A Simple Algebraic Formulation for the Scalar Linear Network Coding Problem

Abhay T. Subramanian, Andrew Thangaraj, *Member, IEEE*

***Abstract*—In this work, we derive an algebraic formulation for the scalar linear network coding problem as an alternative to the one presented by Koetter and Médard in their work. We first show an equivalence between network information flow and group-valued circulations. Given a general network coding problem, we provide an algorithm to generate a graph (specifically, a collection of trees) on which group-valued circulations are equivalent to information flow in the original network. We use this collection of trees to derive a system of polynomial equations that algebraically represents the scalar linear network coding problem. Surprisingly, this system of polynomials has a maximum degree of 2. We illustrate our formulation and its advantages over the formulation presented by Koetter and Médard in terms of the number of variables and equations involved (apart from a reduction in degree) through example networks drawn from the literature.**

I. Introduction

THE idea of network coding over error-free networks, pioneered in [1], has been a subject of active current research. The general idea of linear network coding, where intermediate nodes linearly combine incoming packets, was explored in [2]. A simple and effective algebraic formulation of the general network coding problem was introduced in [3]. This established a direct connection between a network information flow problem and an algebraic variety over the closure of a finite field.

Using the formulations of [2], [3], the multicast network coding problem, where one source transmits at the same rate to a set of sinks, has been characterized almost completely. A linear network code exists for the multicast case in a large enough finite field and can be found in polynomial time [4]. However, the general network coding problem still remains much harder to characterize. The insufficiency of linear coding in the non-multicast case has been demonstrated in [5]. Recent work in [6] and [7] has shown the restrictions imposed on the field characteristic for the scalar linear solvability of a general network coding problem. See [6] for more details on the state-of-the-art for the non-multicast problems.

The algebraic formulation of [3], while being simple and powerful, results in equations that are not readily amenable to easy solution in many cases. In this paper, our main result is to derive an alternative algebraic formulation for the general scalar linear network coding problem. Specifically, we show a correspondence between linear network-coded information flow in a given network and group-valued circulations in an equivalent set of directed trees. In a network, a group-valued circulation is a mapping from the set of edges to a group such that the net flow through any node is conserved under the group addition operation [8]. One can readily see that linear network-coded information flow should be closely related to group-valued circulations. However, since there is no multiplication operation in groups, we show that the network needs to be transformed before a direct relationship can be established between linear network coding over a field and a group valued circulation. We develop an algorithm for this network transformation, which results in a set of trees.

We then use the corresponding flow in the set of trees to derive a system of polynomial equations that provides and algebraic formulation for the network coding problem in the original network. Surprisingly, this set of equations has a maximum degree of only 2. Moreover, the form of the equations has additional structure that can be exploited in several cases. We illustrate this simplification through examples and compare our formulation with the one proposed in [3] in terms of the number of variables and equations involved.

An alternative way of viewing our formulation is that we perform a graphical simplification of the formulation in [3], which uses scaling variables on every link. The crux of our simplification lies in a graph transformation that migrates all scaling variables to the sources. All the intermediate nodes simply perform addition and have only a single outgoing node in the transformed graph (a set of directed trees). In the transformed graph, we get only linear and degree-2 equations relating the scaling variables at the sources.

We will start with a notational description of the network coding problem in Section II. Then, we will provide a brief motivation for the work presented here in Section III. In Section IV, we will introduce some theory on group-valued circulations. Our main result is presented in Section V, where we first give an algorithm that, given a network coding problem, constructs an equivalent group-valued circulation network. Then, we derive a system of polynomial equations that algebraically represents the given network coding problem. At the end of the section, we also give ways to simplify the derived system of equations. In Section VI, we further explain the transformation and algebraic formulation using various example networks drawn from the literature. We also provide results from the application of the algorithms described earlier to a large Internet Service Provider (ISP) network. Finally, in Section VII, we give an algorithm to derive the linear network code (the actual coding coefficients at intermediate nodes as in [3]) from a solution to the system of polynomial equations derived from the transformed graph. Appendix A provides

A. T. Subramanian and A. Thangaraj are with the Department of Electrical Engineering, Indian Institute of Technology Madras, Chennai, India. E-mail: abhay,andrew@iitm.ac.in

Manuscript received July 7, 2008

algorithms that, given a network coding problem, will give a count of the number of variables and equations resulting in our algebraic formulation described in Section V.

II. The Network Coding Problem

The communication network is modelled as a directed, acyclic multigraph, $G = (V, E)$, where the node set V represents the terminals and switches in the network and the edge set E represents the communication links. It is assumed, without loss of generality, that all communication links are error-free and have unit capacity. Any link of higher capacity between two nodes can be modelled as multiple unit capacity links between the nodes.

For a given edge $e = (u, v)$, we denote:

$$u = \text{tail}(e)$$
$$v = \text{head}(e)$$

For each node $v \in V$, we define:

$$I(v) = \{e \in E : \text{head}(e) = v\}$$
$$O(v) = \{e \in E : \text{tail}(e) = v\}$$

Let us further assume the following without loss of generality:

1) A node v is a source node iff $|I(v)| = 0$ and all source nodes produce exactly one unit of data per unit time.
2) A node v is a sink node iff $|O(v)| = 0$ and all sink nodes demand exactly one unit of data per unit time.

In cases where a node v produces (demands) more than one data symbol, we can add virtual source (sink) nodes that produce (demand) exactly one data symbol, have exactly one output (input) link connecting them to v and no input (output) links.

Then, the set of source and sink nodes is defined as:

$$S = \{v \in V : |I(v)| = 0\} = \{s_1, s_2, \ldots, s_{|S|}\}$$
$$T = \{v \in V : |O(v)| = 0\} = \{t_1, t_2, \ldots, t_{|T|}\}$$

Let us now assume that we use a finite alphabet H. For each edge e, an edge function is then defined as a mapping:

$$f_e : H^i \rightarrow H$$

where, $i = 1$ if $\text{tail}(e) \in S$ and $i = |I(\text{tail}(e))|$ otherwise.

Definition 1: The collection of all the edge functions in a given network is defined as a *network code*. If all the edge functions are linear maps with respect to a field alphabet H, then the code is a *scalar linear code*.

Let the data symbol generated at the i-th source node, $s_i \in S$, be denoted by X_i and the data symbol demanded by the j-th sink node, $t_j \in T$, be denoted by Z_j. These also implicitly define a set of connection requirements, denoted by $\mathcal{C}$, for the given network G. Given a network G, the set of source nodes S, the set of sink nodes T and the set of connection requirements $\mathcal{C}$, the network coding problem is to determine all the edge functions such that all the connection requirements are satisfied. If such a set of edge functions exists, then the network coding problem is *solvable*. If a set of linear edge functions, with respect to a finite alphabet H, exists that satisfies all the connection requirements, then the network coding problem is *scalar-linearly solvable*.

In a scalar linear network coded flow (over a field H), the edge function of an edge e can be written as $\sum_{i=1}^{|S|} a_i X_i$, where $a_i, X_i \in H$. We refer to $\sum_{i=1}^{|S|} a_i X_i$ as either the edge function of e or the symbol flowing through e and denote it as a vector $\mathbf{f}_e = [a_1\ a_2\ \cdots\ a_{|S|}]$.

III. Motivation

The scalar linear network coding problem was formulated as a system of polynomial equations in [3]. Our aim in this work has been to arrive at a simpler algebraic formulation for the general scalar-linear network coding problem than the one described in [3]. This advantage of the algebraic formulation that we will describe in the next few sections can be easily noticed when we compare the two formulations for the case of the modified butterfly network shown in Fig. 1 with two sources and four sinks. Note that the network in Fig. 1 is identical to the classic butterfly network under our definition of sources and sinks.

The edge functions under the direct assignment of scaling factors (as in [3]) is shown in Fig. 1. The formulation described

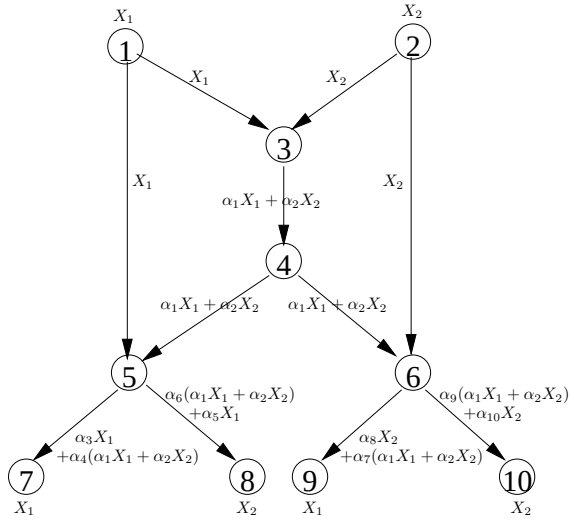

Fig. 1. Flow in the butterfly network.

in [3] gives the following 8 equations in 10 variables:

$$\alpha_3 + \alpha_4\alpha_1 = 1 \qquad \alpha_4\alpha_2 = 0$$
$$\alpha_5 + \alpha_6\alpha_1 = 0 \qquad \alpha_6\alpha_2 = 1$$
$$\alpha_7\alpha_2 + \alpha_8 = 0 \qquad \alpha_7\alpha_1 = 1$$
$$\alpha_9\alpha_2 + \alpha_{10} = 1 \qquad \alpha_9\alpha_1 = 0$$

In contrast, our formulation arrives at just 1 equation

$$a_4 b_3 = 1$$

in 2 variables as given in (8).

