Multicast with Network Coding in Application-Layer Overlay Networks

Ying Zhu, Baochun Li, *Member, IEEE*, and Jiang Guo

***Abstract*— All of the advantages of application-layer overlay networks arise from two fundamental properties: (1) The network nodes in an overlay network, as opposed to lower-layer network elements such as routers and switches, are end systems and have capabilities far beyond basic operations of storing and forwarding; and (2) The overlay topology, residing above a densely connected IP-layer wide-area network, can be constructed and manipulated to suit one's purposes.**

In this paper, we seek to significantly improve end-to-end throughput in application-layer multicast by taking full advantage of these unique characteristics. This objective is achieved with two novel insights. First, we depart from the conventional view that data can only be replicated and forwarded by overlay nodes. Rather, as end systems, these overlay nodes also have the full capability of encoding and decoding data at the message level using efficient linear codes. Second, we depart from traditional wisdom that the multicast topology from source to receivers needs to be a tree, and propose a novel and distributed algorithm to construct a 2-redundant multicast graph (a directed acyclic graph) as the multicast topology, on which network coding is applied. We design our algorithm such that the costs of link stress and stretch are explicitly considered as constraints and minimized. We extensively evaluate our algorithm by provable analytical and experimental results, which show that the introduction of 2-redundant multicast graph and network coding may indeed bring significant benefits, essentially doubling the end-to-end throughput in most cases.

***Index Terms*— Application-layer overlay networks, network coding, application-layer multicast.**

I. Introduction

Due to the lack of a widely available IP multicast service at the network layer in backbone networks, recent research (e.g., [1], [2], [3], [4]) has examined the feasibility and trade-offs of implementing multicast services in the application layer. The general approach common to all existing proposals is to have applications self organize into a logical overlay network, and to transfer data along the edges of such an overlay network using unicast transport services. Each application-layer node communicates only with its neighbors in the overlay network. Multicasting is implemented by forwarding messages along *overlay multicast trees* that are constructed and embedded in the virtual overlay network.

Application-layer multicast, in general, enjoys two attractive advantages over traditional IP multicast: (1) Multicast support in the network layer is not required; (2) Data is transmitted between nodes via unicast, effectively exploiting all existing security, flow control and reliable delivery mechanisms that are readily available and mature. However, an overlay multicast approach, however efficient, cannot perform as well as IP multicast. It is impossible to completely prevent multiple overlay edges from traversing the same physical link, causing unavoidable redundant traffic (identical copies of application-layer messages) on the same link, referred to as link *stress* [1]. Further, unicast communication between end systems involves traversing other end systems, potentially increasing latency. It is therefore critical to evaluate and seek to minimize both the relative increase of end-to-end latencies (caused by link *stretch*[1]) and the increase in per-link bandwidth requirements as compared with network-layer multicast.

Beyond what has been extensively studied in previous work, we emphasize that the advantages of deploying application-layer overlay networks arise from two fundamental properties. (1) Network nodes in an overlay network, as opposed to lower-layer network elements such as routers, are end systems and have capabilities far beyond basic operations of storing and forwarding. (2) The topology of an overlay network can be manipulated willfully to suit one's purposes since it resides on top of a densely connected IP-layer network. The links between nodes can be dynamically created or torn down to construct topologies that are conducive to better network performance. Recent research in application layer multicast (e.g., [1], [2]) has shown that it may well be worth the incurred cost of topology construction and maintenance to profit in increased robustness, flexibility and efficiency.

In this paper, we seek to improve end-to-end session throughput in an application-layer overlay multicast topology by taking full advantage of both of these unique characteristics. We deviate from the conventional view that data can only be replicated and forwarded by overlay routing nodes. Rather, as end systems, these overlay nodes also have the full capability of encoding and decoding data. We apply the mechanism of *network coding* [5], [6], [7] on intermediate overlay nodes. In addition, we also depart from the traditional wisdom that the *multicast topology needs to be a tree* from source to receivers; rather, we seek to construct a 2-redundant *multicast graph* (a directed acyclic graph to be defined in Sec. III) as the multicast topology, on which network coding is applied. Based on these insights, our main contribution is to propose a set of distributed algorithms to construct such multicast graphs and to subsequently assign linear codes and apply network coding, such that as a provable property, the end-to-end throughput may be significantly increased (doubled in

Ying Zhu, Baochun Li and Jiang Guo are with the Department of Electrical and Computer Engineering, University of Toronto. Their email addresses are {*yz, bli, jguo*}*@eecg.toronto.edu*.

[1]Formally, *stretch* is defined as the ratio of path length from the source to the multicast group member along the overlay to the length of the direct unicast path [3].

many cases) for all members of the multicast group. Since overlay multicasts come with the cost of link stress and stretch, we design our algorithm such that these costs are explicitly considered as constraints and optimized. Achieving the objective of increasing end-to-end multicast throughput is particularly important when applications such as content distribution services demand the highest capacity possible in overlay networks. We extensively evaluate our algorithm by using both analytical and simulation-based experimental tools. We show that our algorithm is indeed able to bring significant benefits with respect to increasing end-to-end session throughput.

The remainder of this paper is organized as follows. Sec. II motivates the case for application-layer coded multicast, by presenting concepts, advantages and requirements of network coding. Sec. III presents formal definitions of terms, leading to the main theorem with respect to the maximally achievable throughput with network coding. Our algorithm is formally presented in Sec. IV, along with its provable properties and relevant discussions. Sec. V presents experimental results using simulations. Finally, Sec. VI evaluates our proposal in the context of related work, and Sec. VII concludes the paper.

II. A Case for Application-layer Coded Multicast

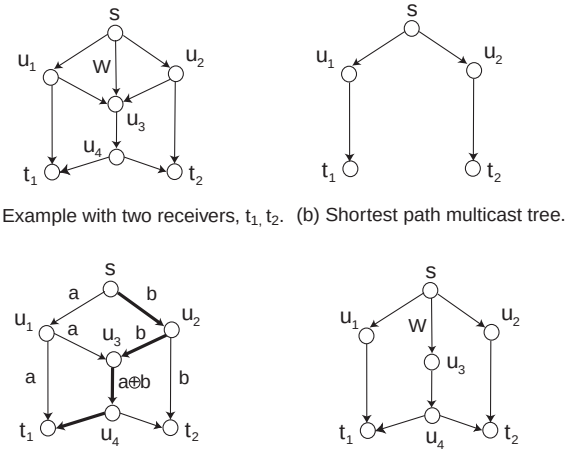
The main contributions of this paper are: (1) constructing multiple data paths from source to multicast group members, and (2) applying the concept of network coding in application-layer multicasts, motivating the case for *application-layer coded multicast*. The objective is to take advantage of alternate paths and excess capacity in the IP-layer network topology, and to significantly increase end-to-end multicast capacity.

The information-theoretic aspects of network coding was first proposed and studied by Ahlswede *et al.* [5]. With network coding, nodes have the capability of encoding and decoding data at the per-message level using efficient linear codes; the aim is to use bandwidth more efficiently and thereby increase network capacity.

We briefly review the concepts of network coding with an example shown in Fig. 1(c). The example shows how the session throughput of an 1-to-2 multicast session may be improved. In the figure, a and b represent two independent information flows originating from the source S. Node u_3 transmits the coded flow $a \oplus b$ along the "bottleneck" link $u_3 - u_4$ to node u_4, which then forwards the coded flow to both destinations t_1 and t_2. Receivers t_1 and t_2 can recover $\{a, b\}$ from $\{a, a \oplus b\}$ and $\{b, a \oplus b\}$, respectively. The session achieves a throughput of $2C$, assuming each link has capacity C. Without network coding, it can be verified that the achievable throughput is only $3C/2$.

However, network coding is not the panacea when it comes to increasing multicast session throughput. There exist many topologies — including all forms of multicast trees — where network coding fails to be more effective with respect to improving throughput. It helps to increase throughput only in network graphs that conform to special patterns. It is extremely difficult to manipulate nodes in the IP layer to construct multicast graphs that conform to specific patterns, and it is infeasible to modify all IP routers and switches to support coding. Overlay networks, on the other hand, have exactly the properties that could be leveraged to employ network coding for higher throughput in application-layer multicast: flexibility in topology construction, and capability of encoding and decoding.

Traditionally, the fundamental topological structure of multicast, whether it be IP-layer or application-layer, is a tree. Hence, every multicast group member in the tree has only one path from the source root; its throughput is limited by this path. To increase throughput by adding another path from the source to each receiver, one is faced with two problems: (a) The gain may be overshadowed by the *cost*[2] of the additional links and nodes in the alternative paths. (b) One must ensure that the alternative paths do not conflict with the original paths in order to avoid throughput-limiting bottlenecks.



(a) Example with two receivers, t_1, t_2. (b) Shortest path multicast tree.

(c) Throughput is doubled with network coding and alternative paths (dark edges form second path for t_1, second path for t_2 is the mirror image).

(d) Throughput is doubled using alternative paths, but not using network coding.

Fig. 1. The effects of network coding: an example.

