figure 3 implements the outline of the LIF algorithm given above. To prove correctness we have to verify the loop invariant.

LEMMA 4. *Loop-Invariant (1) holds for $(C_t, B_t, \mathbf{a}_t)$.*

PROOF. (By induction) Before the loop over the vertices, Loop-Invariant (1) is trivially satisfied. Now assume as induction hypothesis that Invariant (1) holds for $(C_t, B_t, \mathbf{a}_t)$. We show that it holds for $(C'_t, B'_t, \mathbf{a}'_t)$.

In C_t we replace edge $f_\leftarrow^t(e)$ by edge e, hence the size of C'_t is the same as the size of C_t. According to the algorithm, $\mathbf{b}(e)$ is chosen linearly independent to $B_t \setminus \{\mathbf{b}(f_\leftarrow^t(e))\}$. Hence, $\mathbf{b}(e) \cdot \mathbf{a}_t(f_\leftarrow^t(e)) \neq 0$ by Lemma 3, and $\mathbf{a}'_t(e)$ is well defined. Finally, we verify $\mathbf{b}(c) \cdot \mathbf{a}'_t(c') = \delta_{c,c'}$ for all $c, c' \in C'_t$ by a short calculation:

$$\begin{aligned}
\mathbf{b}(e) \cdot \mathbf{a}'_t(e) &= \mathbf{b}(e) \cdot \left(\mathbf{b}(e) \cdot \mathbf{a}_t(f_\leftarrow^t(e))\right)^{-1} \mathbf{a}_t(f_\leftarrow^t(e)) = 1, \\
\mathbf{b}(e) \cdot \mathbf{a}'_t(c) &= \mathbf{b}(e) \cdot \mathbf{a}_t(c) - (\mathbf{b}(e) \cdot \mathbf{a}_t(c))(\mathbf{b}(e) \cdot \mathbf{a}'_t(e)) \\
&= 0 \quad \text{for } c \neq e \\
\mathbf{b}(c) \cdot \mathbf{a}'_t(e) &= \left(\mathbf{b}(e) \cdot \mathbf{a}_t(f_\leftarrow^t(e))\right)^{-1} (\mathbf{b}(c) \cdot \mathbf{a}_t(f_\leftarrow^t(e))) \\
&= 0 \quad \text{for } c \neq e \\
\mathbf{b}(c) \cdot \mathbf{a}'_t(c') &= \mathbf{b}(c) \cdot \left(\mathbf{a}_t(c') - (\mathbf{b}(e) \cdot \mathbf{a}_t(c'))\mathbf{a}'_t(e)\right) \\
&= \mathbf{b}(c) \cdot \mathbf{a}_t(c') - (\mathbf{b}(e) \cdot \mathbf{a}_t(c'))(\mathbf{b}(c) \cdot \mathbf{a}'_t(e)) \\
&= \mathbf{b}(c) \cdot \mathbf{a}_t(c') = \delta_{c,c'} \quad \text{for } c, c' \neq e.
\end{aligned}$$

∎

Remark. If the vectors in B_t are arranged as the rows of a matrix $\mathbf{B}_t$ and the columns $\mathbf{a}_t$ are correspondingly arranged as a matrix $\mathbf{A}_t$, then the invariant is equivalent to $\mathbf{A}_t = \mathbf{B}_t^{-1}$. In this notation, the method of updating the inverse vectors $\mathbf{a}_t$ in the LIF algorithm is a special case of the Sherman-Morrison formula [16, Section 2.7].

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

LEMMA 5. *If Line **(*)** in Figure 3 is implemented by choosing random $m_e : P(e) \to \mathbb{F}$ until the condition "$\forall t \in T(e) : \mathbf{b}(e)$ is linearly independent of $B_t \setminus \{\mathbf{b}(f_\leftarrow^t(e))\}$" is satisfied, then the algorithm can be implemented to run in 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 round robin to each sink until for some sink no augmenting paths are left. Hence, we can find h augmenting paths for each sink in time $\mathcal{O}(|T| \cdot |E| \cdot h)$.[2]

The algorithm then constructs the finite field $\mathbb{F}$ of 2^m elements for m chosen such that $|\mathbb{F}| < 4|T|$. This involves the creation of a lookup table of $|\mathbb{F}|$ entries for incrementing elements (Conway's "Zech-logarithm", e.g.,

[2]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 [9].

