Distributed Asynchronous Algorithms for Multicast Network Coding

Tracey Ho, Ben Leong, Ralf Koetter and Muriel Médard

***Abstract*— We propose distributed asynchronous algorithms for network coding in multi-source multicast networks with cycles. Our algorithms find an acyclic subset of connections between links, and set up a network code over these connections. They offer advantages in terms of link usage and coding complexity over previous distributed algorithms based on random coding, while requiring modest additional coordination.**

I. Introduction

The distributed randomized network coding approach of [1], [2], [3] provides a simple distributed way to do network coding on a given set of links for multi-source multicast. Each link transmits a random linear combination, in a finite field, of data received from incident incoming links. This approach, which codes maximally over all available capacity and involves minimal coordination among network nodes, can be viewed as lying at one end of a spectrum trading off coordination overhead against resource usage and the degree/complexity of network coding. At the other end of the spectrum, centralized approaches and synchronous/iterative decentralized techniques have been proposed for finding good network coding solutions that optimize resource usage [4], [5].

This paper considers decentralized asynchronous techniques that represent a start in bridging the gap between these extremes. Such approaches are potentially useful in various practical scenarios, such as wireless ad hoc or sensor networks. We note that network coding is primarily useful in scenarios where network capacity is a limited or costly resource; where network capacity is plentiful, routing approaches present a reasonable alternative.

One significant practical issue is dealing with cycles in networks. Many existing works generally assume that an acyclic graph is given, or centrally construct an acyclic subgraph [6] or related graph [7]. This allows use of a burst-oriented [8], pipelined [6] or batched [3] approach which involves buffering information at interior network nodes in order to code them with other incoming information from the same batch. An alternative for cyclic networks is to take a continuous coding approach [9], [2] where information from different time periods is combined at intermediate nodes, and memory is needed at the receivers for decoding what is essentially a convolutional code. Both specifying and decoding the code are more complex than in the acyclic case. Another possibility proposed for cyclic networks is for a node that is part of a cycle to subtract off its previous contribution to data returning to it along the cycle [10]. This requires a node to know what it has contributed to data on its incoming links, which makes it more suited to centralized scenarios.

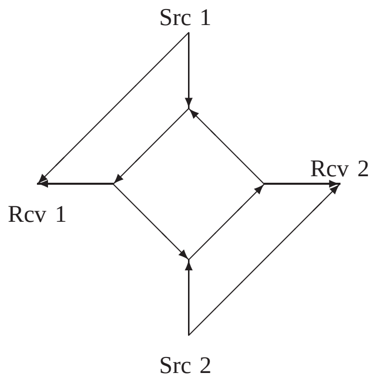

Fig. 1. An example of a multicast connection problem in which no acyclic subgraph is sufficient to accommodate the desired connections: all four links of the cycle in the middle of the network must be used.

The simpler burst/batch approach requires an *acyclic network code*, i.e. one which does not include a directed cycle of links each transmitting data that is a function of data on its predecessor in the cycle, possibly coded together with other inputs. For some network connection problems, such as those in Figure 1, there is no valid acyclic network code. However, such examples seem to be the exception rather than the norm; for instance, in the case where every link in the network has equal capacity in both directions, we do not know of an example in which coding over cycles is needed.

As such, we investigate decentralized techniques for obtaining acyclic network codes. Our approach restricts consideration to a subset of *link connections* (i.e. pairs of incident links such that the data on one link influences that on the other) of the graph that does not include cycles, and then finds a network coding solution involving a subset of the chosen link connections, if such a solution exists. We assume a network model in which reverse channels exist on each link, allowing downstream nodes to communicate control information to

Tracey Ho is with Bell Laboratories, Murray Hill, NJ 07974, e-mail: trace@mit.edu. This work was done while a postdoctoral scholar at the Coordinated Science Laboratory, University of Illinois.