We propose a new application-layer multicast strategy that, by appropriate use of network coding, will resolve these problems and achieve the higher throughput without commensurate cost or complexity. We use the previous example to illustrate how we apply network coding advantageously. The overlay network is represented by the graph in Fig. 1(a), in which t_1 and t_2 are the two receivers in the multicast group, and s is the source. Each edge has the same bandwidth of 1 except that the bandwidth available on edge (s, u_3) is $w \ll 1$. This is the case when, for example, s can not sustain an outgoing bandwidth of much more than 2. The usual all-widest-paths multicast tree is shown in Fig. 1(b); the widest alternative paths are added in Fig. 1(c), while the other choice of (narrower) alternative paths are shown in Fig. 1(d).

Without network coding, it is impossible to double throughput in Fig. 1(c), since the alternative paths to t_1 and t_2 interfere with each other's widest paths such that they cannot

[2]The *cost* of a link embodies critical metrics of concern, such as bandwidth, latency and loss rate.

both double their throughput. The conflict can be eliminated by choosing the paths in Fig. 1(d), but the bandwidth of the alternative paths is much less (narrower) than that of the original tree, again making it not feasible to double the throughput. This example exemplifies the two concerns specified previously. With network coding, however, the graph in Fig. 1(a) can be safely used in the multicast to double the throughput to both receivers, as shown in Fig. 1(c). The tremendous power of network coding lies in the fact that any conflicts resulting from interfering paths in the multicast graph can be resolved to obtain the same throughput for each receiver as if it was the only receiver.

III. Preliminaries

We consider a network graph with a source node s and a set of multicast group members as *receiver* nodes (or simply *receivers*). We define two notions of maximum flow and k-redundant multicast graph.

Definition 1 (individual maxflow). Given any single receiver t, we say that the *individual maximum flow* (or simply "individual maxflow") of t is the maximum flow from s to t when data flows from s *only* to t, and the other receivers are not considered except as part of the subgraph serving the data flow from s to t.

Definition 2 (simultaneous maxflow). When all multicast group members receive data flowing from s simultaneously, *i.e.*, a multicast, the maximum flow that a receiver t achieves is the *simultaneous maximum flow* of t.

Definition 3 (k-redundant multicast graph). A *k-redundant multicast graph* for single-source multicast is a directed acyclic graph (DAG) which has the following two properties:

1) The set of all nodes, A, is the union of three disjoint subsets $\{s\} \cup A_I \cup A_T$: $\{s\}$, the source, indegree$(s) = 0$ and outdegree$(s) > 0$; A_I, the *intermediate* nodes (who are not members of the multicast group), denoted by $u_i, 1 \leq i \leq n_I$, $1 \leq$ indegree$(u_i) \leq k$ and outdegree$(u_i) > 0$; A_T, the *receiver* nodes (*i.e.*, multicast group members), denoted by $t_i, 1 \leq i \leq n_T$, indegree$(t_i) = k$ and outdegree$(t_i) \geq 0$;
2) If each edge in the graph has unit bandwidth, then for any node whose indegree is k, its individual maxflow is k.

In particular, since the indegree of all receivers is k, the individual maxflow of each receiver t is k, if the graph has unit-bandwidth edges. If edges have different bandwidths, then the individual maxflow of t is clearly k times the bandwidth of the bottleneck edge between s and t. Without loss of generality, we can assume that all the edges have the *same* bandwidth (that of the bottleneck). This assumption will be made throughout this section to simplify presentation of proofs.

Let $n = 1 + n_I + n_T$ be the total number of nodes. In this paper, we only consider the case of 2-redundant multicast graphs. The justifications are two-fold: (1) As k increases, both the sustainable physical link stress leading to overlay nodes and the limited number of intermediate nodes and receivers significantly decrease the probability of finding multiple good paths from the source to each receiver. In the example of Fig. 1(a), when the sustainable stress on all nodes is no greater than 3 incoming and outgoing flows, it is infeasible to construct a k-redundant multicast graph if $k > 2$. (2) As k increases, the code assignment algorithm (Sec. IV-F) becomes more complex and averse to the dynamics of node joins and departures.

We now establish the sufficiency of a maximum indegree of 2 in a 2-redundant multicast graph. It is not immediately clear why no node in the graph needs more than two incoming edges to ensure individual maxflow of 2 to each receiver. We prove that a maximum indegree of 2 is sufficient by first proving the following observations (Proposition 1 and 2) about disjoint paths.

Definition 4 (disjoint paths). We say two paths from s to t are *disjoint* if they do not share any *common edges*.

Proposition 1: Given a node t with indegree 2 in a 2-redundant multicast graph with source s and unit-bandwidth edges, t has two disjoint paths from s if and only if t has individual maxflow of 2.

Proof: ($\Rightarrow$) This direction is straightforward. It is obvious that if there are two disjoint paths to t, then it has maxflow of 2. ($\Leftarrow$) A maxflow of 2 implies (in this case where edges have unit bandwidth) that there are two flows, f_1, f_2, each of bandwidth 1. Each flow clearly must define a path from s to t, let p_1, p_2 denote the paths for f_1, f_2, respectively. If p_1 and p_2 share a common edge e, then f_1 and f_2 must share the bandwidth of e, which is only 1. So the value of each flow is $1/2$, this is a contradiction. Therefore, p_1 and p_2 do not share any common edges and are two disjoint paths to t from s. □

Proposition 2: It is not necessary for any node in a 2-redundant multicast graph to have indegree of greater than 2 to obtain two disjoint paths for each receiver from s.

Proof: We only need to show that by constructing two disjoint data paths for a receiver t, it is not necessary for adding a third incoming edge to any node in the existing multicast graph. When constructing the first path p_1, suppose, by contradiction, that a third incoming edge is to be added to a node u. This clearly is not necessary, since a path must exist from u to t, $u \rightarrow t$, and a path must already exist from s to u (due to connectedness), $s \rightarrow u$, and so p_1 can be simply a concatenation of $s \rightarrow u$ and that from $u \rightarrow t$. This contradicts the necessity of adding the third incoming edge to u.

When constructing the second path p_2, suppose, again by contradiction, that a third incoming edge is to be added to a node u. Since u has an indegree of 2, by property 2 in Definition 3 and Proposition 1, u must have two disjoint paths from s, denote them by su_1, su_2, respectively. There are two cases. *Case 1:* First path for t, p_1, is disjoint from at least one of su_1 and su_2. Without loss of generality, suppose p_1 is disjoint from su_1, then p_2 can be constructed by concatenating su_2 and $u \rightarrow t$. It is clear that p_2 thus formed is disjoint from p_1. *Case 2:* Both su_1 and su_2 share common edges with p_1. Without loss of generality, suppose the common edge closest to t is shared by su_1 and p_1. Let $v \rightarrow w$ denote this common

edge, and also let $s \rightarrow v$ denote the segment of su_1 until v and $w \rightarrow t$ denote the segment of p_1 from w to t. A new first path for t, p_1', can then be constructed from concatenating $s \rightarrow v$ and $w \rightarrow t$; and p_2 is formed by su_2 followed by $u \rightarrow t$. It is easy to see that p_1' and p_2 are disjoint. □

The property of 2-redundant multicast graphs that the maximum indegree of a node is 2 not only reduces complexity in the algorithm to construct the graph, but moreover contributes significantly towards minimizing stress.

Towards the objective of constructing a 2-redundant multicast graph, Proposition 1 shows the importance of constructing two disjoint paths from the source s to a receiver t. Further, directed acyclicity must be preserved in the construction of the multicast graph, *i.e.*, there must be no directed cycles in the graph. Directed cycles in the network introduce great complexity and difficulty in determining the linear codes used for multicast [5], [6]. In fact, Koetter *et al.* [6] have not presented any solution for the case of general networks with cycles. We next show that, in order to construct a 2-redundant acyclic multicast graph, the set of intermediate nodes, A_I, needs to be non-empty.

Proposition 3: A 2-redundant multicast graph with only receivers, *i.e.*, every node beside s has two disjoint directed paths from s, contains a directed cycle.

Proof: Let t_1 be any node that has a directed edge from s. Since it has two disjoint paths from s, there is a second directed edge pointing to it from another node, say, t_2, denoted by $t_2 \rightarrow t_1$. Similarly, t_2 has at least one incident directed edge from a node that is not s, say, t_3. A chain of nodes may be built with $t_2 \rightarrow t_1$, $t_3 \rightarrow t_2$, and so forth. Since there exists a finite number of nodes, eventually t_i becomes the same node as t_j with $j < i$, *i.e.*, t_j is in the chain before t_i, thus forming a directed cycle. □

Obviously, there may not exist two disjoint paths from s to each u_i, which leads to the conclusion that multicast group members may not serve as intermediate nodes. We need to recruit dedicated high-degree relay nodes in the overlay network as intermediate nodes, who do not belong to the multicast group. For this purpose, we may deploy a pool of end hosts or proxy servers connected to high-bandwidth physical links. Naturally, the number of required intermediate nodes must be minimized, and scalable to large-scale multicast groups. Such a pool of dedicated nodes is the price we pay to exploit the power of network coding to significantly increase throughput. These intermediate nodes do not place much additional stress on the network, since they do not require multiple data delivery paths from the source. Facilitated by intermediate nodes, we seek to construct 2-redundant multicast graphs with no directed cycles (a DAG).