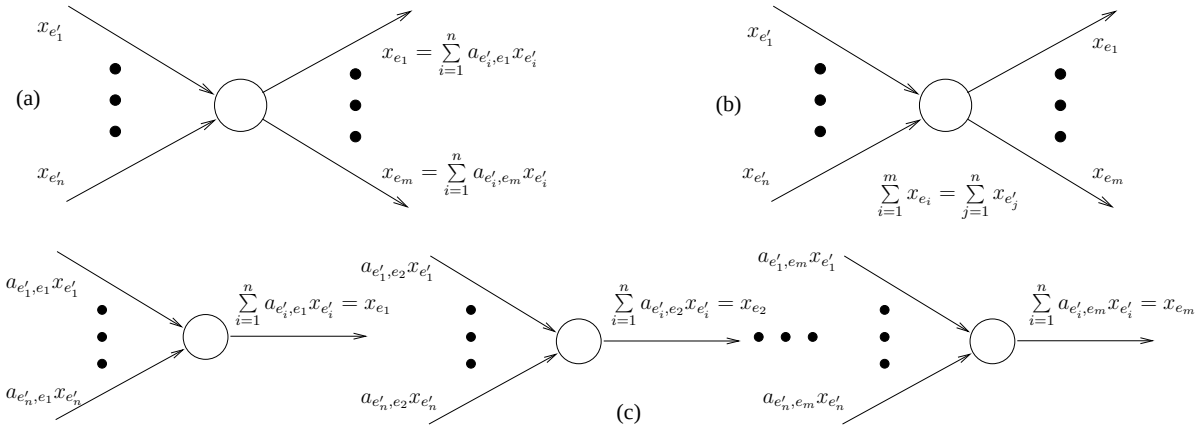

Fig. 2. (a)Picture of a node employing a linear network coding scheme. (b) Picture of a node for a group-valued circulation. (c) Transformation of node shown in (a).

IV. Group-valued Circulations

Consider a directed multigraph $G = (V, E)$ and a finite abelian group H. Then, a group-valued circulation or an H-circulation is defined as a mapping $f : E \to H$ such that the following equation is satisfied at each node $v \in V$:

$$\sum_{e:\text{head}(e)=v} f(e) = \sum_{e:\text{tail}(e)=v} f(e) \tag{1}$$

This can be understood as a conservation of flow – the sum of the symbols entering a node is equal to the sum of the symbols leaving it, with addition over H. This is shown in Fig. 2b.

Tutte's theorem (refer [8]) characterizes the number of group-valued flows in a given graph. The following Lemma is an adaptation of Tutte's theorem for counting the number of group-valued circulations in a given graph.

Lemma 2: The number of group-valued circulations on a given multigraph G is given by $|H|^m$ where $|H|$ is the group size and m is the number of edges left after contracting non-loop edges one after another till all edges in the graph are loops.

Proof: Let us assume first that all edges of $G = (V, E)$ are loops (head(e) = tail(e) $\forall$ $e \in E$). Then, given a finite abelian group H, every map $E \to H$ is an H-circulation on G. Hence, the number of circulations is given by $|H|^m$ where m is the number of loops.

Now assume that there is an edge, e_0 that is not a loop. Consider the multigraph G_1 which is same as G but with the edge e_0 contracted. It can be seen that the circulations on G_1 correspond bijectively to H-circulations on G [8, Thm. 6.3.1]. Thus, the non-loop edges can be contracted one by one till all the remaining edges are loops. At this point, the number of circulations can be reduced to $|H|^m$ where m is the number of edges (loops) left in the graph. ■

V. Network Information Flow as Group-valued Circulations

In a linear network coding scheme, each output link of a node carries a linear combination of the symbols received by that node through all its input links. Let us assume a coding scheme over a field F. Let $x_e \in F$ represent the symbol flowing through edge e. Then, the general form of the relation between the symbols flowing through one of the output links, $e \in O(v)$, and the set of input links, $I(v)$, of a node $v \in V$ can be written as:

$$x_e = \sum_{e' \in I(v)} a_{e',e} x_{e'} \tag{2}$$

where the coefficients $a_{e',e}$ are elements of F. This is shown in Fig. 2a. Note that the linear network-coded flow is not immediately a group-valued circulation. The conservation laws of Fig. 2a and 2b are not the same. In this section, we develop a transformation of the network, which results in an equivalence.

A. Equivalence for one node

Consider the node shown in Fig. 2a. There are a total of $|O(v)|$ linear equations that need to be satisfied at this node. Hence, an equivalent graph with group-valued circulations must have $|O(v)|$ copies of this node, each satisfying one of these equations. If e_i denotes the i-th output link of the node, the equation to be satisfied at the i-th copy of the node in the equivalent graph is given by:

$$x_{e_i} = \sum_{e' \in I(v)} a_{e',e_i} x_{e'} \tag{3}$$

Hence, the node shown in Fig. 2a will have to be replicated as many times as the number of its output links along with its entire set of input links. In addition, the flow in the input links will have to be appropriately scaled. After these transformations, the linear-network coded flow in one node shown in Fig. 2a is equivalent to the group-valued circulation in the graph shown in Fig. 2c.

B. Equivalence for network

We now extend the construction of the equivalent graph to the entire network. The single node transformation cannot be applied to all nodes simultaneously. A careful sequencing of nodes is necessary as discussed below.

In the transformation of a single node v, since all edges $e \in I(v)$ are replicated $|O(v)|$ times, the number of edges in $O(v')$, $v' \in \{\text{tail}(e) : e \in I(v)\}$ may increase. However, in order to apply the single node transformation on a node $v' \in G$, the complete set $O(v')$ needs to be known beforehand along with the symbols that should flow on each edge $e \in O(v')$. To solve this problem, we notice that $O(v')$ will not change once the single-node transformation discussed above has been applied to all nodes $v \in \{\text{head}(e) : e \in O(v')\}$. Hence, the transformation can be applied to v' only after it has been applied to all nodes $v \in \{\text{head}(e) : e \in O(v')\}$.

This sequencing can be achieved by applying the transformation in the topological order defined by the original directed acyclic network. A standard algorithm for finding such a topological ordering of the nodes is given below [9].

Algorithm 1: Topological Sorting

Input: A directed acyclic graph, $G = (V, E)$.

1) Associate with each node v, a value $N(v)$ that is initialized to $|O(v)|$.
2) Pick a node v such that $N(v) = 0$, do
   - For each edge $e \in I(v)$, $N(\text{tail}(e)) \leftarrow N(\text{tail}(e)) - 1$.
   - $N(v) \leftarrow -1$
   - Append v to the ordering, P.
3) If any node has not been added to the ordering yet, go to Step 2. Else terminate.

Output: P, a topological ordering (permutation) of the nodes such that there is no edge $v_i \rightarrow v_j$ if $i \leq j$.

Every iteration of Step 2 effectively removes one node from the graph. After this operation, the resultant graph is still a directed acyclic graph. Since every directed acyclic graph always has at least one node with out-degree 0, this algorithm will terminate once all nodes have been added to the ordering.

The final algorithm which takes a network coding problem and constructs an equivalent group-valued circulation network is given below:

Algorithm 2: Graph Transformation

Input: A directed acyclic graph $G = (V, E)$, set of sources S, set of sinks T, connection requirements $\mathcal{C}$.

1) Obtain a topological ordering P for the graph $G = (V, E)$ using Algorithm 1.
2) Let $G'(V', E') = G(V, E)$.
3) Loop through the nodes $v \in V$ in the order defined by P, do
   a) If $O(v) > 1$, for each edge $e \in O(v)$, do
      - Add a new node v' to V' with one output link connecting it to head(e) and one input link e' for each $e'' \in I(v)$ such that tail(e') = tail(e'').

Output: $G' = (V', E')$, a transformed network such that group-valued circulations in G' are equivalent to network-coded information flows in G.

Theorem 3: The final transformed network is made up of a set of directed trees. Each sink is the root of one tree. All leaf nodes are copies of one of the source nodes. Scaling is done only at the leaf nodes.