Ben Leong is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, e-mail benleong@csail.mit.edu

Ralf Koetter is with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801, e-mail: koetter@csl.uiuc.edu

Muriel Médard is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, e-mail: medard@mit.edu

upstream nodes. The basic elements of our approach are:

Stage A

- an acyclic subset of link connections is found

Stage B

- upstream nodes communicate to downstream nodes what information is available along various paths using the chosen link connections
- downstream nodes request information accordingly from selected upstream nodes
- intermediate nodes transmit appropriate information in response to requests

Our approach for Stage A uses a Bloom filter [11], [12] to keep track efficiently of the links traversed by data forming part of the linear combination on each link. We compare, by simulation on random geometric graphs, the resulting multicast capacity with the multicast capacity of the original graph. Preventing cycles in network coding is more complicated than preventing cycles in routing because in network coding, the same data may need to be carried over multiple paths in different linear combinations. As an illustration of this, we compare our algorithm with a simpler algorithm that uses distance vectors to disallow cycles.

Stage B sets up an acyclic network code on the set of link connections found in Stage A. We present an approach that finds a valid network code, if one exists on the given link connections, with error probability decreasing exponentially in the length of the code. We show that the control overhead is proportional to the number of receivers and links.

These ideas characterize a class of distributed asynchronous algorithms that maintain a feasible multicast solution, while offering a compromise between coordination overhead and performance. The additional communication overhead compared to the completely distributed approach of [1] is reasonable, while the savings in link usage and amount of coding are potentially quite significant in some networks.

Our algorithm also allows for some flexibility in the choices of links and network code to be made according to some objective. However, achieving an optimal solution is complicated by the fact that in some cases, such as that of Figure 2, optimizing some objective for one receiver conflicts with optimizing it for another. It is interesting to note that while network coding allows different receivers in a multicast session to share links without affecting each others' rate, this does not extend to other objectives such as delay or network code complexity. In such scenarios, optimal allocation of network resources requires a global objective function combining different receivers' objectives. Trading off optimally between the objectives of two receivers at an intermediate node may depend on the availability of other paths to those receivers and the trade-offs along those paths. We do not address optimization issues fully in this paper, but note that our approach provides a potentially useful framework upon which various policies, e.g. based on pricing, may be imposed to achieve different network objectives.

II. Model

The linear network coding model we use is from [9]; we give a brief description here. The network is modeled as a directed graph of unit capacity links, and one or more unit rate sources are located at various nodes. This gives rise to an elegant and simple mathematical framework for network coding, without sacrificing generality, as links of larger capacities can be modeled as parallel links between the same nodes, and sources of higher rate can be modeled as multiple sources at the same node.

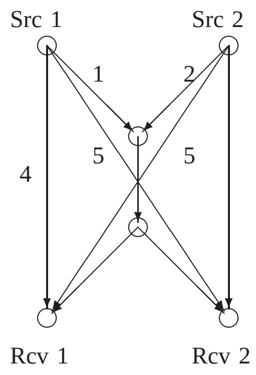

Fig. 2. An example of a network where optimizing delay for receiver conflicts with optimizing delay for another. The labels on each link represent the delay of that link; delay of unlabeled links is 1. To optimize delay, each receiver separately will choose to use the bottleneck link in the middle of the network. For the batch approach, optimizing delay for receiver 1 affects delay for receiver 2 since data from source 1 must be delayed at the bottleneck link to be combined with data from source 2. For the continuous approach, optimizing delay for receiver 2 affects delay for receiver 1 since data from source 2 is combined with newer data from source 1, which cannot be decoded at receiver 1 until the data from source 1 arrives separately.

We denote by (v, w) a directed link from *origin* node v to *destination* node w, and denote by $o(l)$ and $d(l)$ the origin and destination nodes respectively of a link l.

Information is transmitted as vectors of bits. Linear coding[1] is carried out on vectors of length u in the finite field $\mathbb{F}_{2^u}$. The process $Y(j)$ on a link j is a linear combination of *inputs* of node $v = \mathrm{o}(j)$, i.e. processes X_i generated at v and processes $Y(l)$ on incident incoming links l. This is represented by the equation

$$Y(j) = \sum_{\{i \,:\, X_i \text{ generated at } v\}} a_{i,j} X_i + \sum_{\{l \,:\, \mathrm{d}(l) = v\}} f_{l,j} Y(l)$$

where $a_{i,j} \in \mathbb{F}_{2^u}$, $f_{l,j} \in \mathbb{F}_{2^u}$. An illustration of linear coding at a network node is given in Figure 3.

