Restricted Dynamic Steiner Trees for Scalable Multicast in Datagram Networks

Ehud Aharoni and Reuven Cohen, *Member, IEEE*

Abstract— **The paper addresses the issue of minimizing the number of nodes involved in routing over a multicast tree and in the maintenance of such a tree in a datagram network. It presents a scheme where the tree routing and maintenance burden is laid only upon the source node and the destination nodes associated with the multicast tree. The main concept behind this scheme is to view each multicast tree as a collection of unicast paths and to locate only the multicast source and destination nodes on the junctions of their multicast tree. The paper shows that despite this restriction, the cost of the created multicast trees is not necessarily higher than the cost of the trees created by other algorithms that do not impose the restriction and therefore require all nodes along the data path of a tree to participate in routing over the tree and in the maintenance of the tree.**

Index Terms—**Dynamic trees, multicast routing, routing scalability, Steiner trees.**

I. Introduction

IN A POINT-TO-POINT communication network, multicast is treated as the problem of creating, maintaining, and updating efficient multicast trees, rooted at the multicast source nodes and spanning the groups of destination nodes. A multicast tree is dynamically created as network nodes join and leave the destination group. It is maintained by some of the network nodes, mainly those sitting on its data path. It needs to be updated following changes in the connectivity or in the load of the underlying network topology.

Internet employs the distance vector multicast routing protocol (DVMRP) [9] in order to construct a shortest-path tree for every multicast {source, group} pair. This protocol, which is a "pruning" variant of the reverse path-forwarding (RPF) algorithm [7], suffers from two main scaling problems as follows [2], [8]. First, each network node has to maintain routing information for every multicast tree.[1] Second, prune messages should be periodically exchanged between the network nodes in order to maintain an updated version of each multicast tree.

The protocols presented in [2], [4], and [8] address these problems. A tree spanning the members of the group is established in the network, and only the nodes sitting on the data path of the tree are required to maintain routing information associated with the tree. When a new node joins the multicast group, all of the nodes on the shortest path between the new node and some node on the tree (the predetermined core in [2], the rendezvous point(s) in [8], or the closest node in [4]) join the tree and update their multicast routing tables accordingly.

A. Tunnel-Based Multicast

When multicast becomes pervasive, more techniques must be employed in order to avoid scaling problems. If thousands of trees are created, deleted, and updated every minute, the processing burden on the network nodes will be excessive, even if the burden associated with every multicast tree is laid only upon the nodes sitting on the data path of that tree. This is because these nodes will have to respond very often to join or delete requests and to changes in the load or connectivity of the underlying network. This burden can be significantly reduced using the concept of tunneling [19] as follows. The multicast packets are routed over a tree formed by a collection of *multicast links* (tunnels), where each multicast link is a simple unicast route. The intermediate nodes of each multicast link do not have to maintain routing information for the multicast tree because they are involved only in regular unicast routing. Only the end points of each multicast link have to maintain routing information for every tree established over the link. For instance, consider a multicast source s, a multicast group $Z = \{z_1, z_2, z_3, z_4, z_5\}$, and the multicast tree depicted in Fig 1. This tree can be viewed as a collection of the following six multicast links: $s \rightsquigarrow b$, $b \rightsquigarrow z_2$, $b \rightsquigarrow z_1$, $z_1 \rightsquigarrow d$, $d \rightsquigarrow z_5$, $d \rightsquigarrow z_3$, and $d \rightsquigarrow z_4$. The end points of these multicast links are $Z \cup \{s\} \cup U$, where U is the set of nodes not in $Z \cup \{s\}$ that have at least three links in the tree (b and d in Fig. 1). Packets sent by the source s have two pairs of source/destination addresses. One pair is of the multicast source s and multicast group Z. This pair remains fixed for the entire routing. The other pair is the addresses of the source and destination of the multicast link over which the packet is routed. This pair is replaced when the packet reaches the end of the multicast link. In terms of Fig. 1, a multicast packet created by the source has a multicast address pair[2] (s, G) and a unicast address pair (s, b). The packet is routed through a based on the unicast address pair. Node a does not need to know anything about the multicast destination G. When the packet is received by the unicast destination b, it is processed by the multicast routing procedure. In the multicast table of b it is indicated that a multicast packet from s to G should

Manuscript received February 21, 1997; revised September 22, 1997 and February 3, 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Ammar.