Proof: Each node in the transformed network will have exactly one output link and the acyclic property of the graph is maintained by the transformation. The underlying undirected graph is a set of disjoint trees, because any cycle in it must imply that either the cycle is also present in the directed graph or that one of the nodes in the directed graph has more than one output link. Hence, the equivalent network is made up of a set of directed trees.

The transformation maintains one output link for each node in the original graph that has $|O(v)| \geq 1$. So, the only nodes that will have $|O(v)| = 0$, and hence be the roots of these trees, are the sink nodes (which had $|O(v)| = 0$ to start with). Hence, each sink would be the root of a directed tree in which all edges are directed towards this root.

Also, the number of input links of a copied node in the transformed graph is equal to the number of input links possessed by the original node. So, the only nodes that will have $|I(v)| = 0$, and hence be leaf nodes in these trees, are copies of the source nodes (which had $|I(v)| = 0$ to start with). ■

From the above theorem, we see that the graph transformation can be applied starting from the sink and working towards the source. The topological sorting is a formal method that achieves such a sequencing.

An example of this transformation applied to the butterfly network (Fig. 3a) can be seen in Fig. 3b. From the theorem, we see that there will be four trees at the end of the transformation rooted at the sink nodes 7, 8, 9 and 10. Working up from Node 7 towards the source and copying necessary nodes, we see that the tree rooted at Node 7 in Fig. 3b results in a straightforward manner. Similarly, the other trees can be obtained. However, we point out later that this intuitive method might not be easy to implement on more complicated networks.

To apply the graph transformation algorithm formally, one possible topological ordering of the nodes is $7-8-9-10-5-6-4-3-1-2$. Nodes 7, 8, 9 and 10 are sink nodes, and occur first in the ordering. Nodes 5 and 6 will be replicated 2 times, since they both have 2 output links. This will result in the replication of the edges e_4, e_5, e_6 and e_7. Node 4 will now have 4 output links and will have to be replicated as many times along with edge e_3. Similarly, Node 3 will also be replicated 4 times along with edges e_1 and e_2. Finally, the source nodes 1 and 2 will be replicated 6 times each since they both now have 6 output links.

Note that the scaling variables in the equivalent group-valued circulation occur only at the leaf source nodes. All the intermediate nodes simply perform addition. This is the main reason for the simplification in the structure of our formulation. However, the flows in the different trees of the equivalent graph are not independent, since they share edges of the original graph. This dependence results in the degree-2 equations of our formulation.

C. Algebraic Formulation

We will describe and illustrate the formulation with the butterfly network (Fig. 3a) for simplicity. The generalization to arbitrary networks follows immediately.

1) Scaling variables: Let us now define one variable for each scalar (a_i's, b_i's) associated with each leaf node as illustrated in Fig. 3b for the butterfly network. The variable names

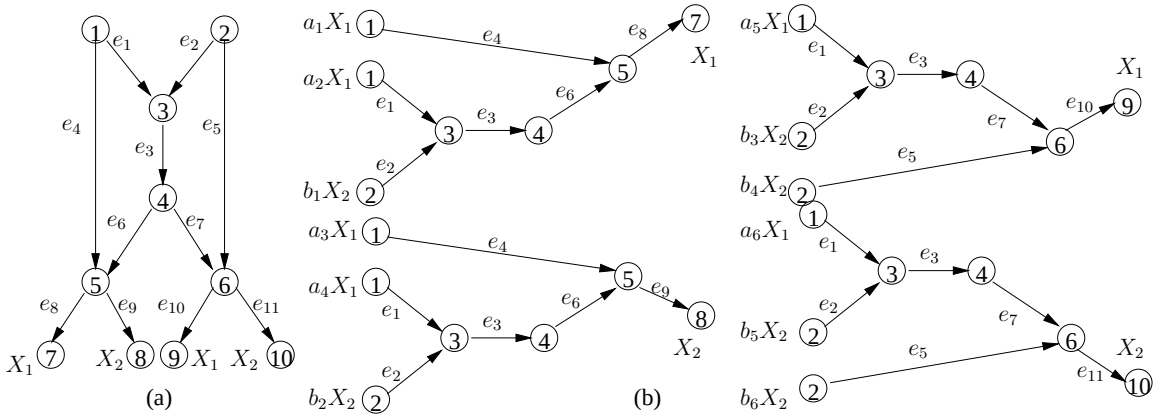

Fig. 3. (a) The (modified) butterfly network with 4 sinks and 2 sources. (b) The final transformed network in which group-valued circulations correspond to information flow in the original network. The transformed network has 4 disjoint trees - one for each each sink node. The appropriate scaling factors at the leaf nodes are denoted by a_i's and b_i's

are chosen as follows. Source nodes 1 and 2 are assigned the variable names a and b, respectively. The subscripts are chosen tree by tree in the transformed network. In the tree with root as Node 7, the two copies of source node 1 are assigned variables a_1 and a_2, while the single copy of source node 2 is assigned the variable b_1. In the tree with root node 8, the variables are a_3, a_4 for the two copies of Node 1, and b_2 for the single copy of Node 2. We continue in this manner to name the scaling variables at the source leaf nodes of the other two trees to get variables a_1, a_2, $\cdots$, a_6 and b_1, b_2, $\cdots$, b_6.

Once values are assigned to the scaling variables (from some field), all edge functions are defined in the transformed network. Hence, the network coding problem has been reduced to finding a feasible assignment of values to these unknown variables (scaling factors at leaf nodes) from a field such that the circulations generated in the transformed graph can be implemented as a code over the given network. The scaling variables need to satisfy two types of conditions as described in Sections V-C3 and V-C4 below for obtaining a valid network flow in the original network that meets the connection requirements.

2) Counting group-valued circulations: By applying the result in Lemma 2 to one particular tree obtained through the transformation, we can see that the number of edges (loops) left after contracting non-loop edges one after another is equal to the number of leaf nodes in the tree. But the symbols flowing from all the leaf nodes are not independent - they carry scaled versions of the input symbols generated at the source nodes in the original network.

A fixed assignment of values to these variables will produce different circulations for different values of X_1 and X_2 (symbols generated at the source nodes). This gives a total of H^2 ($|H|^{|S|}$ in general) circulations each of which corresponds to a unique set of values assumed by the source symbols, X_1 and X_2.

3) No Interference conditions: The required output at each sink must be received without any "interference" from other source symbols to meet the connection requirements. The symbol received at the root of one particular tree in the transformed graph is equal to the sum of the scaled versions of the source symbols flowing from the leaf nodes. This implies that, for each tree, the scalar constants pertaining to the required symbol at the sink must add up to 1 and those pertaining to every other symbol must add up to 0.

In Fig. 3b, the symbols (edge functions) received at the sink nodes 7, 8, 9 and 10 are $(a_1 + a_2)X_1 + b_1X_2$, $(a_3 + a_4)X_1 + b_2X_2$, $a_5X_1 + (b_3 + b_4)X_2$ and $a_6X_1 + (b_5 + b_6)X_2$, respectively. For the symbol at Node 7 to be equal to the required X_1, we have $a_1 + a_2 = 1$ and $b_1 = 0$. Other equations are derived similarly. Hence, in the butterfly network of Fig. 3, we get the following linear equations:

$$\begin{aligned} a_1 + a_2 &= 1 & \qquad b_1 &= 0 \\ a_3 + a_4 &= 0 & \qquad b_2 &= 1 \\ a_5 &= 1 & \qquad b_3 + b_4 &= 0 \\ a_6 &= 0 & \qquad b_5 + b_6 &= 1 \end{aligned} \tag{4}$$

It is clear that, since the flow on the tree is a group-valued circulation, the conservation law implies that intermediate nodes do not scale the symbols, but only perform addition. Note that the no-interference conditions produce linear equations, and no scaling variable appears in more than one linear equation. This is because the set of variables defined for each tree are mutually exclusive and, within each tree, the set of variables corresponding to each input symbol are also mutually exclusive. This surprising property results in a lot of simplifications in our formulation.

For completion, we state the general form of the no-interference conditions below. In general, each unknown scaling variable in the transformed network is associated with exactly one source symbol and one sink (or tree). Let us denote the unknown scalars by a_{ijk} where $i \in \{1, \ldots, |S|\}$ denotes the source symbol associated with the scalar, $j \in \{1, \ldots, |T|\}$ denotes the sink (or the tree) associated with the scalar, and $k \in \mathcal{N}$ is an index among all variables in the same tree associated with the same source symbol. Then, the general form of the "No Interference" conditions can be written as:

$$\sum_k a_{ijk} = \begin{cases} 1 & \text{if } Z(t_j) = X(s_i) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

4) Edge Compatibility conditions: The circulations in the different trees of the transformed network are not independent, because they contain copies of the same edge that can carry only one symbol at a time on the original network. Hence, in the transformed network, these edges must carry "compatible" symbols. The requirement is that symbols flowing through two copies of the same edge in the transformed network must be scalar multiples of each other.