In the distributed randomized network coding of [1], variables $a_{i,j}, f_{l,j}$ are chosen uniformly at random from $\mathbb{F}_{2^u}$. The receivers need only know the overall linear combination of source processes in each of their incoming signals. This information can be sent through the network as a vector, for each signal, of coefficients corresponding to each of the source processes, updated at each coding node by applying the same linear mappings to the coefficient vectors as to the information signals. A randomly chosen network code is successful if each

[1] which is sufficient for multicast [8]

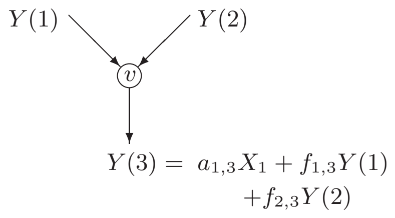

Fig. 3. Illustration of linear coding at a node.

receiver obtains as many linearly independent combinations as the number of source processes. This enables it to decode each source process.

III. Algorithm descriptions and analysis

We describe below a two-stage approach for setting up an acyclic multicast network code in a distributed, asynchronous fashion. The first stage algorithm, which sets up an acyclic set of connections on a given network, is not guaranteed to be optimal, and its performance evaluation is deferred to the next section on simulations. For the second stage, which finds a network code on the set of connections found by the first stage, we provide analytical bounds on overhead and probability of success.

Since the second stage uses the output of the first stage, they must initially be carried out sequentially. Subsequently, both stages or just the second can be run at intervals, to deal with network changes, or to see if a better network code can be found. To distinguish the control messages associated with different runs of the algorithms, the control messages are labeled with run numbers.

A. Choosing an acyclic set of connections

We consider the following cycle-preventing algorithm which sets a boolean variable $b(l, l')$ for each pair (l, l') of incident links, a one indicating that data from l can feed into l', and a zero indicating that the connection is disallowed. Control messages are initiated at the sources and transmitted on all their outgoing links. These messages are transmitted throughout the network and combined with other such messages at intermediate nodes. Each message m contains an epoch number which is initially zero, a vector of distances in hops from each source in m (i.e. sources of messages that have been combined to form m), and a list of all links traversed by messages that have been combined to form m. The list of links can be stored reasonably efficiently using a Bloom filter [11], [12].

All the boolean variables are initially set to zero. A node receiving a control message on a link l checks its list, and, for each incident outgoing link l', sets to one the variable $b(l, l')$ if l' is not in the list and $d(l)$ is at least as many hops away from some source in m than $o(l)$. At regular intervals, with a 50% jitter, each node combines and sends control messages received during the last interval whose boolean variables allow them to be sent on each outgoing link, by combining their distance vectors appropriately and taking the bitwise OR of their Bloom filters.

Whenever a node changes its coding coefficients, it increments the epoch numbers of its messages by some random amount. A node ignores received messages with lower epoch numbers. A node receiving a message with a higher epoch number sets the epoch numbers of its own messages to match this higher number. This mechanism helps to prevent local oscillations by ensuring that a node will accept an input from a neighboring node only after it has taken into account its local changes; we increment the epoch numbers by a random amount instead of a constant amount so that it is highly unlikely for two neighboring nodes to simultaneously increment the epoch number and not have at least one party realize that the other has made a change.

A simple approach we investigate for comparison uses only distance vectors. In this approach, a link is allowed to transmit a signal containing an input from source s only if its origin node is nearer (in terms of hop count) to the s than its destination node, or, if they have the same hop count, if the origin node has a smaller id.

B. Setting up a network code

1) Overview: As outlined in the introduction, there are three main components for this stage:

- upstream nodes communicating to downstream nodes the availability of information along various paths. This entails describing not only connectivity to various sources, but also the available capacities. Compared to describing network topology exactly, less overhead is incurred by providing only partial information in the form of coefficient vectors formed by random network coding, called *advertisement vectors*. Advertisement vectors that are linearly independent indicate the availability of link-disjoint paths to sources; those with linear dependencies are likely to share a bottleneck link.
- downstream nodes requesting information from upstream nodes. Each receiver initiates a set of requests which are propagated upstream along disjoint paths found using advertisement vectors. Information requested by one receiver may be useful to another; requests are propagated upstream only as far as needed to obtain useful information. A node seeking an input whose coefficient vector is linearly independent of a set $\mathcal{O}$ of vectors specifies this using a vector $\underline{v}$ in the nullspace of $\mathcal{O}$. A vector whose dot product with $\underline{v}$ is nonzero is independent of the vectors in $\mathcal{O}$,[2] and is termed *complementary* to $\underline{v}$.
- intermediate nodes transmitting appropriate information in response to requests. A node v transmits on each of its outgoing links l a random linear combination of a subset of its inputs. The subset is chosen so as to satisfy requests received on l.