The authors are with the Department of Computer Science, Technion, Haifa 32000, Israel (e-mail: rcohen@cs.technion.ac.il).

Publisher Item Identifier S 1063-6692(98)04141-7.

[1] It is assumed that all network nodes have multicast routing capability.

[2] Throughout this paper, G represents a multicast address whereas Z represents a dynamic set of nodes that want to receive the multicast packets sent by s to G.

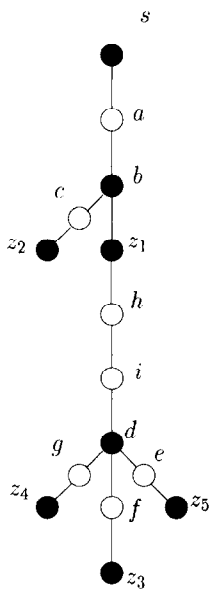


Fig. 1. A multicast tree.

be forwarded to z_2 and to z_1. The copy destined for z_2 has the unicast address pair (b, z_2). It is sent to c and routed as a regular unicast packet whose destination is z_2. At z_2, the packet is processed by the multicast routing procedure. The latter uses its multicast routing table and deduces that the packet is destined for the local node and that it should not be forwarded elsewhere.[3] Node z_1, in contrast, deduces from its multicast routing table that it needs to keep a copy of the packet for itself and to send another copy on the multicast link $z_1 \rightsquigarrow d$.

The concept of tunneling is already used for multicast routing in the Internet multicast backbone (MBONE) [11] and is also proposed in the context of asynchronous transfer mode (ATM) [1]. However, its purpose is not to reduce the maintenance and routing burden at the multicast nodes, but to connect multicast-capable nodes through the general Internet protocol (IP) network. If all of the Internet routers had multicast capability, as in the model considered in this paper, tunneling would not have been used.

It can be easily shown by induction that in a multicast tree rooted at source s and spanning the group Z, the number l of multicast links (tunnels) is $|Z| \leq l \leq 2|Z| - 1$. In addition, only less than half of the nodes that are actively involved in the maintenance of the tree and in the routing of multicast packets from s to Z (i.e., the nodes whose multicast procedure is invoked when a multicast packet from s to G is received) do not belong to $Z \cup \{s\}$. It turns out, however, that in a datagram network it is practically impossible to keep the details of a multicast tree transparent to the intermediate nodes of the multicast links. When a new member v joins an existing multicast group, it might need to connect to some intermediate node v', say, of a multicast link $v'' \rightsquigarrow v'''$, in which case $v'' \rightsquigarrow v'''$ is removed from the tree and three multicast links $v'' \rightsquigarrow v'$, $v' \rightsquigarrow v'''$, and $v' \rightsquigarrow v$ are added. For this process to take place, node v' needs to know that it participates in the multicast tree and to maintain some parameters describing its position in the tree, like the identities of the end nodes of the multicast link for which it acts as an intermediate node. This information needs to be refreshed not only when multicast links are added or removed due to changes in the destination group, but also when the unicast routing tables change. For instance, if in Fig. 1 the unicast routing tables are updated such that the shortest path from d to z_4 goes through nodes j and k rather than through g, nodes g, j, and k must be updated. The conclusion is that the *concept of tunneling does not remove the tree maintenance burden from the intermediate nodes of the multicast links.* It just removes from these nodes the memory and processing burden associated with multicast routing decisions.

[3] In terms of the Internet, z_2 as well as any other node in the multicast group Z can be viewed as a border router connecting an autonomous network of some destination host(s) to the Internet backbone. Thus, the packet will be forwarded by z_2 into the autonomous network and multicast to the destination hosts by means of an internal multicast protocol [18].

B. The Proposed Scheme

In this paper we suggest an approach to solve the scalability problem of establishing, updating, maintaining, and routing over a great number of multicast trees in a datagram network. According to this approach, for every multicast tree rooted at a source node s and spanning a set Z of destination nodes, only the nodes in $\{s\} \cup Z$ need to keep and update information related to the tree routing and maintaining. This is achieved by *viewing a multicast tree as a collection of multicast paths (tunnels)* and ensuring that *only nodes in* $\{s\} \cup Z$ *are selected as end points of the multicast links* (namely, the set U mentioned above is empty). In terms of Fig. 1, this means that nodes b and d, which do not belong to $\{s\} \cup Z$, cannot function as end nodes of multicast links but only as intermediate nodes.