For example, given a network with two source nodes producing symbols X_1 and X_2, suppose the transformed network has two copies of an edge e, denoted by e_1 and e_2, carrying symbols $a_1X_1+b_1X_2$ and $a_2X_1+b_2X_2$, respectively. In order to implement a network code on the original network, it is necessary and sufficient that the two edge functions on e_1 and e_2 are scalar multiples of each other. If they are not scalar multiples, it is clear that the original edge will need to carry two symbols per unit time in the original network, which is not possible. If they are scalar multiples, a network code can be obtained for the original network as shown later in Section VII. Intuitively, since scaling variables are allowed on links of the original network, scalar multiples in duplicated edges that arise from multiple outgoing links in the original graph can be accommodated by suitable scaling in different outgoing links.

Hence, the scalars a_1, a_2, b_1, b_2 of the edge functions of e_1 and e_2 need to satisfy the following condition:

$$a_1b_2 = a_2b_1$$

These type of degree-2 equations relate the scaling factors for every pair of symbols flowing through every pair of copies of an edge in different trees of the transformed network. Notice that the equation is equivalent to the fractional form $\frac{a_1}{a_2} = \frac{b_1}{b_2}$ modified to avoid division by zero problems when some variable takes the value zero. However, the fractional form is more intuitive and can be readily extended to obtain edge compatibility conditions when more than two sources are involved.

In our illustrative example of Fig. 3b, the edge e_3 is copied four times. Since there are $\binom{4}{2} = 6$ ways of choosing two copies among the four, there will be six edge compatibility conditions for e_3. The symbols on the copies of e_3 on the trees with root nodes 7, 8, 9 and 10 are $a_2X_1+b_1X_2$, $a_4X_1+b_2X_2$, $a_5X_1+b_3X_2$ and $a_6X_1+b_5X_2$, respectively. Hence, in fractional form, we need $\frac{a_2}{a_4} = \frac{b_1}{b_2}$ (roots 7 and 8), $\frac{a_2}{a_5} = \frac{b_1}{b_3}$ (roots 7 and 9), $\frac{a_2}{a_6} = \frac{b_1}{b_5}$ (roots 7 and 10), $\frac{a_4}{a_5} = \frac{b_2}{b_3}$ (8 and 9), $\frac{a_4}{a_6} = \frac{b_2}{b_5}$ (8 and 10) and $\frac{a_5}{a_6} = \frac{b_3}{b_5}$ (9 and 10).

In the degree-2 form, the edge compatibility conditions for the four copies of the edge e_3 are listed below:

$$\begin{aligned} a_2b_2 &= a_4b_1 & a_2b_3 &= a_5b_1 \\ a_2b_5 &= a_6b_1 & a_4b_3 &= a_5b_2 \\ a_4b_5 &= a_6b_2 & a_5b_5 &= a_6b_3 \end{aligned} \tag{6}$$

For the butterfly network example, we do not get any other edge compatibility conditions. For edges e_6 and e_7, the equations are identical to the ones listed above. Also, there are no equations for edges e_1, e_2, e_4 and e_5 since these edges have scaled versions of the same symbol flowing through them.

We see that the compatibility conditions can be simplified as not all of them are independent. However, since the variables can take the value zero, simplifying the equations needs to be done very carefully. Hence, we simply enumerate all equations at this stage and simplify later.

We have seen that not all duplicated edges result in distinct compatibility conditions. In general, edge compatibility equations will be required for each edge e in the original network that satisfies the following conditions:

1) Number of copies of head(e) in the transformed network > 1 (or the edge will not be replicated at all)
2) Number of different source nodes having a path to $e > 1$ (since if two copies of e carry a_1X_1 and a_2X_1, these will be scalar multiples of each other for any value assigned to a_1, a_2)
3) $|I(\text{tail}(e))| > 1$ (or the equations will be same as that for $e' \in I(\text{tail}(e))$)

For completion, we now state the general form of the edge-compatibility conditions in terms of nodes of the transformed network for added simplicity. Given a node $v \in V$ in the original network, the general form of the condition for two copies of v, denoted by v_1 and v_2, belonging to the j_1-th and j_2-th trees respectively in the transformed network can be written as:

$$\begin{aligned} &\left(\sum_{k\in h_{i_1j_1}(v_1)} a_{i_1j_1k}\right)\left(\sum_{l\in h_{i_2j_2}(v_2)} a_{i_2j_2l}\right) = \\ &\qquad \left(\sum_{m\in h_{i_1j_2}(v_2)} a_{i_1j_2m}\right)\left(\sum_{n\in h_{i_2j_1}(v_1)} a_{i_2j_1n}\right) \end{aligned} \tag{7}$$

where $h_{ij}(v)$ denotes the set of leaf nodes in the j-th tree that are copies of the source node s_i and have a path to v. A careful study of the general form shows an edge compatibility condition needs to be introduced for every two copies, $v_1, v_2 \in V'$, of node $v \in V$ and for every two sources $s_{i_1}, s_{i_2} \in S$ such that (a) $|I(v_1)| > 1$, (b) $v_1 \in V_{j_1}, v_2 \in V_{j_2},\ V_{j_i} =$ Set of nodes in the j_i-th tree, and (c) $h_{i_1j_1}(v_1) \neq \phi, h_{i_2j_1}(v_1) \neq \phi$.

5) Simplifying the equations: As pointed out before, the linear equations (No Interference conditions) possess the special property that each of them involves a mutually exclusive set of variables. Using this property, we can simplify the system of equations in the following two ways:

1) It is possible that some of the variables never occur in the non-linear equations (Edge Compatibility conditions). From (6), we can see that a_1 is one such variable in the example of the butterfly network. It can be easily seen that the linear equation involving a_1 can be trivially satisfied for any value assigned to the other variables involved in the same linear equation by choosing an appropriate value of a_1 (which does not have any other condition on it). Hence, a_1 along with the linear equation it occurs in can be removed from the system as trivially solvable.
   Therefore, the first simplification would involve elimination of variables (and their corresponding linear equations) that do not occur in any non-linear equation.

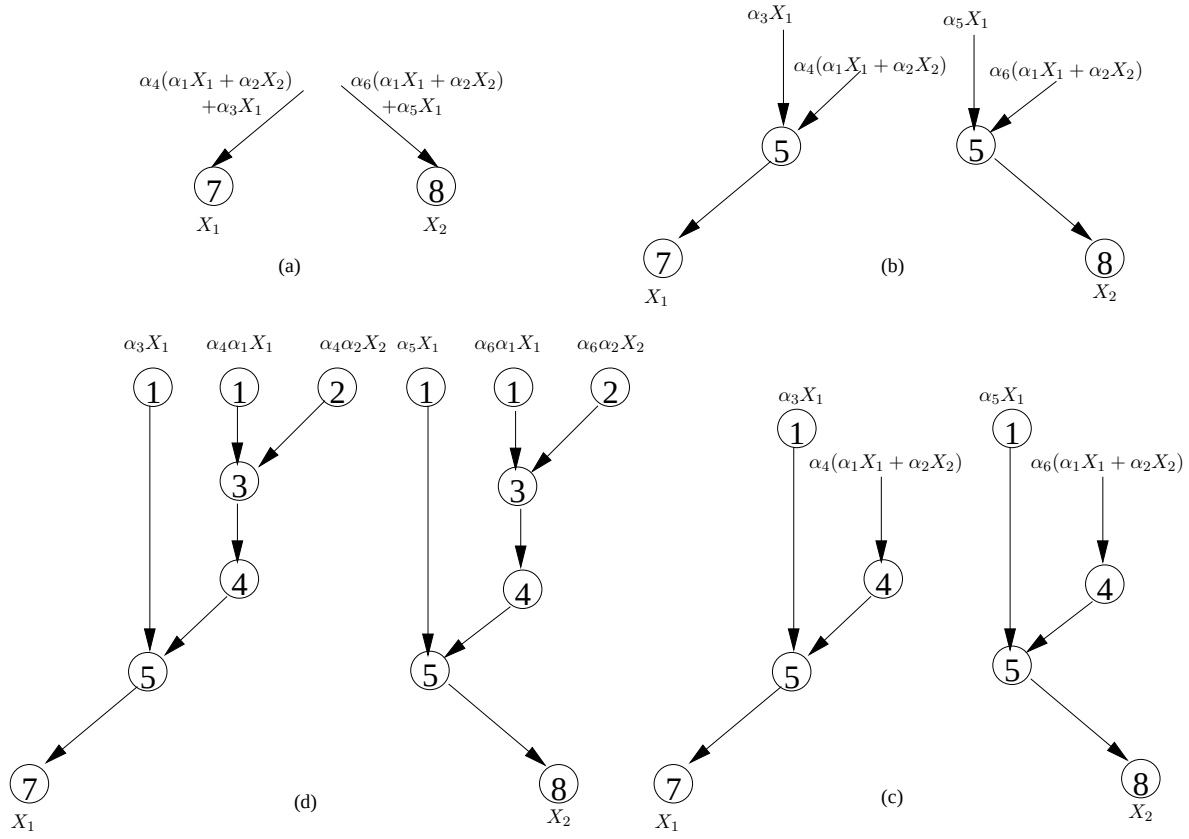

Fig. 4. Illustration of flow transformation.