Function LIF(V, E, s, T)
$h := \min_{t \in T} \min \{|C| : C \text{ is } s\text{-}t \text{ cut}\}$ -- $= \min_{t\in T} |\text{max flow from } s \text{ to } t|$
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 the field of size 2^m, $m = \lfloor \log_2 \max\{|T(e)| : e \in E\}\rfloor + 1$

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 -- **(*)**
   $\forall t \in T(e) : \mathbf{b}(e)$ is linearly independent of $B_t \setminus \{\mathbf{b}(f^t_{\leftarrow}(e))\}$
  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
   $\mathbf{a}'_t(e) := \left(\mathbf{b}(e) \cdot \mathbf{a}_t(f^t_{\leftarrow}(e))\right)^{-1} \mathbf{a}_t(f^t_{\leftarrow}(e))$ -- 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 $(h, \{m_e : e \in E\}, \{(C_t, \mathbf{a}_t) : t \in T\}, \mathbb{F})$.

Figure 3: Linear Information Flow (LIF) coding 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.

[10, 8]). Using this table, any arithmetic operation in $\mathbb{F}$ can be computed in constant time.[3]

Initializing C_t, B_t, and $\mathbf{a}_t(c)$ takes time $\mathcal{O}(|T| \cdot h^2)$. 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 $\mathbf{b}(e)$ and testing linear independence using the vectors $\mathbf{a}_t(c)$ takes time $\mathcal{O}(h|T|)$. 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 $\mathbf{a}'_t(c)$ for all t and all $c \in C_t$ takes time $\mathcal{O}(|T| \cdot h^2)$.

Combining all the parts, we get the claimed expected time bound of $\mathcal{O}(|E| \cdot |T| \cdot h^2)$. Sink $t \in T$ can reconstruct the vector of input symbols $\mathbf{x}$ at s by computing $\mathbf{x} = \sum_{c \in C_t} \mathbf{a}_t(c) y(c)$, where $y(c)$ denotes the symbol received over edge $c \in C_t$. This takes time $\mathcal{O}(h^2)$. ■

2.2 Deterministic Implementation

We now explain how the algorithm LIF in Figure 3 can be implemented deterministically. We develop a deterministic method for choosing the local coding vectors in step (*) using the following two lemmas that we formulate as pure linear algebra problems without using graph theoretic concepts.

LEMMA 6. *Consider pairs $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{F}^h \times \mathbb{F}^h$ with $\mathbf{x}_i \cdot \mathbf{y}_i \neq 0$ for $1 \leq i \leq n \leq |\mathbb{F}|$. Then there is a linear combination $\mathbf{u}$ of $\mathbf{x}_1, \ldots, \mathbf{x}_n$ such that $\forall i : \mathbf{u} \cdot \mathbf{y_i} \neq 0$.*

PROOF. Let V denote the vector space spanned by $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and let d denote its dimension. Then $K_i = \{\mathbf{x} \in V : \mathbf{x} \cdot \mathbf{y}_i = 0\}$ is a $d-1$ dimensional subspace of V for all i. Clearly, $\mathbf{u}$ satisfies $\mathbf{u} \cdot \mathbf{y_i} \neq 0$ iff $\mathbf{u} \notin K_i$. As the K_i have at least the null vector in common, the number of linear combinations $\mathbf{u}$ as required in the lemma is

$$\begin{aligned} U = \left| V \setminus \bigcup_{i=1}^{n} K_i \right| &\geq |V| - \left(1 + \sum_{i=1}^{n} |K_i \setminus \{\mathbf{0}\}|\right) \\ &= |\mathbb{F}|^d - \left(1 + n\left(|\mathbb{F}|^{d-1} - 1\right)\right) \\ &= \left(1 - \frac{n}{|\mathbb{F}|}\right)|\mathbb{F}|^d + n - 1. \end{aligned}$$

If $n < |\mathbb{F}|$ then $U \geq |\mathbb{F}|^{d-1} \geq 1$. If $n = |\mathbb{F}|$ then $U \geq |\mathbb{F}| - 1 \geq 1$ because a field has at least two elements. ■

[3] 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 $|T|$.