The main advantage of the proposed scheme is that for every multicast tree, only the nodes in $\{s\} \cup Z$, which *in any case must keep routing and maintenance information regarding the tree*, are required to do so. The existence of the tree is absolutely transparent to any other network node, even to those nodes that sit on the tree data path and route packets of the tree as intermediate nodes of multicast links. Neither changes in the unicast routing tables that affect the routes between the end points of multicast links nor changes in the multicast destination group can put any maintenance burden on nodes not in $\{s\} \cup Z$.

The constraint imposed by the proposed scheme, where only nodes in $\{s\} \cup Z$ can be located in the tree junctions and serve as end nodes of the multicast links, may lead to the creation of inefficient multicast trees. This issue is extensively studied in the paper. As will be shown, the differences between the cost of the trees generated without this constraint and the trees generated under this constraint are minor and may therefore justify the significant reduction in the maintenance overhead.

The problem of establishing a low-cost tree spanning a partial set of the network nodes is known as the Steiner tree problem. When the set of nodes dynamically changes, a different problem—referred to as the dynamic Steiner tree (DST)—is defined [14]. Section II discusses these problems and presents the dynamic greedy algorithm (DGA) [14] for

establishing and updating a low-cost multicast tree when the destination group dynamically changes and when there is no restriction to locate only nodes from $\{s\} \cup Z$ in the junctions of the multicast trees. In Section III we present a new algorithm, referred to as restricted DGA (R-DGA), that locates only nodes from $\{s\} \cup Z$ in the junctions of the multicast trees. Based on R-DGA, we then present a possible protocol for creating, updating, and maintaining low-cost multicast trees. In Section IV the performance of R-DGA is studied. We first prove that despite the restriction, the worst-case performance ratio of R-DGA is the same as that of DGA. Then we present simulation results that show that the differences in the actual cost of the trees established by a slightly improved version of R-DGA (improved R-DGA, described in Section IV-B) and those established by DGA are not significant. We also calculate the average number of nodes not in $\{s\} \cup Z$ that are included in the trees generated by DGA and by the shortest-path algorithm (SPATH) (as suggested in [2] and [8]). We use the result in order to compare the burden laid on an average network node by R-DGA and by the other protocols. SPATH is a simple algorithm that connects each joining node to the tree via the shortest path to the source. We refer to it and its performance in several places throughout the paper. A detailed study of its performance can be found in [10]. In Section V we discuss the problem of creating, maintaining, and updating low-cost multicast trees that guarantee some upper bound on the distance between the source node and every destination node. Such a requirement might be of importance for many multicast applications. A new algorithm, referred to as constrained R-DGA (C-R-DGA) is presented and its performance is studied. Section VI concludes the paper.

II. The Dynamic Steiner Tree Problem

The Steiner tree problem can be formulated as follows:

Given a graph $G(V, E)$, a nonnegative weight for each $e \in E$, and a subset $Z \subseteq V$, find a subnetwork T of G such that there is a path between every pair of vertices in Z, and the total cost of T is a minimum.

The Steiner tree problem is NP-complete [12]. It remains NP-complete even if all edge weights are equal. Several heuristics are known for this problem [22]. One of them, suggested in [13] and [20], is the minimum cost paths heuristic (MPH). This heuristic has a worst-case performance of two times the optimum cost solution [5]. No heuristic with a better worst-case performance is known [22]. The MPH works as follows [22].

Step 1: Choose an arbitrary vertex z from Z. Let $Z' = \{z\}$ and $T = \{z\}$.

Step 2: Find in $Z - Z'$ the vertex z closest to T. Add z to Z', and add to T the minimum cost path joining z to Z.

Step 3: If $Z' \neq Z$, return to Step 2; otherwise, T is the solution.