2) Since each linear equation involves a mutually exclusive set of variables, we can eliminate one variable using each linear equation easily. Eliminating this variable from the non-linear equations (note that this does not increase the degree of the system) might reduce some of them to linear equations which can again be used to eliminate more variables iteratively.

In the case of the butterfly network, after the first step of simplification, we are left with 8 variables, 4 linear equations and 6 non-linear equations.

In the second step of the simplification, after the first round of elimination of variables using the linear equations (4) in (6), we are left with 4 variables: a_2, a_4, b_3 and b_5 and the 6 equations as shown below.

$$\begin{aligned} a_2 &= 0 & b_5 &= 0 \\ a_2b_5 &= 0 & a_2b_3 &= 0 \\ a_4b_5 &= 0 & a_4b_3 &= 1 \end{aligned}$$

Subsequently, a_2 and b_5 can also be eliminated, using the linear equations above, leaving just 2 variables and the relation:

$$a_4b_3 = 1 \tag{8}$$

Hence, the network coding problem for the example of the butterfly network has been reduced to solving only one (non-trivial) equation given in (8).

To summarize, our main result has been to show that network information flow in a given network is equivalent to group-valued circulations in a transformed network (a set of directed trees, one for each sink). We have used this equivalence to derive a set of polynomial equations (with maximum degree 2) which provide a new and simple formulation for the scalar linear network coding problem.

VI. Examples and Comparison

A. Illustration

To further clarify the graph and flow transformation, we illustrate, in Fig. 4, the steps of the transformation for the flow in the butterfly network of Fig. 1 to the sinks at Node 7 and Node 8.

In Fig. 4a, we start with the flows to the sink nodes 7 and 8. The scaling factors are eliminated and moved one node up in Fig. 4b by copying Node 5. The same process continues in Figs. 4c and 4d till we get two trees and the scaling factors are at the leaf nodes. At this point, we have a group-valued circulation in the trees.

Notice that the relationship between the scaling variables in our formulation, shown in Fig. 3b, and the direct formulation (Fig. 1) can now be seen readily. For instance, $a_1 = \alpha_3$, $a_2 = \alpha_4\alpha_1$, $b_1 = \alpha_4\alpha_2$, $a_3 = \alpha_5$, $a_4 = \alpha_6\alpha_1$, $b_2 = \alpha_6\alpha_2$. This, along with similar relationships for nodes 9 and 10, is the substitution of variables that results in our formulation from the direct formulation. Our algorithm for graph transformation along with the no interference and edge compatibility conditions perform this substitution implicitly resulting in linear and degree-2 equations with possibilities for simplification. Finally,

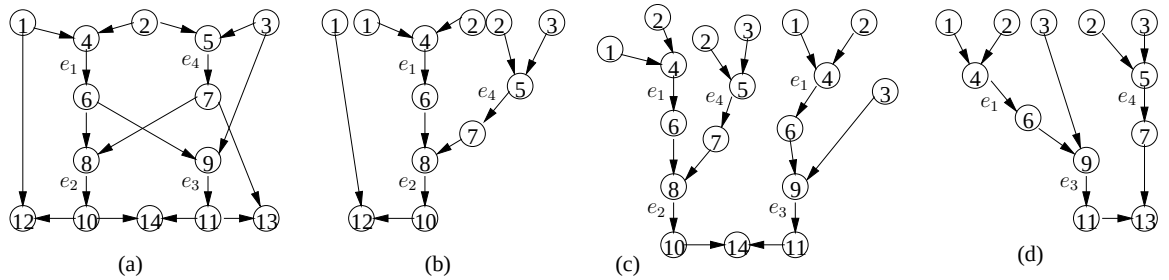

Fig. 5. (a) An example network that is solvable only over fields with characteristic 2. There are three sources - 1, 2 and 3 - producing symbols X_1, X_2 and X_3 respectively. There are three sinks - 12, 13 and 14 - demanding symbols X_3, X_1 and X_2 respectively. (b),(c),(d) The final transformed network with 3 trees - one for each sink node.

we obtain the simple equation, $a_4b_3 = 1$, which is not obvious even when the substitution is clearly specified.

B. Another Example

Consider the network shown in Fig. 5a taken from [5], [6], where it has been proved to have linear coding solutions only over fields of characteristic 2. Nodes 1, 2 and 3 are sources producing X_1, X_2 and X_3 respectively. Nodes 12, 13 and 14 are sinks demanding X_3, X_1 and X_2 respectively. The trees in the equivalent group-valued circulation network are shown in Fig. 5b,c,d. Notice that the intuitive method of starting with the sink and moving up towards the source for performing the graph transformation needs care in its execution for this example.

The set of equations generated by the "No Interference condition" are:

$$\begin{aligned}
&\text{Node 12: } a_1 + a_2 = 0; b_1 + b_2 = 0; c_1 = 1\\
&\text{Node 13: } a_3 = 1; b_3 + b_4 = 0; c_2 + c_3 = 0\\
&\text{Node 14: } a_4 + a_5 = 0; b_5 + b_6 + b_7 = 1; c_4 + c_5 = 0
\end{aligned} \quad (9)$$

The set of equations generated by the "Edge Compatibility condition" for edges e_1, e_2, e_3 and e_4 respectively are:

$$\begin{aligned}
e_1 &: a_2b_3 = a_3b_1; a_2b_5 = a_4b_1; a_2b_7 = a_5b_1;\\
&\quad a_3b_5 = a_4b_3; a_3b_7 = a_5b_3; a_4b_7 = a_5b_5\\
e_2 &: a_2(b_5 + b_6) = a_4(b_1 + b_2); a_2c_4 = a_4c_1;\\
&\quad (b_1 + b_2)c_4 = (b_5 + b_6)c_1\\
e_3 &: a_3b_7 = a_5b_3; a_3c_5 = a_5c_2; b_3c_5 = b_7c_2\\
e_4 &: b_2c_3 = b_4c_1; b_2c_4 = b_6c_1; b_4c_4 = b_6c_3
\end{aligned} \quad (10)$$

Using the linear equations to eliminate variables iteratively, we get 9 equations in 6 variables shown below.

$$\begin{aligned}
&a_2b_3 = b_1;\ a_2 = -a_4b_1;\ a_4b_3 = -1;\\
&a_2c_4 = a_4;\ c_4 = a_4c_2;\ b_3c_4 + c_2 = 0;\\
&b_1c_2 + b_3 = 0;\ b_1c_4 + 1 = 0;\ b_3c_4 = c_2
\end{aligned} \quad (11)$$

From equations $b_3c_4 + c_2 = 0$ and $b_3c_4 = c_2$, we can derive the relation $2c_2 = 0$. Substituting $c_2 = 0$ in the above system leads to the condition $1 = 0$ which is not possible. Hence, we must have $2 = 0$, which implies that the system is not solvable in any field with an odd characteristic. Also, in characteristic 2, setting all variables to 1 in the above equations, is seen to be a solution. This example demonstrates that, in practice, working with the equations derived through our formulation can be advantageous.

For this example, the direct formulation of [3], as illustrated in [6], results in 17 equations in 22 variables.

TABLE I
COMPARISON OF FORMULATIONS

Example	*Group-valued Circulations* Var.[1]	Deg. 2 Eqns[1]	*Direct Formulation* Var.	Eqns	Deg.
Butterfly	4	6	10	8	2
Fig. 5a	8	15	14	9	3
[3, Fig. 5]	9	5	14	4	4
[5, Fig. 3]	27	45	50	32	3
[6, Fig. 3]	12	30	22	17	3

[1]After one iteration of elimination using the linear equations

C. Complexity comparison

It has been shown in [10] that the complexity of Gröbner Basis algorithms depends, among other things, on the maximum degree of the starting basis. The degrees of the intermediate polynomials computed during Gröbner Basis calculations has been shown to grow up to 2^{2^d} if the maximum degree of the starting basis is d. Due to these issues, Gröbner Basis algorithms become practically intractable except for small problem instances.