The linear combination $\mathbf{u}$ from Lemma 6 can be found by fixing one coefficient of the linear combination after the other in a greedy fashion.

LEMMA 7. *A linear combination* $\mathbf{u}$ *proven to exist in Lemma 6 can be found in worst case time* $\mathcal{O}\big(n^2(h+|\mathbb{F}|)\big)$.

PROOF. The proof is by induction on n. For $n=1$ we simply set $\mathbf{u}=\mathbf{x_1}$.

For the induction step from $n-1$ to n, we apply the induction hypothesis to obtain a vector $\mathbf{u}$ with $\mathbf{u}\cdot\mathbf{y}_i\neq 0$ for $i<n$. If $\mathbf{u}\cdot\mathbf{y}_n$ happens to be nonzero, we are done.

Otherwise, we apply Lemma 6 to $(\mathbf{u},\mathbf{y}_1),\ldots,(\mathbf{u},\mathbf{y}_{n-1})$, $(\mathbf{x}_n,\mathbf{y}_n)$. As $n\leq|\mathbb{F}|$, there is a linear combination $\mathbf{u}''=\beta\mathbf{u}+\gamma\mathbf{x}_n$ such that $\forall i\leq n:\mathbf{u}''\cdot\mathbf{y}_i\neq 0$. In particular, $\mathbf{u}''\cdot\mathbf{y}_n=\gamma\mathbf{x}_n\neq 0$ since $\mathbf{u}\cdot\mathbf{y}_n=0$. Hence, $\gamma\neq 0$ and $\mathbf{u}'=\mathbf{u}''/\gamma$ is of the form $\alpha\mathbf{u}+\mathbf{x}_n$ and we have $\forall i\leq n:\mathbf{u}'\cdot\mathbf{y}_i\neq 0$. The coefficient α can be found by trying all field elements. All $|\mathbb{F}|$ trials can be made in time $\mathcal{O}(n(h+|\mathbb{F}|))$ by precomputing the scalar products $\mathbf{u}\cdot\mathbf{y}_i$ and $\mathbf{x}_n\cdot\mathbf{y}_i$ for $1\leq i<n$. We get a total time of $\mathcal{O}\big(n^2(h+\mathbb{F})\big)$ for n iterations.[4] ∎

Lemma 7 can be used to find the linear combination m_e in the LIF algorithm:
Apply Lemma 7 to $\{(\mathbf{b}(f^t_\leftarrow(e)),\mathbf{a}_t(f^t_\leftarrow(e))) : t\in T(e)\}$, i.e., let $\mathbf{b}(e)=\mathbf{u}=\sum_{t\in T(e)}u_t\mathbf{b}(f^t_\leftarrow(e))$ denote a vector with $\mathbf{b}(e)\cdot\mathbf{a}_t(f^t_\leftarrow(e))\neq 0$. This value for $\mathbf{b}(e)$ can also be obtained by setting $m_e(p)$ to the sum of all u_t with $f^t_\leftarrow(e)=p$.

The deterministic part of Theorem 1 can now be proven analogously to the proof of Lemma 5. We just have to substitute the expected time $\mathcal{O}(|T|\cdot h)$ needed by the randomized routine for choosing m_e by the time $\mathcal{O}\big(|T(e)|^2\cdot(|\mathbb{F}|+h)\big)=\mathcal{O}\big(|T|^2(|T|+h\big)$ needed to apply Lemma 7. We obtain a total execution time of $\mathcal{O}\big(|E|(|T|h^2+|T|^2(|T|+h))\big)=\mathcal{O}\big(|E|\cdot|T|(h^2+|T|^2)\big)$.