When the set Z represents a multicast group, it cannot be assumed to be known in advance since nodes can dynamically join and leave the multicast group. This is known as the DST problem, formally stated as follows [14]. Let $R = \{r_0, r_1, \cdots, r_K\}$ be a sequence of requests, where each r_i is a pair (v_i, ρ_i), $v_i \in V$, $\rho_i \in \{\text{add}, \text{remove}\}$. Let Z_i be the set of nodes in the multicast group after step i, consisting of every node v for which there exists $j \leq i$ such that $r_j = (v, \text{add})$ and $r_l \neq (v, \text{remove})$ for all $j < l \leq i$. Then,

Given a graph $G(V, E)$, a nonnegative weight for each $e \in E$ and a sequence R of requests, find a sequence of multicast trees $\{T_1, T_2, \cdots, T_K\}$ where T_i spans Z_i and has a minimum cost.

DST can be solved using any heuristic for the static Steiner tree problem, like the MPH mentioned above, if we allow the multicast tree to be completely rebuilt after each change. This is, however, an unrealistic approach since it requires a lot of coordination among the network nodes [3], and it is very likely that a new request r_i will come up before T_{i-1} is ready. The nonrearrangeable version of DST, referred to as DST-N [14], requires that if r_i is an add request, T_i must include T_{i-1} as a subgraph (namely, no link can be removed from the old tree), whereas if r_i is a remove request, T_{i-1} must include T_i as a subgraph (namely, no link can be added to the old tree).

Imase and Waxman have shown [14] that when the request sequence consists of only add requests, for any algorithm A that solves DST-N there exists an instance such that for every i $0 < i \leq K$

$$\frac{A(Z_i)}{\text{OPT}(Z_i)} \geq 1 + \frac{1}{2}\lfloor \log(|Z_i| - 1) \rfloor$$

where K is the length of the request sequence, $\text{OPT}(Z_i)$ is the optimum solution for a tree spanning Z_i, and $A(Z_i)$ is the cost of a tree spanning Z_i created by A. They have also presented the DGA, where a new node appended to the multicast group is connected to the existing multicast tree through the cheapest path leading to any node in the tree, and shown that DGA has a worst-case performance ratio (competitiveness) of $\lceil \log(|Z_i|) \rceil$—that is within two of the optimal algorithm.

If both add and remove requests are allowed and if the tree cannot be rearranged, no upper bound on the worst-case performance ratio exists [14]. Several algorithms [4], [14], [15] have been proposed in order to accommodate both add and remove requests while restricting the number of rearrangements required in order to derive a new efficient tree T_i from the old one T_{i-1}.

III. R-DGA Heuristic for Scalable Multicast

In this section we describe a new algorithm, referred to as the R-DGA, and then present an R-DGA-based protocol for creating and maintaining dynamic multicast trees.

A. The R-DGA Algorithm

Throughout this paper we assume that unicast routing from v to v' is performed over the shortest path $v \rightsquigarrow v'$. We also assume that every node v' knows the cost of the shortest path $v \rightsquigarrow v'$ from every other network node v. This kind of information can be provided by the protocol that updates the unicast routing tables, even in the general case where the cost

function and the delay function are different, or the costs of $v' \rightsquigarrow v$ and $v \rightsquigarrow v'$ are different.

R-DGA responds differently to add requests and to remove requests. When a request r_i to add a node v to the existing multicast group Z_{i-1} is received, R-DGA uses DGA with a small but significant change: the new node v can be connected *only to a node in the former multicast group* Z_{i-1}. The cost of the path $v' \rightsquigarrow v$ is checked for every $v' \in Z_{i-1}$. The node $v' \in Z_{i-1}$ for which the cost of $v' \rightsquigarrow v$ is minimum is selected as the parent of v, and a new multicast tree T_i is created by appending the multicast link $v' \rightsquigarrow v$ to the previous multicast tree T_{i-1}. When a request r_i to remove v from a multicast group Z_{i-1} is received, v is removed from T_{i-1} along with all of the multicast links connecting v to its parent and to its children. Then, every child v' of v in T_{i-1} is connected to a new parent $v'' \in Z_i(v')$ from which the shortest path is of minimum cost. The set $Z_i(v')$ is a subset of $Z_i = Z_{i-1} - \{v\}$ consisting of all of the nodes in Z_i that have joined the tree (in the last time) before v'. The selected node v'' becomes the new parent of v in T_i by appending the multicast link $v'' \rightsquigarrow v$ to T_{i-1}.