Differences in the details of the algorithm give rise to variants that have different overheads, delay to obtaining a solution, and worst-case performance bounds. In our algorithm description below, we will distinguish in particular three variants 1, 2 and 3 at those points in which they differ. The

[2]This is similar to the approach used by [6] to test if a vector is independent of a given set of vectors.

best bounds on worst-case overhead and delay are obtained for variant 1; variants 2 and 3 proceed with decreasing degrees of caution and generally obtain solutions faster, but at the risk of requiring more work in the worst-case.

2) Detailed description: Each node stores, for each of its incoming incident links, an advertisement vector and a *current vector*, and possibly a number of *reduced advertisement vectors* associated with particular receivers. For simplicity of exposition, we consider each source process to arrive at its corresponding source node via a virtual link, whose advertisement vector is the unit vector with a single nonzero entry at the position corresponding to the source index.

The advertisement vector $\underline{a}(l)$ of each link l is is formed at its origin node as a random linear combination, specified by randomly chosen coefficients $f^{\mathcal{A}}_{i,l}$, of the advertisement vectors of its incoming incident links

$$\underline{u}(l,\beta) = \sum_{i:d(i)=o(l)} f^{\mathcal{A}}_{i,l}\underline{a}(i),$$

and is communicated to its destination node. Within one run of the algorithm, the advertisement vector of a link is updated if and only if the advertisement vector of one of its incident incoming links has changed. Since the algorithm is run on an acyclic set of link connections, the advertisement vectors will stabilize as long as the time scale of network topology changes is long compared to the time scale of advertisement vector updates.

A reduced advertisement vector $\underline{u}(l,\beta)$ is set in the course of the algorithm for (link, receiver) pairs (l,β) such that l is in the set $\mathcal{Q}(\beta)$ of links traversed by requests from β. Otherwise, for $l \notin \mathcal{Q}(\beta)$, $\underline{u}(l,\beta)$ is set or updated if the reduced advertisement vector $\underline{u}(l',\beta)$ of an incident incoming link l' is set or updated, according to:

$$\underline{u}(l,\beta) = \sum_{i:d(i)=o(l)} f^{\mathcal{A}}_{i,l}\underline{w}(i,\beta), \qquad (1)$$

where $\underline{w}(i,\beta) = \underline{u}(i,\beta)$ if $\underline{u}(i,\beta)$ has been set, or $\underline{w}(i,\beta) = \underline{a}(i)$ otherwise.

The algorithm builds a network coding solution by modifying the current vectors. The current vector of each link is initialized to the all-zero vector, and is updated in response to requests and whenever the current vector of any incoming incident link has changed. The current vectors set by the algorithm give the linear combinations of data to be sent on each link in the network coding solution.

We denote by $\mathcal{A}(\mathcal{L})$ and $\mathcal{C}(\mathcal{L})$ the set of advertisement vectors and current vectors respectively of links in a set $\mathcal{L}$.

Receivers wait for the advertisement vectors to stabilize before initiating requests. Each receiver β first chooses a set $\mathcal{T}_\beta$ of incident incoming links whose advertisement vectors form a full rank set.[3]

Associated with each link $l \in \mathcal{T}_\beta$ is a *frontier link* and a *frontier vector*, which are initialized to l and $\underline{a}(l)$ respectively, and updated in the course of the algorithm as described below. We denote by $\mathcal{F}(\mathcal{L})$ the set of frontier vectors corresponding to links in a set $\mathcal{L} \subset \mathcal{T}_\beta$.