As an example, the multicast graph in Fig. 1(c) is 2-redundant, with receivers t_1 and t_2 both having two disjoint paths from s (as explained in Sec. II).

We now proceed to illustrate the essence of linear codes.

Definition 5 (linear coding multicast). *Linear coding multicast* views a block of data flowing over an edge as a vector and assigns a linear transformation for each node u in the multicast graph such that:

- for each outgoing edge e of u, the vector sent out on e is a linear combination of the vectors of the incoming edges;
- for source s, any vectors can be sent out on its outgoing edges;
- all the vectors are in the same infinite-dimensional vector space over a base field.

Linear codes are the coefficients that determine the linear transformations. For example, in Fig. 1(c), the data sent by u_3 on the outgoing edge is a linear combination of $\{a, b\}$: $1 \cdot a + 1 \cdot b = a + b$, where $+$ is defined in a finite field, e.g., GF(256). Such a linear combination is represented as $(1, 1)$.

The main result of network coding, first proposed in [7] (a slightly weaker version was proposed earlier in [5]), is stated in the following theorem. It essentially states that in an acyclic network with a source and multiple receivers, the maximum individual throughput of each receiver can always be achieved as if there was no interference at all from data flowing in the network to the other receivers, by using only linear coding.

Theorem 1 (Li and Koetter). For every multicast graph, there exists a set of linear codes that could be used for multicast (linear coding multicast) such that simultaneous maxflow of t_i is equal to individual maxflow of t_i.

Proof: The interested reader is referred to [6] or [7] for detailed proofs using different methodologies. □

IV. Algorithm and Analysis

The ultimate goal of our algorithm is to build and maintain a 2-redundant multicast graph as defined in Sec. III. There are several non-trivial challenges. Our algorithm addresses each of these challenges, and much of the complexity lies in tackling all of them in conjunction. (1) In order to subsequently apply network coding, we need to correctly construct a 2-redundant acyclic multicast graph from the source to all members of the multicast group[3]. During the construction process, data delivery paths should be optimized in the multicast graph to the receivers. Each receiver essentially has two paths from the source; both paths should be carefully chosen to maximize the aggregated throughput to the receiver. (2) We need to minimize the number of intermediate nodes with a given number of receivers while preserving good performance. (3) Minimizing stress is paramount since it directly determines how much actual bandwidth a virtual link has and high stress can severely diminish end-to-end throughput. These problems embody the fundamental objective of maximizing multicast performance (end-to-end throughput and latency) while minimizing the penalty incurred by elevating the functionality of multicast from the IP layer to the application layer and by using a multicast graph instead of a multicast tree. In particular, minimizing stretch is an integral part of optimizing the paths and minimizing stress is covered by imposing a maximum node degree in the multicast graph.

The algorithms we developed for our multicast scheme are fully distributed. Our scheme mainly consists of three steps. The first step is building a relatively densely connected graph of the set of *all* nodes in the group, A, referred to

[3]Henceforth, the terms multicast group members and *receivers* will be used interchangeably.

as the *rudimentary graph*. Fig. 2(a) shows a simple example of a rudimentary graph of a small multicast group with 7 nodes; the intermediate nodes, A_I, consist of $\{u_1, u_2, u_3\}$. The second and third steps are carried out for data delivery. In the second step, a spanning tree of *only* the intermediate nodes with source s as the root is constructed, referred to as *rudimentary tree*. With node r_1 being the source s, the rudimentary tree is shown, as darkened edges with incident nodes, in Fig. 2(a). Using the rudimentary graph and tree, the third step constructs the 2-redundant multicast graph by carefully selecting two paths through intermediate nodes from the source for each leaf receiver. In our simple example, the leaf receivers $A_T = \{r_2, r_3, r_4\}$ each has two disjoint paths from s, as shown in Fig. 2(b).

Both the rudimentary tree and the multicast graph have the degree constraint: every node has degree $\leq \Delta$; *i.e.*, Δ is the maximum node degree.

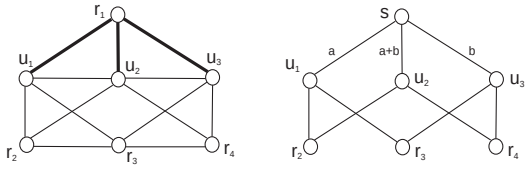


Fig. 2. A simple example of a multicast group, with rudimentary graph, rudimentary tree, and the resulting 2-redundant multicast graph.

A. Rudimentary graph

When a node joins the group, it is given a set of nodes already in the group, these are its initial neighbors in the rudimentary graph. The new node contacts its neighbors so they are made aware of it. Every node maintains a set of neighbors with which it periodically exchanges group information. That is, each node stores a list of the addresses of all the nodes it knows about in the group and periodically exchanges it with its neighbors' lists, and updates accordingly. After a node joins, the information will eventually propagate through the rudimentary graph.

Each edge (or link), e, in the rudimentary graph has associated with it a 2-tuple *weight*, $w(e) = (\beta, \lambda)$, where β is the link bandwidth and λ is the link latency. Each path, $p = (e_1, e_2, \ldots, e_m)$, also has an associated weight, $w(p) = (\beta, \lambda)$, and $\beta = \min(\beta_i, i = 1, \ldots, m), \lambda = \sum_{i=1}^{m} \lambda_i$, where $w(e_i) = (\beta_i, \lambda_i)$. As usual, path bandwidth is the minimum of the link bandwidths and path latency is the sum of the link latencies. Let $w(p_1) = (\beta_1, \lambda_1), w(p_2) = (\beta_2, \lambda_2)$ be the weights of p_1, p_2, respectively. p_1 is *better* than p_2 if $\beta_1 > \beta_2$, **or** $\beta_1 = \beta_2$ and $\lambda_1 < \lambda_2$.

Also, periodically, each node u chooses randomly another node v in the group that is not a neighbor and by sending a probing message, estimates the bandwidth and latency of the direct overlay link, (u, v). If the direct link is better than most of its (direct) links to its current neighbors, then v is added as a neighbor of u and the edge (u, v) is added to the rudimentary graph. The goal is to have overlay links (edges) that have good performance in the rudimentary graph. Let x be a current neighbor of u. From the same motivation, if (u, x) is much worse than the links u has to its other neighbors, and both u and x use this link rarely (*i.e.*, use it to reach very few nodes), then u drops x as its neighbor and (u, x) is removed from the rudimentary graph.

The dynamics of adding high-quality edges and dropping poor-quality edges is vital to the performance of the entire multicast scheme. Because, ultimately, the edges in the rudimentary graph are used to construct the data delivery paths, whose performance depends directly on the property (*i.e.*, weight) of these edges.

The graph resembles the Narada mesh [1], with the following important differences: (1) For every *intermediate* node, the number of its neighbors that are *intermediate* nodes must be no larger than Δ. This is necessary to ensure that the maximum degree of nodes in the rudimentary tree (of intermediate nodes) constructed from this graph is limited by Δ. (2) For *every* node (intermediate or not), the *total* number of its neighbors may be larger than Δ. The algorithm for building the multicast graph later explicitly enforces the Δ degree constraint, so nodes in the rudimentary graph can have more than Δ neighbors. Since these extra links are for control messages and not for data transmission, more of them can be allowed without raising concerns about performance degradation. (3) The subgraph of intermediate nodes with their incident edges must be connected.

Definition 6 (core graph). The *core* graph is the subgraph of the rudimentary graph with the set of vertices A_I and all the incident edges.

The core graph is kept connected by the same heuristics used for keeping the entire graph connected.

B. Rudimentary tree

The rudimentary tree is built from the subgraph consisting of the core graph and s (with its $\leq \Delta$ edges incident with the core). We adopt the distributed algorithm proposed by Wang *et al.* [8] based on distance vectors that finds the *shortest widest* paths. The widest path, or the path with the highest end-to-end bandwidth, is selected; and if there is more than one widest path, the shortest, one with the lowest end-to-end latency, is selected.

C. Multicast graph

The basic idea of building the multicast graph is to use the rudimentary tree as a basis and add edges, when necessary, from the rudimentary graph. Construction of the 2-redundant multicast graph G_{2r} adheres to the following rules for source node s and every intermediate node u:

1) s has k intermediate nodes as children and $2 \leq k \leq \Delta - 1$ (outdegree(s) $= k$);
2) total degree of u = indegree(u) + outdegree(u) $\leq \Delta$;
3) $1 \leq$ indegree(u) ≤ 2;
4) number of children of u that are leaf receivers $\leq \Delta - 1 -$ indegree(u).

The fourth rule forces an intermediate node to leave at least one outdegree available for adding an edge to another intermediate node. This ensures that the search for a path is *always* successful as long as there are enough intermediate nodes in the rudimentary graph, without resorting to expensive exhaustive search.

We shall use a running example throughout the description of the algorithm to help illustrate how the algorithm works. The core graph of the rudimentary graph and the rudimentary tree of the example are shown in Fig. 3. Note that the maximum degree, Δ, is 4 in our example.