This algorithm is demonstrated in Fig. 2. The original graph is shown in Fig. 2(a), and the multicast source is node s. The number near every link indicates the cost and the delay on the link for both directions. When node g is added to the multicast group, it joins the tree through the unicast path multicast link $s \rightsquigarrow g$ whose cost is six. When node c is added, it can join the existing tree by a multicast link from s or from g. Since the latter is cheaper (three versus seven), c joins the tree through the multicast link $g \rightsquigarrow c$. Packets from the source s to c will be routed through $s \rightsquigarrow g$ and then through $g \rightsquigarrow c$, namely, over the path $s-f-g-f-c$. Other multicast algorithms, like those based on DGA or on the SPATH [2], [8], [14] would avoid this unnecessary loop between f and g by creating three multicast links $s \rightsquigarrow f$, $f \rightsquigarrow g$, and $f \rightsquigarrow c$. However, this is the price we pay by requiring that only the source node and destination nodes will be located in the forks of the tree. The next node added to the multicast group is d. It can join the existing tree by a multicast link from s, from g, or from c. The cost of $s \rightsquigarrow d$ is eight, of $g \rightsquigarrow d$ is two, and of $c \rightsquigarrow d$ is three. Thus, $g \rightsquigarrow d$ is added to the tree, and the resulting tree is shown in Fig. 2(b). Next, suppose that g is removed from the multicast group. The multicast links $s \rightsquigarrow g$, $g \rightsquigarrow c$, and $g \rightsquigarrow d$ are removed, and new parents should be selected for c and d. For node c, only node s can serve as the new parent of c, because no node in the updated destination group has joined the group in the last time before c. Thus, the multicast link $s \rightsquigarrow c$ is added to the tree. Regardless of the new parent selected for c, both s and c can become the new parent of d. Since $c \rightsquigarrow d$ is cheaper than $s \rightsquigarrow d$ (three versus eight), $c \rightsquigarrow d$ is added to the tree. The resulting tree is shown in Fig. 2(c).

It is interesting to note that the problem of sending the same multicast data over the same link more than once arises also in the context of ATM, when the multicast tree is established over a set of ATM point-to-point virtual paths [1]. If the number of virtual paths (VP's), which are the equivalent to the multicast links in the present paper, has to be minimized, a VP will be established only between members

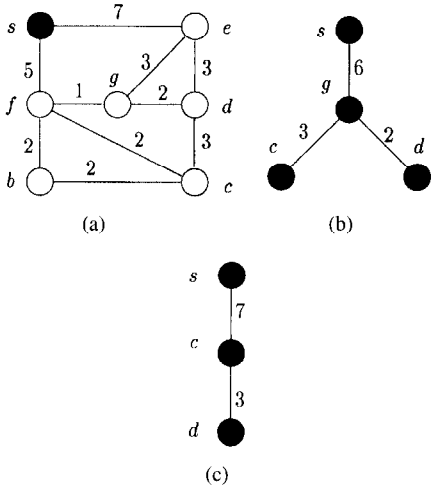


Fig. 2. R-DGA execution example. (a) Network graph. (b) g, c, and d are added. (c) g is deleted.

of the multicast group, as with R-DGA. In [1] this problem is addressed with an emphasis on possible enhancement to the VP concept (introducing VP's with intermediate exits) and on asymmetric VP's, whereas in the present paper the problem is addressed with an emphasis on the dynamic nature of the group and on the tradeoff between the routing burden and the communication cost.

B. An R-DGA Protocol

A possible protocol based on R-DGA for creating and maintaining a multicast tree is as follows. A node v wishing to join a multicast group G sends a request message to the source s over a reliable unicast [transmission control protocol (TCP)] connection. The source s responds with a list Z_{i-1} of the identities of the nodes currently in G. Then it generates an updated list $Z_i \leftarrow Z_{i-1} \cup \{v\}$. When node v receives the response from s, it selects as a parent the node u from $Z_{i-1} \cup \{s\}$ for which $u \rightsquigarrow v$ is of minimum cost. It then sends a message asking u to be its parent. Node u updates its multicast routing table by adding the multicast link $u \rightsquigarrow v$. Consequently, when u receives later a message for G, it will send one copy to v over the unicast route $u \rightsquigarrow v$. When a node v wants to leave G, it sends a remove request to s. It attaches to this request a list of identities of its children in the current tree. Node s generates an updated list Z_i of the multicast group by removing v from Z_{i-1}. Then, for every child u of v in the existing tree, s determines the list $Z_i(u)$ of nodes that had joined G before u and have not left it yet. A response message is then sent back to v, containing a different list for every child of v. Node v then sends a message to every child u, asking u to find a new parent from the list $Z_i(u)$ provided by s. The multicast link from v to u can be removed either when v asks u to find a new parent or after u notifies v that it has a connection to the tree through a new parent. In either case some of the packets sent by s might not be received by u due to the change.