Let $\mathcal{S}_\beta$ be a subset of $\mathcal{T}_\beta$ such that $\mathcal{C}(\mathcal{S}_\beta) \cup \mathcal{A}(\mathcal{T}_\beta\backslash\mathcal{S}_\beta)$ is a full rank set. Requests are made on links in $\mathcal{T}_\beta\backslash\mathcal{S}_\beta$.[4] β sends requests one at a time, waiting each time for a response or timeout before sending the next. The set $\mathcal{S}_\beta$ is updated with each answered request.

Each request consists of a forwarding path, a *link request vector* whose function is to select appropriate links on which to forward the request, and a *solution request vector* used in checking if the request has reached a link with useful information. For a request made on some link $l \in \mathcal{T}_\beta$,

- the link request vector is any vector in the nullspace of $\mathcal{F}(\mathcal{T}_\beta\backslash\{l\})$,
- the solution request vector is any vector in the nullspace of $\mathcal{C}(\mathcal{S}_\beta) \cup \mathcal{F}(\mathcal{T}_\beta\backslash(\{l\} \cup \mathcal{S}_\beta))$
- the forwarding path $P(l)$ is extended with each successive request on l, consisting of the path taken by the previous request on l together with the frontier link $f(l)$.[5]

A request is forwarded directly to the end node of its forwarding path without processing at intermediate nodes.

A node v receiving a request on a link l_1 checks its incoming links to find a *complementary* link l_2 for the request, i.e. such that $\underline{w}(l_2,\beta)$ is complementary to the link request vector. If the current vector of l_2 is complementary to the solution request vector, the request is considered *answered*. An *answer message* specifying the new frontier link l_2 and frontier vector $\underline{w}(l_2,\beta)$ are sent to β. v sets $\underline{u}(l_2,\beta)$ to the all zero vector, and calculates $\underline{u}(j,\beta)$ for each outgoing link $j \notin \mathcal{Q}(\beta)$ according to (1). If j is a frontier link associated with some other link in $\mathcal{T}_\beta$, the corresponding frontier vector is set to $\underline{u}(j,\beta)$ and communicated to β. Otherwise, $\underline{u}(j,\beta)$ is transmitted on j. Each node v' that receives a reduced advertisement vector $\underline{u}(l,\beta)$ on an incoming link l stores its value and similarly calculates and sends the reduced advertisement vector or frontier vector for each of its outgoing links $j \notin \mathcal{Q}(\beta)$.

For variant 1, in finding a complementary link l_2, v checks only incoming links that have not been previously checked in response to requests from the same receiver. If the current vector of l_2 is not complementary to the solution request vector, an *update message* specifying the new frontier link l_2 is sent to β rather than an answer message. The reduced advertisement vectors are updated as before.

In variants 2 and 3, if the current vector of l_2 is not complementary to the solution request vector, v checks whether l_2 is the only complementary link for the request. If so, it forwards the request upstream on l_2. If there is more than one complementary link for the request, an update message is sent to β and the reduced advertisement vectors are updated as in variant 1; in variant 3, an additional *temporary request* is sent upstream on l_2. This temporary request has the same link and solution request vectors as the original request, and is

[3]If there is more than one such set, the choice can be made randomly or according to various criteria, e.g. include the maximum number of incoming links with independent current vectors. A possible extension would be for advertisement vectors to be accompanied by other related information such as prices or delays, which can be used to influence the choice of inputs.

[4]We assume that this algorithm is carried out on a directed acyclic set of connections; note that requests are sent on each link in the opposite direction to that of data flow on the link.

[5]Path information is added to a request as it is forwarded beyond $f(l)$.

forwarded upstream by each successive node on an incoming link complementary to the request, until one is found whose current vector is complementary to the solution request vector.

In variants 2 and 3, a request may be forwarded by one or more links beyond the existing frontier link before it is answered with a new frontier link. Let P be the set consisting of these links together with the new frontier link. As the answer message is forwarded along the reverse of the path taken by the request, for each link $l \in P$, node $d(l)$ sets $\underline{u}(l, \beta)$ to the all zero vector and updates $\underline{u}(j, \beta)$ for each outgoing link $j \notin \mathcal{Q}(\beta)$ as before.

Each node v maintains, for each link (v, v') on which it receives requests, a set $\mathcal{I}((v, v'))$ of incident incoming links. Each answered request or temporary request that passes through (v, v') is associated with the complementary link chosen by v in response to the request; the union of these incoming links for all receivers forms $\mathcal{I}((v, v'))$. The current vector of link (v, v') is a random linear combination of the current vectors of links in $\mathcal{I}((v, v'))$.