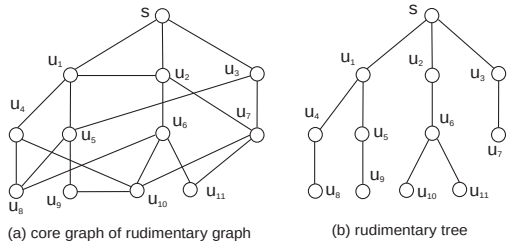


Fig. 3. The core graph of rudimentary graph and the rudimentary tree for an example set of intermediate nodes, with $\Delta = 4$.

We now need to define two additional terms.

Definition 7 (leaf intermediate). An intermediate node v is called a *leaf intermediate* if v does not have any downstream intermediate nodes in G_{2r}, *i.e.*, none of v's children in G_{2r} is an intermediate node.

Definition 8 (saturation). We say v is *saturated* if either degree(v) $= \Delta$, or v is a leaf intermediate and degree(v) $= \Delta - 1$.

For instance, in the G_{2r} in Fig. 4(b), u_7 is a leaf intermediate node while u_3 is not. If we assume $\Delta = 4$ for the graph in Fig. 4(b), then u_1 will be saturated if one leaf receiver is connected to it. While u_7 will be saturated if two leaf receivers are connected to it. Even when a leaf intermediate node is saturated, there is an outdegree preserved from leaf receiver to allow possibility of adding at least one child intermediate node.

The 2-redundant multicast graph G_{2r} is initialized to the rudimentary tree, so initially it has no leaf receivers. Adding each leaf receiver t entails constructing two disjoint data delivery paths for t. Edges not in the tree (but in the rudimentary graph) may be added to G_{2r} in the process. We now describe procedures for obtaining two disjoint paths, p_f and p_s, for a leaf receiver t. The distributed algorithms are formulated in Table I and Table II.

We seek to create, for t, two data delivery paths from s, denoted by p_f and p_s, denoting the first path and the second path, respectively. Moreover, p_f and p_s do not share any edges in common.

To find p_f, t first contacts all its neighbors in the rudimentary graph that are intermediate nodes and finds out which are unsaturated. If t has unsaturated neighbors, then t compares their tree paths appended with the edge from them to t, and selects node u with the best path. Let u_f denote the node that is parent of t in p_f, then u_f is assigned u. If all of t's neighbors are saturated, t initiates a breadth-first search of the tree to find the first k unsaturated nodes. Comparison of the k tree paths appended by respective edges from these nodes to t yields node u with the best path. As above, u_f is set to u. In either case, $p_f = P(s, u_f) \cup (u_f, t)$, where $P(s, u_f)$ is the tree path from s to u_f.

The primitive of the breadth-first search of the tree used in the procedure will recur in later procedures. The details of its implementation are presented in Sec. IV-D. The number k represents a trade-off between efficiency of the algorithm and optimality of the path constructed. It should not be too high to avoid near-exhaustive search of the tree for an unsaturated node, when there are many leaf receivers saturating many nodes.

Suppose we want to construct p_f, p_s for t with the rudimentary graph and tree in Fig. 3, and the existing 2-redundant multicast graph is as shown in Fig. 4(a). Nodes u_1, u_2, u_3 are saturated.

TABLE I

PROCEDURE FOR CONSTRUCTING p_f FOR t

Leaf receiver t:
  if not all t's intermediate node neighbors are saturated (u_i's)
    u = Find_best_path($t, \{u_i\}$)
    $u_f = u$ and $p_f = p \cup (u, t)$
  else if all neighbors of t are saturated
    Breadth-first search of tree,
      halt when k unsaturated nodes $\{u_i\}$ found
    u = Find_best_path($t, \{u_i\}$)
    $u_f = u$ and $p_f = p \cup (u, t)$

Find_best_path($t, \{u_1, \ldots, u_m\}$):
  request u_i's for weights $\{w(p_i) = (\beta_i, \lambda_i)\}$ of their paths in tree
  for each (unsaturated) u_i
    $\beta =$ bandwidth of edge (u_i, t)
    compute $w(p_i \cup (u_i, t)) = (\min(\beta_i, \beta), \alpha + \alpha_i)$
  choose the best (shortest widest) path, p
  return u that corresponds to p

TABLE II

PROCEDURE FOR CONSTRUCTING p_s FOR t

Leaf receiver t:
  find k unsaturated intermediate nodes $\{u_i\}$, which are not in p_f, from its neighbors and/or random probing.
  send p_f to each u_i and request best path p_i from s to u_i that does not intersect with p_f and its weight $w(p_i)$
  $u =$ Find_best_path($t, \{u_i\}, \{p_i\}, \{w(p_i)\}$)

Find_best_path($t, \{u_i\}, \{p_i\}, \{w(p_i)\}$):
  for each u_i
    $\beta =$ bandwidth of edge (u_i, t)
    compute $w(p_i \cup (u_i, t)) = (\min(\beta_i, \beta), \alpha + \alpha_i)$
  choose the best (shortest widest) path, p
  return u that corresponds to p

Intermediate node u_i:
  Upon receiving p_f and request for paths from s to u_i disjoint from p_f
  if u_i's tree path $P(s, u_i)$ does not intersect with p_f then
    return $p = P(s, u_i)$ and $w(p)$
  else if u_i's tree path intersects p_f then
    if u_i has an alternative path p from s then
      return p and $w(p)$
    else if u_i has indegree 1 then
      contact a different child c of s than the one whose subtree u_i is in.
      c conducts breadth-first search of its subtree and returns to u_i first unsaturated or leaf intermediate node v.
      return $p = P(s, v) \cup (v, u)$ and $w(p)$

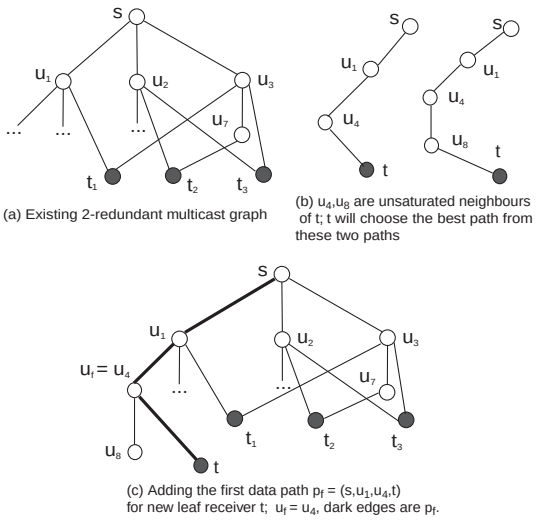


Fig. 4. Examples of 2-redundant multicast graphs. Ellipsis indicates part of graph there is not shown.

In our example, the unsaturated nodes t considers are u_4, u_8. t chooses the best path from the directed paths (s, u_1, u_4, t) and (s, u_1, u_4, u_8, t), as shown in Fig. 4(b). If the better path is the first one, then p_f for t is shown in Fig. 4(c). To keep the figures clean, the directions on the edges in the multicast graphs in later figures are omitted. The directions of the edges are uniformly downward.

For reference, the procedure for determining p_s is presented in Table II. The leaf receiver t first finds k unsaturated nodes $\{u_i\}$, which are not in p_f, from its neighbors. If there are fewer than k such neighbors, t randomly probes intermediate nodes. Now each u_i is requested by t to give the best path, p_i, from s to u_i. After t receives p_i's and $w(p_i)$'s from all the u_i's, t simply selects the best path among $\{p_i \cup (u_i, t)\}$. Let u_s denote the parent intermediate node of the best path p that t selects. The second data delivery path to t is $p_s = p \cup (u_s, t)$.

Each u_i, when requested by t, first checks if its tree path $P(s, u_i)$ intersects with p_f. If not, then $P(s, u_i)$ is returned. If it does intersect, then u_i finds an alternative path from s. An alternative path may already exist (if u_i has indegree 2), in which case, u_i replies with that. Otherwise, u_i sends a message to a child node c of s that is different from the child s who is upstream from u_i. It is a request for c to do a breadth-first search of its own subtree and reply to u_i the first unsaturated or leaf intermediate node found. Let v denote this node. u_i replies to t with $P(s, v) \cup (v, u)$.

Note that the search will *always* be successful, since a breadth-first search will always eventually find a leaf intermediate node. This is not true if the search was only for unsaturated nodes. Moreover, the algorithm requires for each leaf intermediate node to leave one outdegree free for an edge to another intermediate node (recall the definition of node saturation). This is the reason that we allow adding a second incoming edge to u_i. The rationale behind the allowance is quite intuitive: we are trying to find two best paths from s to t which must be disjoint, it is entirely possible that they do not *both* belong to the rudimentary tree.

Returning to our example, we let $k = 2$. For constructing p_s, t first finds two unsaturated nodes that are not in p_f: u_7 and u_8. t then sends requests to u_7, u_8 for their best paths from s. The tree path of u_7 from s does not intersect with p_f, so u_7 will return its tree path $p = (s, u_3, u_7)$. If t chooses $p \cup (u_7, t)$ as the best path, then p_s is shown in Fig. 5(a). Since u_8 is in the same subtree as u_4, it will need to find an alternative path from s, $p = (s, u_2, u_6, u_8)$. If $p \cup (u_8, t)$ is a better path than $\{(s, u_3, u_7)\} \cup (u_7, t)$, then p_s is shown in Fig. 5(b).