In the light of these results, our work on deriving a system of polynomial equations whose maximum degree is only 2 becomes important and will help in reducing the running complexity of Gröbner Basis algorithms that may be used to solve the system.

A comparison between the number of variables, equations and maximum degree of the the system of polynomial equations derived based on our formulation described in the previous section and the formulation proposed by Koetter *et al* in [3] is shown in Table I. Appendix A explains how to count the number of variables and equations obtained in our formulation. It can be seen that, apart from having a maximum degree of only 2, the number of variables is also lesser in our new formulation for every case. Also, the number of equations is more than the number of variables in most cases for our formulation which makes it viable to use special algorithms such as [11] for solving the derived system of equations.

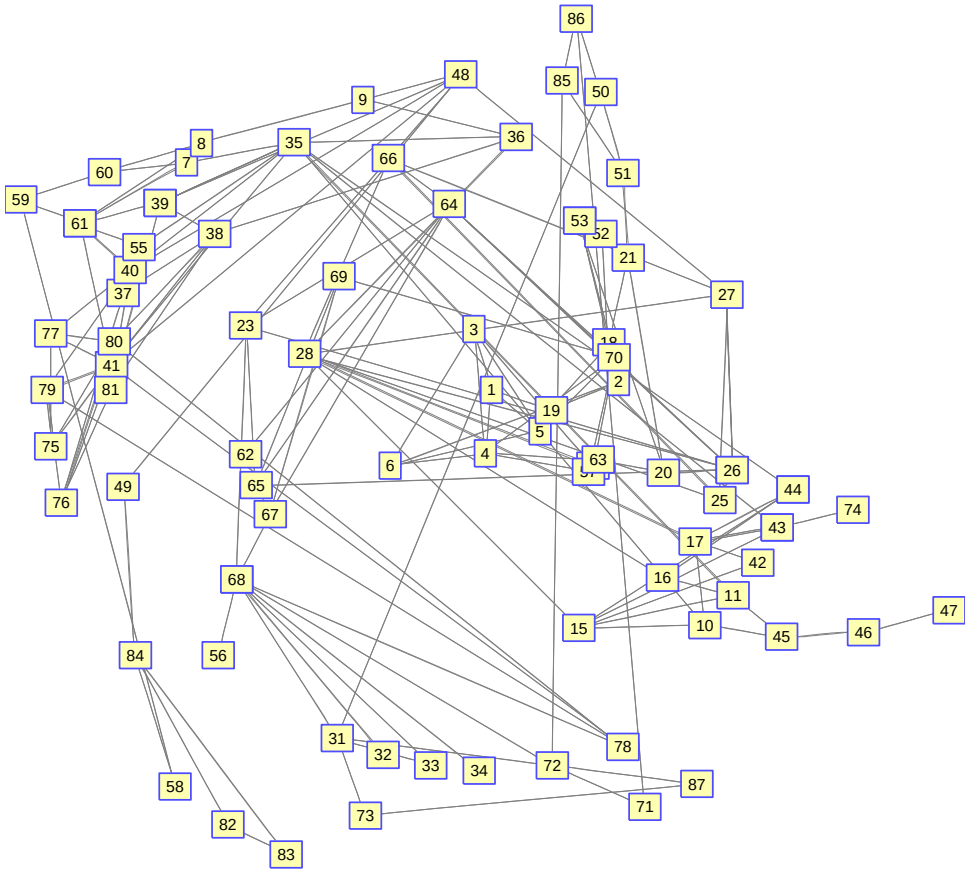

Fig. 6. An ISP network over Europe with 87 nodes and 161 edges.

D. A Bigger Example

Consider an ISP network topology shown in Fig. 6 taken from [12]. The network has 87 nodes and 161 edges. Assuming all links have unit capacity, we tried the following three cases with different number of sources on this network. Further, we assumed characteristic 2 in our simplification steps. The five nodes 31, 41, 47, 69 and 82 were set as sources in all the following cases. Each sink demands data from one of the source nodes. The sinks and their demands were chosen at random depending on graph connectivity.

1) *5 sources (all rate 1) and 10 sinks.* The direct formulation (from [3]) gives a system of 44 equations in 30 variables. Our formulation initially results in 44 linear equations and 3 degree-2 equations in 316 variables. After applying the simplification steps, we are left with only 3 degree-2 equations in 7 variables assuming solution exists in a characteristic 2 field. In fact, setting all the remaining 7 variables to zero results in a valid solution to the three equations (some other scaling variables are non-zero). Hence, a solution over GF(2) is possible.
2) *5 sources (one with rate 2, others rate 1) and 12 sinks.* The direct formulation yields a system of 50 equations in 40 variables in this case. In comparison, our formulation initially resulted in 50 linear equations and 34 degree-2 equations in 330 variables. But after applying the simplification steps, we are left with only 13 degree-2 equations in 17 variables assuming solution exists in a characteristic 2 field. Again the all-zero solution is valid for the remaining 17 variables resulting in a network code over GF(2).
3) *5 sources (all with rate 2) and 11 sinks.* The direct formulation yields a system of 88 equations in 180 variables. Our formulation initially gives 88 linear and 11198 degree-2 equations in 632 variables. But on applying the simplification steps, assuming characteristic 2, it turns out that the system is not solvable over characteristic 2.

Hence, we see that the algebraic formulation of scalar linear network coding based on group valued circulations appears to work better even over large networks with a few sources and sinks.

VII. Network Code from Group-valued Circulations

We now describe an algorithm to obtain a network code for the original network from the group-valued circulations on the transformed network. Note that this completes the proof of the sufficiency of edge compatibility conditions.

First, we will briefly describe the algorithm and then present a notational version of the same. A solution to the system of polynomial equations in our formulation consists of a set of values assigned to the scaling variables at the leaf source nodes in the group-valued circulation network such that the

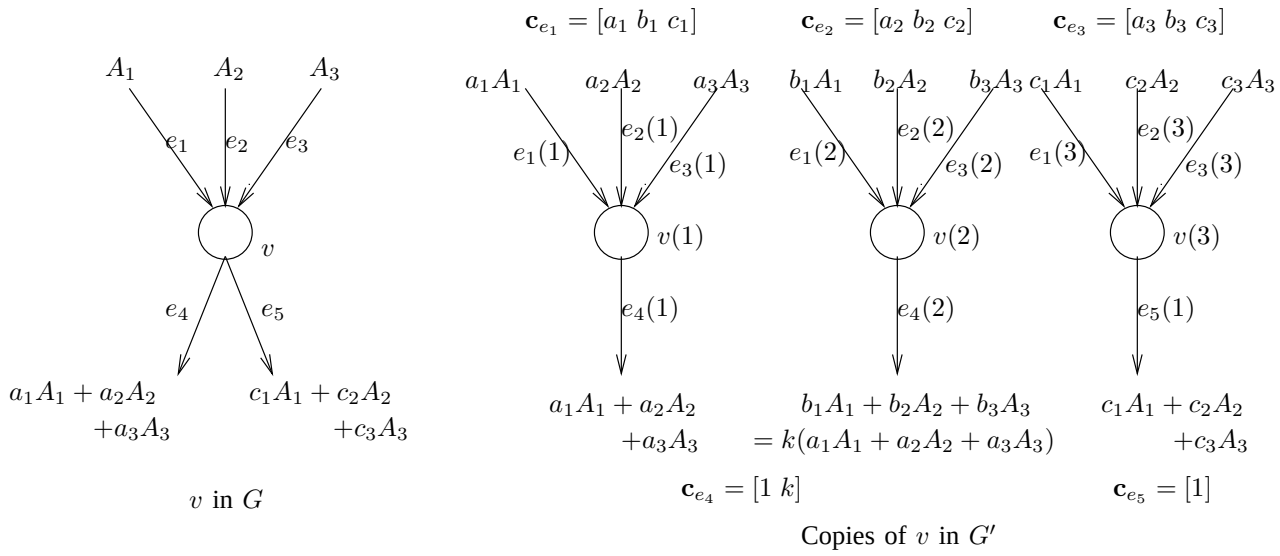

Fig. 7. Determining the vectors $\mathbf{f}$ and $\mathbf{c}$ for outgoing links

no interference conditions as well as the edge compatibility conditions are satisfied. The algorithm to construct a network code from such a solution consists of propagating the values of these coefficients from the source nodes to the sink nodes through the transformed network.