When β gets an answer or update message on a link $l \in \mathcal{T}_\beta$, it waits for potential updates of other current and frontier vectors. If it gets an update message, it sends a new request on l. If it gets an answer message, it checks whether the current vector of l is linearly independent of $\mathcal{C}(\mathcal{S}_\beta)$. If so, it adds l to $\mathcal{S}_\beta$ and proceeds to send requests in similar fashion on links in $\mathcal{T}_\beta \backslash \mathcal{S}_\beta$. If, however, the current vector of l is linearly dependent on $\mathcal{C}(\mathcal{R})$ for some $\mathcal{R} \subset \mathcal{S}_\beta$, the receiver chooses a link $l' \in \mathcal{R}$ whose frontier vector has weight greater than one. l replaces l' in $\mathcal{S}_\beta$, and a new request is made on l'.

In a dynamically changing network, nodes may leave or fail in the middle of the algorithm, and a request formed using outdated advertisement or current vectors may time out or be answered with an error message. *Reset messages*, which reset $\underline{u}(l, \beta)$ to $\underline{a}(l)$ for each $l \in P(l)$, and reset $f(l)$ to l, may be used to clear algorithm state for links $l \in \mathcal{T}_\beta$ whose advertisement vectors have changed owing to network changes. The sets $\mathcal{T}_\beta$ and $\mathcal{S}_\beta$ may also have to be changed. Policies may also be set for renewal and clearing of requests, particularly if multicast group membership changes frequently.

3) Analysis:

Lemma 1: The algorithm maintains the invariant that the paths $P(l), l \in \mathcal{T}_\beta$, for each receiver β are part of a feasible flow solution from the sources to β, unless the network changes during the run of the algorithm.

Proof: We show by induction that the invariant above holds, and that the frontier vectors of β at any stage correspond to advertisement vectors on a modified network in which links in $P(l), l \in \mathcal{T}_\beta$, have been removed.

Note that a set of r links with independent advertisement vectors must be connected by r link-disjoint paths to the r sources. The r links in the set $\mathcal{T}_\beta$ form such a set, which gives the base case for induction.

Assume that the induction hypothesis holds up to some point in the algorithm, and that link l is the next complementary link chosen in response to a request from β. Since $\underline{w}(l, \beta)$ is complementary to the link request vector, it is independent of the other frontier vectors, and there exist disjoint paths from l and the other frontier links to the sources that do not go through links in $P(l), l \in \mathcal{T}_\beta$. The procedure for updating reduced advertisement vectors upon choosing l is equivalent to removing l from the network. This completes the induction. ■

Theorem 1: Given a feasible problem on a static acyclic connection graph, the algorithm finds a solution with probability at least $(1 - d/q)^{2\nu}$ with at most νd requests, where ν is the number of network links, d is the number of receivers and q is the size of the finite field in which coding is done. Variant 1 carries out at most $3\nu d$ tests for complementary vectors, while variant 2 carries out at most $(\Gamma + 2)\nu d$ such tests, where Γ is the maximum in-degree of a node.

Proof: The randomly chosen advertisement vectors form a full rank set at every receiver with probability at least $(1 - d/q)^\nu$, by the random coding result of [1]. Given a successful set of advertisement vectors, by Lemma 1, the algorithm then finds a set of disjoint paths $P(l), l \in \mathcal{T}_\beta$ from β to the sources, unless the receiver obtains a full rank set of inputs before this happens. By coding randomly over the union of the paths formed for each receiver, a valid solution is found with probability at least $(1 - d/q)^\nu$ [1]. The overall success probability is thus $(1 - d/q)^{2\nu}$.

In a static network, each link is tested for being complementary to a request at most once for each receiver, since a link that is not complementary to a request cannot be part of a valid flow solution through paths $P(l), l \in \mathcal{T}_\beta$, and thus will not be complementary to subsequent requests. Thus, the total number of such tests is at most νd for variant 1, and $\nu d\Gamma$ for variant 2.