3. REFINEMENTS

3.1 Pipelining

So far our analysis only covers sending h symbols from s to the sinks. In reality, we want to send h symbols per time step in a pipelined fashion. If nodes naively forward linear combinations of the symbols received in the previous time step, we can get a mix of the information sent during different time steps that is difficult to decode.

Still, this naive algorithm works fine for *layered graphs* where a node v can be mapped to a layer $\ell(v)$ such that $\ell(s)=0$ and edges only exist between adjacent layers. In that case, the information received by a node at layer i at time step τ is a linear combination of the information sent from s at time step $\tau-i$. Acyclic graphs that are not layered, are easy to convert into layered graphs by replacing edges spanning $k+1$ layers by a chain of k new nodes. Figure 4 gives an example. Since these new nodes have indegree and outdegree 1, no additional coding is necessary. They can be emulated by the original network by simply introducing time delays, i.e., data sent along an edge spanning $k+1$ layers is delayed by k steps.

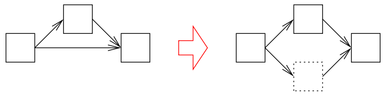

Figure 4: Transformation into a layered graph.

3.2 Edge Capacities

We now consider acyclic graphs with integer edge capacities $c(e)$. By replacing each capacitated edge by $c(e)$ parallel unit capacity edges, Section 2 immediately yields pseudopolynomial time algorithms, i.e., algorithms with running time polynomial in $|V|$, $|E|$, and $\sum_{e\in E}c(e)$. However, $\sum_{e\in E}c(e)$ can be exponential in the number of bits needed to define the input graph. Hence, the question arises how to handle graphs with very large edge capacities. Again, Section 2 (almost) suffices to solve this problem:

THEOREM 8. *Let* h *denote the maximum flow in an acyclic network* $G=(V,E)$ *with edge capacities* $c:E\to\mathbb{N}$. *For any* $\epsilon>0$ *such that* $\epsilon h\in\mathbb{N}$,[5] *linear codes for network information flow 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 $t\in T$ and reduce $c(e)$ to $\max_{t\in T}f^t(e)$. In particular, 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 network $G'=(V,E')$ with unit capacity edges 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 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 $h\epsilon/w$, or $h\epsilon$ on all paths on one flow. Hence, the total used capacity is at least $h-h\epsilon=(1-\epsilon)h$. ∎

3.3 Cyclic Graphs

Ahlswede et al. [1] explain how to convert a cyclic graph $G=(V,E)$ with maximal obtainable rate h into an acyclic graph $G'=(V',E')$ with $|V'|=a\cdot|V|$, $|E'|=\mathcal{O}(a\cdot|E|)$ and rate $h'\geq(a-|V|)h$. They further explain how the communication on this graph can be emulated in a time steps on the cyclic graph G. This

[4] A more detailed analysis shows that at most n trials are needed and that only a constant expected number of trials are needed if the coefficients are tried in random order and $|\mathbb{F}|\geq 2n$.

[5] 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.

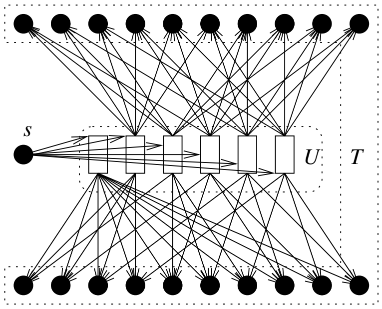

Figure 5: An example where three symbols per time step can be delivered. Without coding, the best we can do is to send three symbols every two time steps.

transformation can be used together with our polynomial time algorithm for finding coding schemes. By choosing $a = |V|/\epsilon$ we obtain a coding scheme that achieves a rate of $(1-\epsilon)h$.