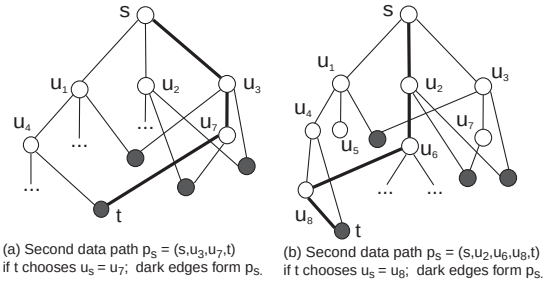


Fig. 5. Two examples of adding a second data path, p_s; darkened lines represent p_s.

D. Breadth-first search primitive

Breadth-first search of a tree to find unsaturated nodes can be made much more efficient than blind and exhaustive search. Keeping record of the states of saturation of the subtrees of children is not hard. When a node first becomes saturated, it simply sends that information upstream in the multicast graph. Recursively, a node, which is the root of subtree R, knows R is saturated when all its children have sent saturation notification to it. In this case, the breadth-first search takes guidance from the indicators of subtree saturation at the roots of subtrees. If the subtree is saturated, then no node in the subtree is searched henceforth.

E. Analysis

We prove the correctness of the algorithm, show some bounds on the number of intermediate nodes required, and discuss scalability.

Theorem 2. The graph constructed by the algorithm is indeed a 2-redundant multicast graph.

Proof: The graph is 2-redundant by construction, *i.e.*, it is constructed to be such that each leaf receiver has two disjoint paths from the source. We only need to show that the graph constructed is acyclic. We show by induction that this holds. The initialization of the graph is the rudimentary tree with directed edges from parent to child, which is certainly acyclic. Suppose an existing multicast graph is acyclic. We now prove that after adding a new leaf receiver t, the multicast graph remains acyclic. Let p_f, p_s be the two paths chosen for t. If p_f, p_s were already in the multicast graph, then only two directed edges to t are added, and the resulting graph is clearly still acyclic.

Suppose new edges are added to the graph for p_f and p_s. For p_f, only an edge directed to t is added, so no cycle is introduced.

For p_s, there are two cases. The first case is when the parent node of t in p_s, u_s, is not in the same subtree as the nodes in p_f. p_s is the path $P(s, u_s)$ in the rudimentary tree plus the added edge from u_s to t, which is the same scenario as for p_f above, hence no cycle exists.

The second case is when u_s is in the same child-subtree of s as nodes in p_f. So a directed edge (w, u_s) not in the rudimentary tree is added (in addition to (u_s, t)), where w is in a different subtree. We prove by contradiction that the directed edge (w, u_s) cannot be part of a cycle. Suppose that a cycle containing (w, u_s) indeed exists. Since w and u_s are in different child-subtrees of s, there must be a directed edge (u, v) in the cycle such that u is in the same child-subtree of s as u_s, while v is in a different child-subtree. Moreover, u is a descendant of u_s. The edge (u, v) was added before and when it was added, u_s was unsaturated. But (u, v) exists only if a breadth-first search from the top yielded that u_s was saturated, otherwise, (u_s, v) would have been chosen instead. This is a contradiction. □

With the restriction of maximum degree, Δ, on the nodes in the multicast graph, the number of intermediate nodes needed increases with the number of leaf receivers. For n_T leaf receivers, we would like to determine the maximum and minimum number of intermediate nodes possible. It may be easier to consider the contrapositives of these: (1) the maximum number of leaf receivers possible, in a 2-redundant multicast graph, $n_{T\max}$, for a given number of intermediate nodes, and (2) the minimum number of leaf receivers that can saturate a 2-redundant multicast graph using the above algorithm, $n_{T\min}$.

Theorem 3. Given n_I intermediate nodes,

$$n_{T\max} = \lfloor((\Delta - 2)n_I + 1)/2\rfloor, \quad (1)$$

$$n_{T\min} = \lfloor((\Delta - 4)n_I + 1)/2\rfloor. \quad (2)$$

Proof: It is easy to see that to prove Equation (1) is equivalent to finding the number of leaves in a $(\Delta-1)$-regular tree. (An m-regular tree is a tree in which every non-leaf (*i.e.*, intermediate) node has exactly m children.) The total number of children in tree $= (\Delta - 1)n_I =$ number of non-leaf nodes $+$ number of leaf nodes $+$ root $= n_I + n_T + 1$, and the claim follows directly.

Now we prove Equation (2). As above, consider the $(\Delta-1)$-regular tree. Starting with this tree, the number of leaf nodes decreases every time the algorithm adds a second incoming edge to a non-leaf (intermediate) node. There are two incident nodes of this additional edge and the degree of each decreases by 1, which means that the number of children of each decreases by 1. So each such second incoming edge added decreases the total number of children by 2. The worst case is when every non-leaf node has a second incoming edge. Therefore, the least number of total children is $(\Delta - 1)n_I - 2n_I = n_I + n_T + 1$, and the claim follows. □

Now we discuss the issue of scalability, by first noting that each node only exchanges control messages with a constant number of neighbors. The steps of building the rudimentary graph and the rudimentary tree is a variant of the distance-vector algorithm (also known as the distributed Bellman-Ford algorithm), which has been proven to converge and has time complexity of $O(VE)$, where V is the number of nodes and E is the number of links. Since the number of neighbors is constant for each node, the complexity for n nodes $O(n^2)$. Similarly, the overhead of control messages for constructing the rudimentary graph and tree is the same as that for the distance-vector algorithm, variants of which are commonly used in realistic networks (*e.g.,* Border Gateway Protocol). Hence, our protocol is clearly scalable, in terms of both time complexity and control overhead, to high numbers of group nodes. It is easy to see that all the procedures in the last step are dominated, in time complexity and control overhead, by the procedure of finding k unsaturated nodes. All other operations are constant time with respect to n (the number of nodes). To find k unsaturated nodes, it takes constant time if a constant number of random probes are successful; otherwise, a special version of a breadth-first search, described in Sec. IV-D, is executed. A node needs to exchange control messages with at worst $O(\log n)$ other nodes, since our breadth-first search primitive includes record-keeping at the nodes. Thus the control overhead in the worst case is $O(\log n)$. Overall, it is clear that our graph-construction algorithm is scalable to large multicast group sizes.

To handle discrepancy in bandwidths or rates of two incoming flows, we resort to existing flow control mechanisms (e.g., TCP) to synchronize the incoming flow rates, as it is traditionally done for matching incoming and outgoing rates in a flow-controlled reliable connection. We know that the end-to-end throughput in a multicast tree is determined by the minimum bandwidth link in the tree. Since two incoming flows synchronize rates in a 2-redundant multicast graph, the end-to-end throughput is *twice* the minimum bandwidth in the two (disjoint) paths. To resolve a difference in latency of two incoming signals, buffer is needed; the buffer size is proportional to the latency difference and is finite. Since the nodes in the overlay network are end systems (with abundant memory space), the issue of available memory for buffering is not likely to be significant. It is also possible and not hard to add optimization techniques to the existing algorithm to minimize the latency difference when finding a pair of disjoint paths during graph construction.

F. Linear coding multicast

Once a multicast graph is constructed, a set of linear codes must be found to realize linear coding multicast. Both Koetter *et al.* [6] and Li *et al.* [7] give algorithms for constructing the linear codes. However, because both papers are theoretical in nature, Li *et al.* [7] with an information-theoretic perspective while Koetter *et al.* [6] has an algebraic-geometric formulation, their briefly described algorithms have been included mainly for completeness. The algorithms are moreover centralized, difficult to implement in a distributed manner, and intended for general multicast scenarios. Since the graphs constructed by our algorithm are of a specific structure, a more light-weight

algorithm tailored for this specific type of multicast graphs can be devised.

We propose a distributed algorithm that is easy to implement for obtaining the linear codes for the 2-redundant multicast graph. We observe that coding is only needed at the source s and the intermediate nodes, and that an intermediate node has either one or two incoming edges (by construction). Furthermore, since each leaf receiver has exactly two paths from the source s, s sends the data vector (a, b) and both a and b should be obtained by each receiver. We assume that there is a function *gen*(m) that generates a sequence of m transformation vectors $\{(p_1, q_1)^T, (p_2, q_2)^T, \ldots, (p_m, q_m)^T\}$ such that

- p_i, q_i are elements in a field;
- (p_i, q_i) and (p_j, q_j) are linearly independent, $\forall i \neq j$;
- $(p_i, q_i)^T$ defines a linear transformation of data vector (a, b): $c_i = (a, b)(p_i, q_i)^T = p_i \cdot a + q_i \cdot b$ (p_i, q_i are coefficients of a linear combination of a, b), *i.e.*, $(p_i, q_i)^T$ determines a vector c_i, $\forall i$.