For each link added to $\mathcal{Q}(\beta)$, at most two tests for complementarity of current vectors are carried out, one at the frontier link and one at β. The total number of such tests is at most $2\nu d$.

Each request finds at least one complementary link. Thus, each receiver sends at most ν requests.

■

IV. Simulations

We are still in the process of implementing the Link Request algorithm described in Section III-B in our event-driven simulator. We have however carried out some simulations of our Bloom-filter-based algorithm for choosing an acyclic set of connections (<u>Loop Free</u>), comparing it to a simple algorithm based solely on distance vectors (<u>Distance Vector</u>). Descriptions of these algorithms are given in Section III-A.

We generated several hundred random geometric networks of moderate size (15 to 50 nodes) with two to four randomly placed sources and up to eight randomly placed receivers. Any two nodes within a specified range are connected by a unit capacity link in each direction; the ranges are chosen so that connected networks are obtained. The average degree of the nodes in our networks range from 6 to 9.

We measured the success rate of the two algorithms, i.e. the probability that a particular sink obtains a full rank set of inputs with random linear coding, given that there exists a feasible code for the network (there may be no acyclic code for the network, but we do not have an efficient algorithm for determining this). We use a relatively small finite field size of 29. Our simulation results are shown in Figure 4.

Our results demonstrate that it is progressively more difficult to find a feasible acyclic set of connections as the number of sources are increased. Nevertheless, our Bloom-filter-based algorithm outperforms the distance-vector-based algorithm, with the performance gap increasing with the number of sources and network nodes.

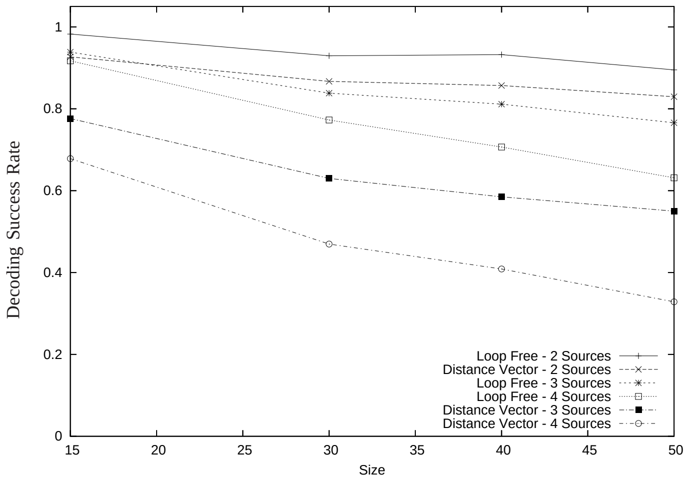

Fig. 4. Plot of success rate against network size (number of nodes) for random graphs.

V. Conclusion

We have presented distributed asynchronous algorithms for obtaining low-complexity acyclic network coding solutions for multicast on cyclic graphs. Our algorithms improve on existing distributed randomized network coding in that they can save on link usage and can operate on cyclic graphs without requiring convolutional decoding, bringing such approaches closer to practical use.

Acknowledgments

We would like to thank the anonymous reviewers for their comments which helped greatly to improve the presentation of the paper.

References

[1] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in Proceedings of 2003 IEEE International Symposium on Information Theory, June 2003.

[2] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing, October 2003.

[3] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in Proceedings of 41st Annual Allerton Conference on Communication, Control, and Computing, October 2003.

[4] D. Lun, M. Medard, T. Ho, and R. Koetter, "Network coding with a cost criterion," MIT LIDS TECHNICAL REPORT P-2584, 2004.

[5] Y. Wu P. A. Chou and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," submitted to the IEEE Transactions on Communications, 2004.

[6] 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 Transactions on Information Theory, Submitted July 2003.

[7] R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network information flow," IEEE Transactions on Information Theory, vol. 46, pp. 1204–1216, 2000.

[8] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," IEEE Transactions on Information Theory, vol. 49, pp. 371–381, 2003.

[9] R. Koetter and M. Médard, "An algebraic approach to network coding," IEEE/ACM Transactions on Networking, October 2003.

[10] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," submitted to the IEEE Transactions on Information Theory, 2004.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commununications of the ACM, vol. 13, no. 7, pp. 422–6, July 1970.

[12] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing, 2002.