4. THE GAP TO MULTICASTING WITHOUT CODING

THEOREM 9. *There are acyclic networks with unit capacity edges where multicasting with coding allows a factor $\Omega(\log|V|)$ larger rate than multicasting without coding.*

PROOF. Consider the network $G = (V, E)$ with vertices $V = \{s\} \cup U \cup T$ where

$$U = \{1, \ldots, 2h\}, T = \{t \subseteq U \mid |t| = h\}$$

and edges

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

The network has three layers: The source s constitutes the first layer, the second layer U has $2h$ nodes, and the sink nodes T constitute the third layer — one node for each h-element subset of U. Note that $h = \Theta(\log|V|)$. Figures 2 and 5 give examples for $h = 2$ and $h = 3$ respectively.

Since the min-cut has size h, the rate with coding is h.[6] Without coding, the rate is less than 2. To see this, suppose the source attempts to send $2n$ symbols $b_1, \ldots, b_{2n}$ to each of the sinks in T using n consecutive uncoded transmissions. Let U_i denote the subset of intermediate nodes receiving b_i. As the edges are unit capacity, the source can send at most $2hn$ symbols in total to the intermediate nodes, so $\sum_{i=1}^{2n} |U_i| \le 2hn$. This implies that there is an i for which $|U_i| \le h$.

For any h-element subset t of $U \setminus U_i$, the sink t does not receive the symbol b_i. ■

[6]In this network it suffices if the source s encodes the information using a code of length $2h$ that conveys h information symbols and allows reconstruction of the information from any h received symbols. This is called a $[2h, h]$ MDS code.

5. 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 10. *Linear network coding schemes using finite fields of size $|\mathbb{F}| \ge 2 \cdot |E| \cdot |T|$ and achieving rate h can be found in expected time $\mathcal{O}(|E| \cdot |T|h + |T|\mathcal{M}(h))$ where $\mathcal{M}(h) = \mathcal{O}(h^{2.376})$ 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 $\mathbf{b}(e)$ (time $\mathcal{O}(\sum_{e \in E} |P(e)| \cdot h) = \mathcal{O}(|E| \cdot |T| \cdot h)$). 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 2 and the analysis of the "careful" algorithm from Section 2 it can be seen that the success probability is at least 1/2: The careful algorithm performs $\sum_{e \in E} |T(e)|$ independence tests, each going wrong with probability $1/|\mathbb{F}|$. Without the tests, the failure probability is at most $\sum_{e \in E} |T(e)|/|\mathbb{F}| \le 1/2$.The expected number of repetitions will be constant. ■

This algorithm is quite similar to the one by Ahlswede et al. [1]. The main difference is that we choose random linear local coding functions rather than completely arbitrary random functions. 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.[7] 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, using flow arguments, 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 Medard [14] arrive at a similar requirement for $|\mathbb{F}|$ as Theorem 10 using quite different algebraic arguments.

6. DISCUSSION

Our polynomial time algorithms for multicasting in acyclic networks are quite simple and could perhaps be used in practice if the rate h is not too large (otherwise

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

decoding could become too expensive). In addition, the algorithms are not limited to maximizing the rate. In a particular application, e.g. streaming multimedia content in a large computer network, the rate h' *required* could be fixed. In this situation, one could find a cheap or otherwise favorable subgraph with minimum cut at least h' and use our algorithms to construct codes to achieve multicast rate h' from the server to each client.

There are a number of open problems for graphs with cycles. Is there a polynomial time algorithm that finds a coding scheme with rate *exactly* h? From a practical point of view even an approximate algorithm would be interesting if it allows coding schemes that are faster to decode. Currently, rate $(1-\epsilon)h$ requires decoding time $\Omega(h \cdot |V|/\epsilon)$ per decoded symbol which looks forbidding for large graphs.

Very little is known about the problem with multiple sources. Will we again find that with coding optimal solutions are both easier to find and more efficient than solutions without coding?