Hence c_i and c_j are linearly independent, $\forall i \neq j$. Let $C = \{c_1, \ldots, c_m\}$. Essentially, *gen*(m) generates codes for m linear transformations. For example, *gen*(5) could be $\{(1,0)^T, (0,1)^T, (1,1)^T, (2,1)^T, (1,2)^T\}$, then the corresponding C would be $\{a, b, a+b, 2a+b, a+2b\}$. Any two elements from C are linearly independent and therefore, a and b can be obtained from them. It follows that a leaf receiver is able to get both a and b as long as it receives *any two distinct* elements from the set C. We assign only one linear transformation to every intermediate node u, so u sends out the same data over all its outgoing edges.

- u has 1 incoming edge: the identity transformation is assigned, *i.e.*, data on the incoming edge is forwarded, with no encoding, on all the outgoing edges;
- u has 2 incoming edges: a transformation vector $v_u = (v_1, v_2)^T$ is assigned by the algorithm, so if x is received on one incoming edge and y on the other, then $(x, y)(v_1, v_2)^T = v_1 x + v_2 y$ is sent on all of u's outgoing edges.

We will now describe how the v_u's are obtained for those u with indegree 2. The distributed algorithm has two phases, *AssignCodes* and *DisseminateCodes*. Due to the special topology of the 2-redundant multicast graph, we can assume that the source s is multicasting a 2-dimensional data vector (a, b) to every leaf receiver. *Every* node $u \in A_I$ will determine a vector $w_u = (p_u, q_u)^T$ such that the data sent on its outgoing edges is $(a, b)(p_u, q_u)^T$. This way, a node u with indegree 2 can obtain these from its two parent nodes and together with its own w_u, it can easily obtain v_u, as will be shown later.

In the *AssignCodes* phase, s first multicasts a message through the rudimentary tree to initiate the *AssignCodes* phase. If an intermediate node u has 2 incoming edges, then it sends a message containing its address to s requesting a code. When enough time has passed for all the nodes to have a chance to send requests, suppose m requests were received and s has j children, then s generates $m+j$ linear codes using *gen*$(m+j)$ and sends to each requesting node one of the first m codes. This vector will be the w_u vector of u. The pseudocode is given in Table IV.

The w_i for nodes i with indegree 1 are determined in the *DisseminateCodes* phase (summarized in Table V), followed by obtaining v_u for every u, The last j vectors generated by *gen*$(m+j)$ are sent by s to its children, one to each child. A child i of s assigns the received vector to its w_i. Each node u with one incoming edge simply sets its w_u to the vector received and forwards it on all its outgoing edges. (Also, u has the identity transformation.)

Each node u with two incoming edges already has $w_u = (p_u, q_u)^T$ from the *AssignCodes* phase and passes it onto its outgoing edges. Node u also receives $(p_1, q_1), (p_2, q_2)$ on its incoming edges, respectively. Now u needs to determine $v_u = (v_1, v_2)^T$. Let α, β denote the data received on the two incoming edges, respectively, then $(\alpha, \beta)(v_1, v_2)^T$ is the data u sends out on its outgoing edges.

We know u should send out $(a, b)(p_u, q_u)^T$, but we also know

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} p_1 & p_2 \\ q_1 & q_2 \end{pmatrix} = \begin{pmatrix} \alpha & \beta \end{pmatrix}$$

We have

$$\begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} p_u \\ q_u \end{pmatrix} = \begin{pmatrix} \alpha & \beta \end{pmatrix} \begin{pmatrix} p_1 & p_2 \\ q_1 & q_2 \end{pmatrix}^{-1} \begin{pmatrix} p_u \\ q_u \end{pmatrix}$$

So the product of the matrix and vector on the right of (α, β) is the vector v_u. This is correct only if the matrix in the equation is invertible, *i.e.*, (p_1, q_1) and (p_2, q_2) are linearly independent. Two edges carry the same (p, q) only when they come out of the same intermediate node. But since the construction algorithm ensures that the paths containing one upstream node of u do not intersect with any path containing the other upstream node of u, except at s. Therefore (p_1, q_1) and (p_2, q_2) are linearly independent. The same logic applies to the leaf receivers, so the data on one incoming edge and the data on the other incoming edge are linearly independent. This, in fact, proves the correctness of the algorithm. Encoding at any intermediate node u is completely defined by the transformation vector v_u if u has indegree 2 and no encoding is done at intermediate nodes with indegree 1. Decoding at the leaf receivers is simple, because in the *DisseminateCodes* phase, each leaf receiver gets the codes from its two upstream nodes and can use these to decode the data they receive.

It only remains to find gen(m). We define gen(m) to be a set such that every (p, q) in the set is distinct and p, q are two prime numbers with $p \neq q$. It is clear that any two vectors are linearly independent, because they are only linearly dependent if one is a multiple of the other, which is impossible when they are not equal and are vectors of prime numbers. A function for generating primes in increasing order starting from 2 is used. It is straightforward to code such a function or find an existing efficient function. We give the algorithm in Table III.

Complexity analysis of linear codes algorithm

In the *AssignCodes* phase, one initiate-session message is multicast through a tree, *i.e.*, $O(n)$ transmissions of the message occur, where n is the number of nodes; then $O(m)$ messages are unicast between the source and intermediate nodes, where m is the number of intermediate nodes. The *DisseminateCodes* phase involves only each node sending one

TABLE III

Vector sequence generation ALGORITHM

Algorithm gen(m)
  $i \leftarrow 1$, $j \leftarrow 1$
  primes[i] $\leftarrow 2$, vectors[m] $\leftarrow (2, 2)$
  Iteration:
    Find the next prime p, smallest $p >$ primes[i]
    for $k = 1$ to i
      $j \leftarrow j + 1$
      vectors[j] $\leftarrow$ (primes[k], p)
      if $j = m$ **then** halt
      $j \leftarrow j + 1$
      vectors[j] $\leftarrow (p,$ primes[k])
    $i \leftarrow i + 1$
    primes[i] $\leftarrow p$

TABLE IV

AssignCodes PHASE

The source s:
  multicast message ⟨start_AssignCodes⟩ in the rudimentary tree
  set $T =$ (largest RTT) $\cdot 2$
  set timer $t = 0$, initialize $m = 0$
  while $t < T$ **do**
    Upon receiving ⟨request_code, *address of* u⟩:
      $m = m + 1$
      node_addr[m] = *address of* u
  $j =$ number of children of s
  obtain *gen*($m + j$)
    $= \{(p_1, q_1)^T, (p_2, q_2)^T, \ldots, (p_{m+j}, q_{m+j})^T\}$
  for $i = 1$ to m
    send ⟨code, $(p_i, q_i)^T$⟩ to node at
      address node_addr[i]
Upon receiving ⟨start_AssignCodes⟩, each node $u \in A_I$:
  if u has 1 incoming edge **then**
    do nothing
  else if u has 2 incoming edges **then**
    send message ⟨request_code, *address of* u⟩ to s
Upon receiving ⟨new_code, (p, q)⟩, each node $u \in A_I$:
  u sets its w_u: $p_u = p$ and $q_u = q$

message, including the codes assigned to it, to each of its downstream nodes. So the number of messages transmitted is exactly the number of edges in the multicast graph, the upper bound of which is $2n$. Hence, the total number of control messages transmitted for the two phases is $O(n)$. Let T denote the largest round-trip time or delay from the source to any node via the paths in the multicast graph. The largest round-trip time between any two nodes via unicast, let it be denoted by t, is obviously dominated by T. The *AssignCodes* phase requires at most time $3T$ for exchanging messages. While the *DisseminateCodes* phase requires only time t, since all the transmissions are done in parallel and no forwarding of messages is necessary. As such, the overall time complexity is $O(T)$.

V. Performance Evaluation

The primary objective of our evaluation process is to reveal the strengths and performance of our proposed algorithm with respect to network level metrics, and to compare with previously proposed representative multicast algorithms at both the application and the IP layer. At the application layer, we choose the Narada protocol [1] to evaluate the merits of our algorithm, since as one of the first algorithms proposed in the application layer, it uses a two-step mesh-based process that is similar to our work (a detailed discussion is postponed to Sec. VI). At the IP layer, we assume that IP multicast involves constructing classical shortest-paths trees (e.g., using DVMRP

TABLE V

DisseminateCodes PHASE

The source s (has j children and has gen($m + j$) from last phase):
  for $i = 1$ to j
    send ⟨code, gen($m + i$)⟩ to its ith child
Each node $u \in A_I$ with indegree 1:
  Upon receiving ⟨code, $(p, q)^T$⟩ from its parent:
    set its w_u: $p_u = p$, $q_u = q$
    send ⟨code, $(p_u, q_u)^T$⟩ out on all its outgoing edges, *i.e.,* to all its downstream nodes
    set linear transformation for all outgoing edges to the identity
Each node $u \in A_I$ with indegree 2:
  send ⟨code, $(p_u, q_u)^T$⟩ out on all its outgoing edges
  Upon receiving ⟨code, $(p_1, q_1)^T$⟩, ⟨code, $(p_2, q_2)^T$⟩, respectively, from its two upstream nodes:

$$v_u = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} p_1 & p_2 \\ q_1 & q_2 \end{pmatrix}^{-1} \begin{pmatrix} p_u \\ q_u \end{pmatrix}$$

[9]), composed of the reverse paths from the source to each receiver.