We compute two vectors for every edge e of the graph $G = (V, E)$. The first vector $\mathbf{f}_e = [f_e(1)\ f_e(2)\ \cdots\ f_e(|S|)]$ represents the edge function or symbol $\sum_{i=1}^{|S|} f_e(i)X_i$ sent over edge e. Suppose e is replicated n times to obtain edges e_i'', $1 \leq i \leq n$ in the transformed graph $G' = (V', E')$. The second vector $\mathbf{c}_e = [c_1\ c_2\ \cdots\ c_n]$ is such that the edge function on $e_i'' \in E'$ is $c_i \sum_{i=1}^{|S|} f_e(i)X_i$. Note that such a scaling property is guaranteed for all copies of an edge by the compatibility conditions. Once the vectors $\mathbf{f}_e$ are computed for all $e \in E$, the network code in G is completely known.

Suppose $\mathbf{f}_{e'}$ and $\mathbf{c}_{e'}$ are known for all the incoming edges $e' \in I(v)$ for a node $v \in V$. The vectors $\mathbf{f}_e$ and $\mathbf{c}_e$ can be computed for the outgoing edges $e \in O(v)$ as illustrated for a sample case in Fig. 7. In the figure, a node $v \in V$ with $I(v) = \{e_1, e_2, e_3\}$ and $O(v) = \{e_4, e_5\}$ is replicated thrice into $v(1)$, $v(2)$ and $v(3)$ in G'. The incoming and outgoing links are replicated as shown. For instance, the edge e_1 is replicated thrice as $e_1(1)$, $e_1(2)$ and $e_1(3)$. Suppose there are three source nodes $S = \{s_1, s_2, s_3\}$, and $\mathbf{f}_{e_i} = [\alpha_{i1}\ \alpha_{i2}\ \alpha_{i3}]$ resulting in edge functions $A_i = \sum_{j=1}^{3} \alpha_{ij}X_j$ for $i = 1, 2, 3$. The scaling vectors $\mathbf{c}_{e_i}$ are as shown in the figure.

Using the edge functions and scaling factors on the incoming edges, the edge function of the copies of e_i, $i = 1, 2, 3$ are computed first. For instance, the edge function of $e_2(2)$ is computed as b_2A_2. Then, the edge function for the outgoing links of $v(1)$, $v(2)$ and $v(3)$ in G' are computed by simple addition. As shown in the figure, the symbols sent on $e_4(1)$ and $e_4(2)$ will be scalar multiples. We then assign the symbol on e_4 in G to be the symbol on $e_4(1)$ given by $\sum_{i=1}^{3} a_iA_i = \sum_{j=1}^{3}(\sum_{i=1}^{3} a_i\alpha_{ij})X_j$ (assumed nonzero). Then, $\mathbf{f}_{e_4}$ and $\mathbf{c}_{e_4}$ are assigned suitably.

In this manner, all the nodes are processed in a suitable order to compute the network code for the original graph from the group-valued circulation on the transformed graph. We now introduce some notation to describe the algorithm formally.

A. Notation

Consider the given network $G = (V, E)$ and the equivalent group-valued circulation network $G' = (V', E')$. Then, for each node $v \in V$, let us define the set of network coding coefficients as $a_{e',e}\ \forall\ e' \in I(v), e \in O(v)$ i.e. if $x_{e'}$ is the symbol received on the link $e' \in I(v)$, the symbol sent on $e \in O(v)$ is $\sum_{e' \in I(v)} a_{e',e}x_{e'}$ (see Fig. 2).

Nodes and edges get replicated during the transformation from G to G'. We define some sets to hold information about the replicated nodes and edges. For $v \in V$ ($v \notin S \cup T$) and $e, e' \in E$, define:

$$R_v = \{v' \in V' : v' \text{ is a copy of } v\}$$
$$R_e = \{e'' \in E' : e'' \text{ is a copy of } e\}$$
$$R_{e',e} = \{e'' \in R_{e'} : \text{head}(e'') \in R_{\text{tail}(e)}\}$$

The sets R_v and R_e hold nodes and edges in G' that are copies of v and e , respectively. Two other useful sets are (1) $R_{\text{head}(e)}$ that contains copies of head(e) satisfying the relation $R_{\text{head}(e)} = \text{head}(R_e)$, and (2) $R_{\text{tail}(e)}$ that contains copies of tail(e). The set $R_{e',e}$ contains copies of an edge e' that connect to a copy of e.

Let the vector $\mathbf{f}_e = [f_e(1)\ f_e(2)\ \cdots\ f_e(|S|)]$ represent the edge function $\sum_{i=1}^{|S|} f_e(i)X_i$ sent over edge $e \in E$ in the final linear network code in G. Since the edge compatibility conditions are satisfied, the edge function on each copy of e in R_e will be a scalar multiple of $\mathbf{f}_e$. For $e'' \in R_e$, let the edge function on e'' be $\mathbf{f}_{e''} = c_e(e'')\mathbf{f}_e$. We collect the multiplying factors $c_e(e'')$, $e'' \in R_e$ into a vector $\mathbf{c}_e = [c_e(e'') : e'' \in R_e]$. Note that there is a one-to-one correspondence between elements of the sets $R_{\text{head}(e)}$ and $\mathbf{c}_e$ given by $c_e(e'') \leftrightarrow \text{head}(e'')$ for $e'' \in R_e$. Finally, we define sub-vectors $\mathbf{c}_{e',e} = [c_{e'}(e'') : e'' \in R_{e',e}]$ collecting the multiplying factors on copies of e' that connect to e.

B. The Algorithm

The vectors $\mathbf{f}_e$ and $\mathbf{c}_e$ are initialized for an outgoing link e from the source node as follows. For the i-th source node $s_i \in S$ and $e \in O(s_i)$, $\mathbf{f}_e = [0^{i-1}\ 1\ 0^{|S|-i}]$. For $e'' \in R_e$, the coordinate $c_e(e'')$ of $\mathbf{c}_e$ is equal to the value of the scaling variable at the leaf node tail$(e'') \in R_{s_i}$.

Algorithm 3: Deriving the Network Code

Input: A directed acyclic network $G = (V, E)$, an equivalent group-valued circulation network $G' = (V', E')$, a topological ordering of nodes P (from Algorithm 1) and a solution to the derived system of polynomial equations.

For each node, v in the reverse topological ordering, P', of P, if $v \notin S \cup T$, do

1) Get $\mathbf{f}_{e'}, \mathbf{c}_{e'}$ from tail$(e')\ \forall\ e' \in I(v)$.
2) For each edge $e \in O(v)$
   a) Get $\mathbf{c}_{e',e}$ from $\mathbf{c}_{e'}$ as defined above $\forall\ e' \in I(v)$.
   b) $F_{e',e} \leftarrow \mathbf{c}_{e',e}^T \mathbf{f}_{e'}\ \forall\ e' \in I(v)$, are matrices such that each row corresponds to the symbol flowing through a copy of edge e in G' due to the flow through a copy of edge e'.
   c) $F_e \leftarrow \sum_{e' \in I(v)} F_{e',e}$, is a matrix such that each row corresponds to the symbol flowing through a copy of edge e in G'.
   d) $\mathbf{f}_e \leftarrow$ any non-zero row (say, i) of F_e, or the zero row if F_e is the zero matrix. This is the symbol that will actually flow through e in G.
   e) $a_{e',e} \leftarrow \mathbf{c}_{e',e}(i)\ \forall\ e' \in I(v)$, where i is the row selected in the previous step. This is the set of network coding coefficients of node v corresponding to output link e.
   f) $\mathbf{c}_e(j) \leftarrow (j^{th}$ row of $F_e)/\mathbf{f}_e$ or 0 if $\mathbf{f}_e = \mathbf{0}\ \forall\ j = 1, \ldots, |\mathbf{c}_e|$.

The decoding coefficients at a sink node t_j are given by the set $\{\mathbf{c}_e; e \in I(t_j)\}$. Note that all the matrices in this set have only one element since there is only one copy of each sink node (and and all its input links) in G'.

Output: The set of all network coding coefficients, $a_{e',e}$, for the given network.

C. An Example

We will now present an example of this algorithm applied to a sample solution for the modified butterfly network (Fig. 3). Consider the following solution for the system over $GF(4) = \{0, 1, \alpha, \alpha^2\}$, $\alpha^2 = 1 + \alpha$.

$$\begin{aligned} a_1 = a_5 = b_2 = b_6 = 1 \\ a_2 = a_6 = b_1 = b_5 = 0 \\ a_3 = a_4 = \alpha \\ b_3 = b_4 = \alpha^2 \end{aligned} \tag{12}$$

One reverse topological order of edges is $1-2-3-4-5-6-7-8-9-10$. Nodes 1,2 are source nodes. So, we have $\mathbf{f}_{e_1} = \mathbf{f}_{e_4} = [1\ 0]$, $\mathbf{f}_{e_2} = \mathbf{f}_{e_5} = [0\ 1]$ and from the solution above, we have $\mathbf{c}_{e_1} = [0\ \alpha\ 1\ 0]$, $\mathbf{c}_{e_4} = [1\ \alpha]$, $\mathbf{c}_{e_2} = [0\ 1\ \alpha^2\ 0]$, $\mathbf{c}_{e_5} = [\alpha^2\ 1]$.