Acknowledgements

We would like to thank Rudolf Ahlswede, Ning Cai, Philip Chou, Sidharth Jaggi, Irit Katriel, Piotr Krysta, Anand Srivastav, and Berthold Vöcking for fruitful discussions.

7. REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

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

[3] A. Baltz and A. Srivastav. Approximate implementation of minimum multicast congestion — implementation versus theory. Manuscript.

[4] B. Carr and S. Vempala. Randomized meta-rounding. In *32nd ACM Symposium on the Theory of Computing*, pages 58–62, 2000. also in Random Structures and Algorithms, 20(3):343–352, 2002.

[5] M. Chlebik and J. Chlebikova. Approximation hardness of the steiner tree problem on graphs. In *8th Scandinavian Workshop on Algorithm Theory*, number 2368 in LNCS, pages 170–179, 2002.

[6] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9:251–280, 1990.

[7] E. A. Dinic. Algorithm for solution of a problem of maximum flow. *Soviet Math. Dokl.*, 11:1277–1280, 1970. now spelled 'Dinitz'.

[8] J. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In *ISSAC*, pages 63–74, Lille, 2002. ACM.

[9] S. Even and E. Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4:507–518, 1975.

[10] K. Imamura. A method for computing addition tables in $GF(p^n)$. *IEEE Transactions on Information Theory*, 26:367–369, 1980.

[11] S. Jaggi, P. S. Chou, and K. Jain. Low complexity algebraic multicast network codes. In *International Symposium on Information Theory.* IEEE, July 2003. 1 page abstract, to appear.

[12] K. Jain, M. Mahdian, and M. R. Salavatipour. Packing Steiner trees. In *14th ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[13] K. Jansen and H. Zhang. An approximation algorithm for the multicast congestion problem via minimum Steiner trees. In *3rd International Workshop on Approximation and Randomized Algorithms in Communication Networks (ARANCE)*, Rome, Italy, 2002.

[14] R. Koetter and M. Medard. An algebraic approach to network coding. In *Proceedings of INFOCOM*, 2002.

[15] S.-Y. R. Li, R. W. Yeung, and N. Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003.

[16] W. H. Press, S. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 2nd edition, 1992.

[17] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *11th ACM Symposium on Parallel Architectures and Algorithms*, pages 770–779, 2000.

[18] S. Vempala and B. Vöcking. Approximating multicast congestion. In *10th Int. Symp. on Algorithms and Computation*, number 1741 in LNCS, pages 367–372, Chennai, 1999. Springer.

APPENDIX

A. NOTATION

$\mathbf{a}_t(c)$**:** a vector with the property that $\mathbf{x} \cdot \mathbf{a}_t(c) \neq 0$ if and only if $\mathbf{x}$ is linearly independent of $B_t \setminus \{\mathbf{b}(c)\}$ for some $c \in C_t$.

$\mathbf{b}(e) \in \mathbb{F}^h$**:** global coding vector for edge $e \in E$

B_t**:** the set of global coding vectors $\{\mathbf{b}(c) : c \in C_t\}$

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 $t \in T$

$\Gamma^-(v)$**:** the set of edges entering node $v \in V$

$\Gamma^+(v)$**:** the set of edges leaving node $v \in V$

$G = (V, E)$**:** the graph

h**:** the smallest maximum flow from s to some sink $t \in T$

$\mathrm{start}(e)$**:** the node where edge $e \in E$ departs

$m_e : \Gamma^-(\mathrm{start}(e)) \rightarrow \mathbb{F}$**:** 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 $\left\{f^t_{\leftarrow}(e) : t \in T(e)\right\}$

s**:** source node

s'**:** dummy source node

$T \subseteq V$**:** the set of sink nodes

$T(e) \subseteq T$ the sinks supplied through edge $e \in E$, i.e., $T(e) = \left\{t \in T : e \in f^t\right\}$

$t \in T$**:** a sink node

V**:** the set of nodes

$v \in V$**:** a node

$y(e)$**:** the symbol carried by edge $e \in E$