For the purpose of performance evaluation, we have resorted to simulation experiments, conducted using a locally written, message-level, event-based simulator. In accordance with the goal of simulating a realistic IP-based wide-area network, we have chosen the INET topology generator from the University of Michigan [10], which is fully capable of generating large-scale topologies that conform to the power-law characteristics [11]. Other topology generators may also be used, but we do not expect material deviations if the resulting topology maintains power-law properties.

With such a generated IP-based network topology as an underlying foundation, we selectively connect application-layer end hosts (*i.e.,* overlay nodes) to a subset of IP-layer nodes in the IP topology. We require that the IP-layer nodes in such a subset have very few links (usually just a single link) to other IP-layer nodes, such that they represent edge nodes in the wide-area network with a high probability, rather than core routers. For the purpose of application-layer multicast, each virtual link in the overlay topology represent an unicast path between two end hosts in the IP topology. With respect to network-level metrics, we use a similar simulation environment as Narada to facilitate more accurate comparisons: we assume identical availability of residual bandwidth on all physical links in the backbone IP topology, as well as randomly assigned link delays in the range of 8 – 12 ms.

Given IP-layer physical link delays and available bandwidth, it is straightforward to derive the delays and bandwidth of virtual links in the overlay. The delay of a virtual link is, obviously, the sum of physical link delays that the virtual link traverses. Calculating the bandwidth of a virtual link is more involved, however, since there may exist multiple virtual links sharing the same physical link (*stress*). In this case, the physical link bandwidth is equally divided among all the virtual links passing through. Finally, the available bandwidth on a certain virtual link is the minimum available bandwidth of all the physical links (*i.e.,* the physical link with the most *stress*) that it traverses. With the knowledge of delay and bandwidth of a virtual link, the 2-tuple *weight* of the link, detailed in Sec. IV, may be obtained.

We consider the following performance metrics in our

simulation:

- *Multicast session throughput.* We measure application-layer throughput at each of the receivers.
- *End-to-end delay.* We measure end-to-end latency from the source to the receivers, as perceived at the application layer, which naturally incorporates *stretch*.
- *Stress.* We measure stress on the physical links leading to overlay nodes, and also compute the *normalized stress* (the ratio of stress over the achievable session throughput).
- *Resource usage.* We measure the number of physical links, L, that are actively in service for a multicast session. The resource usage may be defined as $\sum_{i=1}^{L} d_i \cdot s_i$, where d_i is the delay of link i, and s_i is its stress. The resource usage is a metric that corresponds to the consumed network resources for data delivery to all receivers in a multicast session. The resource usage should also be normalized over session throughput.

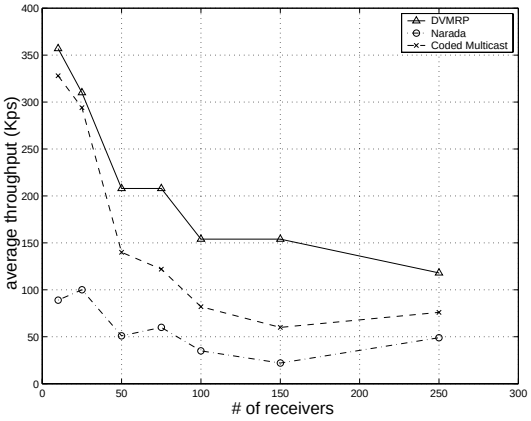


Fig. 6. Multicast session throughput at the receivers in (1) DVMRP; (2) Narada; (3) Coded Multicast.

We plot session throughput at the receivers as a function of the number of receivers. Our Coded Multicast scheme does not perform as well as DVMRP, as expected. The end-system nodes in overlay networks are often at the boundaries of the IP network, where there are fewer underlying physical links. Inevitably, a number of virtual links will map to the same physical link near an end-system, introducing stress on those few physical links around the end-systems. Stress creates bottleneck links that decrease session throughput. This is an inherent problem of application-layer multicast. Thus the figure shows that Narada has much lower throughput than DVMRP. Coded Multicast performs significantly better than Narada, consistently achieving at least doubled throughput for groups of almost all sizes; at times, even achieving more than twice Narada's throughput. This confirms our previous theoretical results that throughput in the multicast graph constructed by our algorithms can be twice as high as in a multicast tree.

In Fig. 7, the average end-to-end delay is plotted as a function of the number of receivers. It is expected that the end-to-end delay in our multicast scheme would be higher than the end-to-end delay for Narada and DVMRP. Because in our scheme, we use alternate paths which are likely not

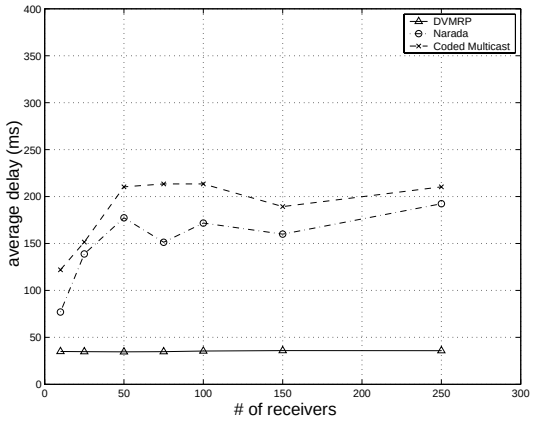


Fig. 7. End-to-end delay averaged over all receivers in (1) DVMRP; (2) Narada; (3) Coded Multicast.

as good as the shortest widest paths in the multicast tree, so they will certainly have higher delays. The delay for Coded Multicast is only slightly higher than the delay for Narada when both are compared to DVMRP. We observe that on average, the difference between Coded Multicast and Narada is only less than $1/4$ of the difference between Narada and DVMRP. Especially for groups of size less than 50 and greater than 150, the increase in delay is very small. Narada and Coded Multicast have end-to-end delay on the same order with both being higher than DVMRP. The increase in delay is slight and not proportional to the increase in throughput.

In the top graph in Fig. 8, the horizontal axis is the link stress and the vertical axis is the number of physical links for a given stress. The number of virtual overlay links for Coded Multicast is higher, so stress on physical links in proximity of the end-systems (overlay nodes) is bound to be higher. That is why there are more physical links with higher stress for Coded Multicast than for Narada. The difference in normalized link stress is not as pronounced, as can be seen from the bottom graph in Fig. 8 which plots normalized link stress as a function of the number of receivers. For groups of size less than 100, normalized link stress of Coded Multicast is actually roughly equal to that of Narada. Furthermore, as group size increases, Coded Multicast has the tendency of approaching Narada, which is grounds for optimism for Coded Multicast.

The normalized resource usage as a function of the number of receivers is plotted in Fig. 9. We observe that for groups of size less than 150, the normalized resource usage of Coded Multicast and that of Narada are comparable. For groups of sizes 150 to 250, the normalized resource usage of Coded Multicast is roughly $1/3$ higher than that of Narada. Both are high, in the order of a few thousands, as compared to DVMRP, which is in the order of a few hundreds.

Since the resource usage is defined to be a sum of products of delay and stress, it is a direct result of the data in Fig. 7 and Fig. 8 that resource usage will be higher for Coded Multicast. However, it can be seen from Fig. 9 that normalized resource usage for Coded Multicast has the same slope as that for Narada, only shifted up by a constant that is small in proportion to the absolute values of the resource usage. The

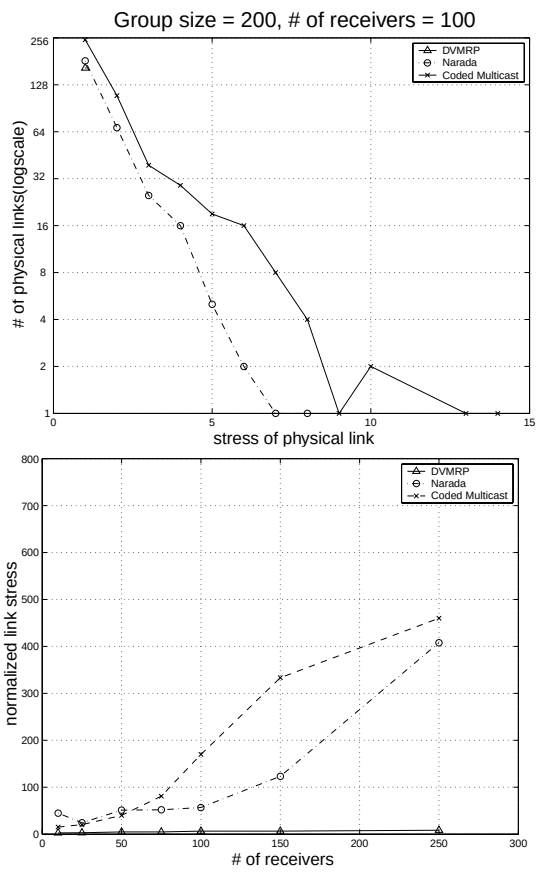


Fig. 8. Top figure: Link stress in (1) DVMRP; (2) Narada; (3) Coded Multicast. Bottom figure: Normalized link stress in (1) DVMRP; (2) Narada; (3) Coded Multicast.