Then, beginning with the iteration for the non-source non-sink nodes as described in the algorithm above, we will first process node 3 - we know $\mathbf{f}_e, \mathbf{c}_e$ for both its input links, e_1, e_2. There are 4 copies of this node as shown in Fig. 3b and all 4 copies have copies of edge e_3 as their only output links. Hence, $\mathbf{c}_{e_1,e_3} = \mathbf{c}_{e_1}$ and $\mathbf{c}_{e_2,e_3} = \mathbf{c}_{e_2}$. After computing F_{e_1,e_3}, F_{e_2,e_3}, we arrive at:

$$F_{e_3} = \begin{pmatrix} 0 & \alpha & 1 & 0 \\ 0 & 1 & \alpha^2 & 0 \end{pmatrix}^T$$

Now, let $\mathbf{f}_{e_3} = [\alpha\ 1]$, the second row of F_{e_3}. Hence, we have $a_{e_1,e_3} = \alpha, a_{e_2,e_3} = 1$, the network coding coefficients at node 3. Also, $\mathbf{c}_{e_3} = [0\ 1\ \alpha^2\ 0]$.

Next, we will move to node 4 which has two output links, e_6, e_7. Hence, we have $\mathbf{c}_{e_3,e_6} = [0\ 1], \mathbf{c}_{e_3,e_7} = [\alpha^2\ 0]$. We can then compute F_{e_3,e_6}, F_{e_3,e_7} and then arrive at:

$$F_{e_6} = \begin{pmatrix} 0 & 0 \\ \alpha & 1 \end{pmatrix}$$

$$F_{e_7} = \begin{pmatrix} 1 & \alpha^2 \\ 0 & 0 \end{pmatrix}$$

Now we can choose $\mathbf{f}_{e_6} = [\alpha\ 1]$, the second row of F_{e_6} and $\mathbf{f}_{e_7} = [1\ \alpha^2]$, the first row of F_{e_7}. Then, the network coding coefficients for node 4 are $a_{e_3,e_6} = 1, a_{e_3,e_7} = \alpha^2$. Completing the last step of the iteration, we get $\mathbf{c}_{e_6} = [0\ 1], \mathbf{c}_{e_7} = [1\ 0]$.

Now we come to node 5. For the output link e_8, we have $\mathbf{c}_{e_4,e_8} = [1], \mathbf{c}_{e_6,e_8} = [0]$ and for e_9, we have $\mathbf{c}_{e_4,e_9} = [\alpha], \mathbf{c}_{e_6,e_9} = [1]$. Then we get:

$$F_{e_8} = [1\ 0]$$
$$F_{e_9} = [0\ 1]$$

So, $\mathbf{f}_{e_8} = [1\ 0]$ and $\mathbf{f}_{e_9} = [0\ 1]$, the only rows of the respective matrices. Then, the network coding coefficients for node 5 are:

$$a_{e_4,e_8} = 1, a_{e_6,e_8} = 0$$
$$a_{e_4,e_9} = \alpha, a_{e_6,e_9} = 1$$

Also, $\mathbf{c}_{e_8} = \mathbf{c}_{e_9} = [1]$.

Similarly, the network coding coefficients for node 6 can also be computed so that sinks 9, 10 receive the required symbols.

VIII. Conclusion

In this work, we have established an equivalence between network information flows and group-valued circulations and used the equivalence to arrive at an algebraic formulation for the network coding problem that is different from the one first proposed by Koetter *et al* in [3]. Given a network coding problem, we have given algorithms to construct an equivalent group-valued circulation network and to arrive at a system of polynomial equations (of maximum degree 2) that represents the scalar linear network coding problem. We have demonstrated the computational advantages of our new algebraic formulation over the traditional approach.

The obvious intuitive connection between network coding and group-valued circulations is quite interesting in itself and lends a new perspective to the network coding problem. We have explored one aspect of this connection by arriving at an alternative algebraic formulation for the problem. Other

aspects and applications of this connection are yet to be explored.

Since our system of equations is equivalent to the one in [3], an alternative way of viewing our formulation is that we simplify the equations in [3] through a recursive graphical method.

When seen in conjunction with [6], an interesting result is obtained from our formulation. In [6], the authors show that solvability of an arbitrary polynomial collection is equivalent to solvability of a network. Our formulation equates the solvability of a network to the solvability of a degree-2 polynomial collection. Hence, we infer that solvability of an arbitrary system of polynomials is equivalent to solvability of a degree-2 polynomial collection obtained by our formulation.

APPENDIX A
COUNTING NUMBER OF VARIABLES AND EQUATIONS

A. Number of variables

It can be seen that the number of variables involved in the system of polynomial equations derived in Section V-C is equal to the number of leaves in the set of trees in the transformed network (which in turn is equal to the number of paths from any of the source nodes to any of the sink nodes). A dynamic programming algorithm for computing this number is as follows.

Algorithm 4: Counting number of variables
Input: A directed acyclic graph, $G = (V, E)$ and a topological ordering of nodes, P (from Algorithm 1).

1) Associate with each sink node $v \in T$, a value $C(v) = 1$.
2) For each node in the topological ordering P do,
   - $C(v) \leftarrow \sum_{e \in O(v)} C(\text{head}(e))$
3) Number of variables = $\sum_{v \in S} C(v)$

Output: The number of variables in the system of equations.

B. Number of linear equations

As shown in Section V, the graph transformation yields as many trees as the number of sink nodes. For each tree, we have as many linear equations as the number of different source nodes that have a path to the associated sink node - one equation for the symbol that is demanded and one equation each for the symbols that are not demanded.

Using the notation used in (7), for a node $v \in V'$ in the transformed network, $h_{ij}(v)$ denotes the set of leaf nodes in the j-th tree of the transformed network that are copies of the source node s_i and have a path to v. Also, let V_j denote the set of nodes in the j-th tree. Let us now define:

$$S(v) = \{s_i : s_i \in S, |h_{ij}(v)| > 0\} \text{ where } v \in V_j \qquad (13)$$

In other words, $S(v)$ is the set of source nodes that have a path to the original copy of the node v in the given network. Hence, the total number of linear equations is given by $\sum_{j=1}^{|T|} |S(t_j)|$.

C. Number of non-linear equations

As seen from (7), we have one non-linear equation for every pair of source symbols flowing through every pair of replicated copies of any edge e such that $|I(\text{tail}(e))| > 1$. The number of copies created of an edge e is given by $C(\text{head}(e))$ computed as part of Algorithm 4. A similar dynamic programming algorithm as Algorithm 4 applied in the reverse topological order of nodes (from Algorithm 1) can be used to compute $S(\text{tail}(e))$, the number of different source symbols flowing through an edge e.

Now, the number of non-linear equations associated with this edge e is given by:

$$Q(e) = \binom{C(\text{head}(e))}{2} \binom{S(\text{tail}(e))}{2}. \qquad (14)$$

Hence, the total number of non-linear equations is given by $\sum_{e \in E} Q(e)$.

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow", *IEEE Trans. Inform. Theory*, vol. 46, pp. 1204–1216, July 2000.
[2] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding", *IEEE Trans. Inform. Theory*, vol. 49, pp. 371, February 2003.
[3] R. Koetter and M. Médard, "An algebraic approach to network coding", *IEEE/ACM Trans. Networking*, vol. 11, pp. 782–794, October 2003.
[4] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction", *IEEE Trans. Inform. Theory*, vol. 51, pp. 1973–1982, June 2005.
[5] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of linear coding in network information flow", *IEEE Trans. Inform. Theory*, vol. 51, pp. 2745–2759, August 2005.
[6] R. Dougherty, C. Freiling, and K. Zeger, "Linear network codes and systems of polynomial equations", *IEEE Trans. Inform. Theory*, vol. 54, pp. 2303–2316, May 2008.
[7] D. Sharma, "Linear network coding for wirelined and wireless networks", Master's thesis, Indian Institute of Science, April 2007.
[8] R. Diestel, *Graph Theory (Graduate Texts in Mathematics), Third Edition*, Springer-Verlag, 2000.
[9] Narsingh Deo, *Graph Theory with Application to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[10] H. M. Möller and F. Mora, "Upper and lower bounds for the degree of gröbner bases", *Lec. Notes in Comp. Sci.*, vol. 174, pp. 172–183, 1984.
[11] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations", in *Lec. Notes in Comp. Sci.* EUROCRYPT, 2000, pp. 392–407.
[12] T. Anderson, R. Mahajan, N. Spring, and D. Wetherall, "Rocketfuel project", http://www.cs.washington.edu/research/networking/rocketfuel.