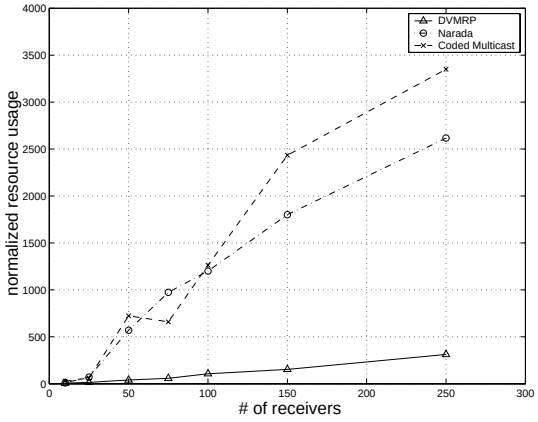


Fig. 9. Normalized resource usage in (1) DVMRP; (2) Narada; (3) Coded Multicast.

trend is that as group size increases, the resource usage of Narada and Coded Multicast are in the same order, and the constant difference will approach negligible.

In summary, the multicast scheme we propose, Coded Multicast, achieves twice the end-to-end throughput of Narada. The penalty incurred by Coded Multicast in end-to-end delay is not proportional to the significant increase in throughput. In all three metrics of delay, stress and resource usage, Coded Multicast performs worse than Narada by a small percentage. We believe that the slight additional penalty of using Coded Multicast is not commensurate with the considerable gain in throughput.

VI. Related Work

This work was mainly inspired by previous work on network coding, first proposed by Ahlswede *et al.* [5], and then developed in [7], [6], from information-theoretic and algebraic approach, respectively. It is shown that per-receiver max-flow throughput can be achieved (details summarized previously in Theorem 1) by applying network coding in a multicast (IP-layer) network. Although exciting insights are provided, the existing studies on network coding have remained largely theoretical, and we are not aware of any published work that studies the feasibility of applying the theoretical insights in network coding to increase throughput in actual multicast sessions over wide-area networks. In this paper, we give algorithms to achieve such a goal in application-layer overlay networks, supported by analytical and simulation results.

There exists an extensive body of research work in the area of multicast routing in wide-area IP networks [9], [12], [13], [14]. Because it has been shown that IP-based multicast lacks flexibility and is difficult to deploy in general, algorithms promoting application-layer overlay multicast have recently been proposed as remedial solutions, [1], [15], [16], [17], [18]. They focus on the issue of constructing and maintaining a multicast tree, with only unicasts between end hosts, and of minimizing the inefficiency brought forth by link stress and stretch. Researchers have recently focused on designing overlay multicast tree construction algorithms that are scalable, using tools including Delaunay Triangulations [4] and hierarchical clusters [3]. It is also possible to design overlay multicast algorithms based on structured overlay networks (ones that impose data on specific nodes based on hash functions), examples include overlay multicast [19] based on CAN [20], as well as Scribe [2] based on Pastry [21]. These approaches may incur performance penalty, and may not be adaptive to dynamic network metrics, which is shown to be critical in overlay multicast routing, [22], [23].

Inspired by two-step algorithms such as Narada [1], our algorithm begins with the construction of the *rudimentary graph* followed by the *rudimentary tree*. Our approach is also similar to most of the existing proposals in the sense that it is a distributed algorithm. However, our proposal distinguishes from all previous work in the following fundamental aspect: *we construct a* ***multicast graph****, rather than a tree*. Although the idea of multiple paths has been studied in the area of distributed Quality-of-Service routing ([8], [23], [24], [25]), none of the previous work had sought to utilize the path diversity *concurrently* to increase end-to-end throughput. Among all previous work, perhaps the work on CoopNet [26] and the position proposal on SplitStream [27] are most similar to our work. Both papers have proposed to utilize multiple multicast trees to deliver striped data, using either multiple description coding or source erasure codes to split content to be multicast. CoopNet proposes a centralized algorithm which does not feature support of optimizing link stress and stretch.

SplitStream proposes a decentralized algorithm to construct a *forest* of multicast trees, with a focus on per-node load balancing. In both work, the inherent concerns of throughput limitations caused by conflicting paths have not been addressed. In comparison, our algorithm constructs an *acyclic multicast graph* from one multicast source, which, combined with coding, introduces a smaller degree of stress on overlay nodes compared with a forest. Further, our algorithm seeks a well-balanced trade-off between the constraints on link stress and the selection of good paths to achieve high throughput. Network coding, while essential to the performance of our algorithm, has not been incorporated in either CoopNet or SplitStream.

There has been research conducted on parallel downloads in peer-to-peer or overlay networks from mirror sites, *i.e.*, multiple sources, in [28], [29], [30]. These papers investigate a different problem than that of overlay multicast that we study in this paper. Parallel downloads consider multiple servers containing replicated content (sometimes in encoded form), and clients obtain desired content by connecting to these servers. In our work, we only have one source holding the data. Instead of parallel unicasts from sources to clients, we construct a topology so that clients do not only receive by unicast from the source. Our approach does not assume that multiple sources are available, and provides more flexibility in efficiently distributing data to receivers by providing a multicast topology that includes multiple hops and store-and-forward routing actions in the nodes.

VII. Concluding Remarks

In this paper, we have proposed a set of distributed algorithms to significantly improve end-to-end multicast session throughput. Such results are achieved by the application of network coding when exploiting path diversity with two disjoint paths to each multicast group receiver. To the best of our knowledge, there do not exist similar proposals in previous literature. With respect to the effectiveness and performance of our algorithm, we have undertaken both analytical and simulation-based studies, which agree with our original claims. We are currently in the process of implementing our algorithms as an application-layer protocol on the wide-area overlay network testbed *PlanetLab* [31].

References

[1] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, pp. 1456–1471, October 2002.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communications*, pp. 1489–1499, October 2002.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.

[4] J. Liebeherr, M. Nahas, and W. Si, "Application-Layer Multicasting With Delaunay Triangulation Overlays," *IEEE Journal on Selected Areas in Communications*, pp. 1472–1488, October 2002.

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

[6] R. Koetter and M. Medard, "Beyond Routing: An Algebraic Approach to Network Coding," in *Proc. of IEEE INFOCOM*, 2002.

[7] S.-Y. R. Li and R. W. Yeung, "Linear Network Coding," *IEEE Trans. on Information Theory, to appear*, 2002.

[8] Z. Wang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, September 1996.

[9] S. Deering, "Multicast Routing in Internetworks and Extended LANs," in *Proc. ACM SIGCOMM*, August 1988.

[10] J. Winick, C. Jin, Q. Chen, and S. Jamin, "INET: an Autonomous System (AS) level Internet Topology Generator, version 3.0," *available online at http://topology.eecs.umich.edu/inet/*, June 2002.

[11] C. Faloutsos, M. Faloutsos, and P. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *Proc. ACM SIGCOMM*, 1999.

[12] A. J. Ballardie, P. F. Francis, and J. Crowcroft, "Core Based Trees," in *Proc. ACM SIGCOMM*, August 1993.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An Architecture for Wide-Area Multicast Routing," in *Proc. ACM SIGCOMM*, August 1994.

[14] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP Architecture for Interdomain Multicast Routing," in *Proc. ACM SIGCOMM*, August 1998.

[15] Y. Chawathe, "Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service," *Ph.D. Dissertation, Univ. California, Berkeley, CA*, 2000.

[16] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," in *Proc. 4th Symposium Operating System Design and Implementation (OSDI)*, October 2000, pp. 197–212.

[17] P. Francis, "Yoid: Your Own Internet Distribution," *available online at http://www.aciri.org/yoid/*, April 2000.

[18] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," in *Proc. of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, San Francisco, CA, March 2001, pp. 49–60.

[19] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast Using Content-addressable Networks," in *Proc. 3rd Int. Workshop on Networked Group Communication (NGC '01)*, London, UK, 2001.

[20] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. ACM SIGCOMM*, 2001, pp. 149–160.

[21] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems," in *Proc. IFIP/ACM Middleware 2001*, November 2001.

[22] S. Shi and S. Turner, "Multicast Routing and Bandwidth Dimensioning in Overlay Networks," *IEEE Journal on Selected Areas in Communications*, pp. 1444–1455, October 2002.

[23] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of Service sensitive Multicast Internet protoCol," in *Proc. ACM SIGCOMM*, 1998.

[24] P. Baccichet, E. Pagani, and G. P. Rossi, "Quality of Service Multipath Multicast Protocol," in *Proceedings of the Fourth International Workshop on Networked Group Communication*, Boston, Massachussetts, 2002.

[25] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions," *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, vol. 12, no. 6, pp. 64–79, November/December 1998.

[26] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002)*, Florida, 2002.

[27] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Content Distribution in a Cooperative Environment," in *Proc. of the Second International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, Berkeley, California, February 2003.

[28] J. Byers, M. Luby, and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads," in *Proc. of IEEE INFOCOM*, 1999.

[29] P. Rodriguez and E. W. Biersack, "Dynamic Parallel-Access to Replicated Content in the Internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 455–465, August 2002.

[30] P. Maymounkov and D. Mazieres, "Rateless Codes and Big Downloads," in *Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[31] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proc. of the First Workshop on Hot Topics in Networks (HotNets-I)*, 2002.