The protocol as described so far retains the correct structure of the tree even if multiple requests are received at the source simultaneously. The requests are processed by the source

sequentially and a response message is sent back to the sender of each request. The request senders, and their children in case of a remove request, can continue their process distributedly, independently of each other. It might happen therefore that for a short period of time, a multicast link $v \rightsquigarrow v'$ is created before v' is connected to the tree. In contrast, cycles of multicast links cannot be created, even for a short time. Hence, looping of multicast packets is not possible, unless some multicast link creates a self-loop due to inconsistency of the *unicast* routing tables. Avoiding this inconsistency is, of course, the task of the unicast routing algorithm rather than of the multicast routing algorithm.

C. The Properties of R-DGA

In the following we discuss some of the properties of this protocol. The most important property is that the protocol can handle many multicast trees that dynamically change because it requires that a minimum number of nodes will participate in the multicast routing over each tree and in maintaining and updating each tree. This is achieved in three steps. Firstly, in order to avoid keeping at every network node routing and maintenance information associated with every active group (as in MBONE), the multicast trees are set up and updated dynamically, according to the exact structure of the destination group. This removes from the picture those network nodes that do not sit on the tree data path. Secondly, the concept of multicast over unicast paths (tunneling) is employed in order to avoid routing burden from nodes sitting on the tree data path. Finally, only the nodes directly related to the multicast tree, namely, the source and the destinations, are located in the end points of the multicast links. This eliminates the routing and maintenance burden from all the other nodes in the tree. The result of these three steps is that only those nodes directly related to each multicast tree will have to encounter the burden associated with routing over the tree, maintaining the tree, and updating the tree following changes in the destination group or in the underlying unicast routing.

Another important property of the protocol is that the established trees are efficient. At first glance it seems that due to the restriction to have only nodes from $\{s\} \cup Z$ in the junctions of the tree, the trees would be of a high cost. This hypothesis can be supported by the example in Fig. 2 as discussed before. However, the next section shows that *despite the restriction, the worst-case performance ratio of R-DGA is the same as that of DGA*. It also shows that *the differences in the actual cost of the trees established by a protocol similar to R-DGA and the trees established by DGA are minor and, in many cases, do not exist at all.*

A third property of the protocol is its *fast* response to changes in the destination group. When a new node is added, a new multicast link is created without affecting the rest of the tree. When a node is removed, then unlike other algorithms that try to minimize the *number* of rearrangements [4], [14], [15], R-DGA minimizes the rearrangement *time*. All of the affected nodes, whose parent has left the group, join the new tree independently of each other. Consequently, the time needed to rearrange the tree following a remove request is roughly equal to the time needed to address a single add request. As for the *number* of rearrangements, while it may be as high as $|Z| - 1$ for extreme cases, it is less than one on the average case. This is because the average degree of a node in a tree is less than two and, therefore, the average number of children of a node in the multicast tree is less than one.

Finally, the protocol is not affected by unicast routing changes, except in those rare cases where the connectivity between end points of a multicast link is broken. If the unicast route between the end points of a multicast link changes, the tree might be less or more efficient, and the changes will be taken into consideration when subsequent add/remove requests are accommodated. However, since the protocol does not consider the intermediate nodes of a multicast link as part of the tree, the unicast routing changes will be transparent to the tree. If a multicast link $u \rightsquigarrow v$ is broken due to loss of connectivity, node v will access the source s and will be treated as a node whose parent has left the tree.

IV. The Performance of R-DGA

In this section the performance of R-DGA is compared to the performance of DGA. We first compare the worst-case performance of both algorithms and then present simulation results for the average case. As stated in Section II, when the request sequence consists of only add requests, DGA has a worst-case performance ratio (competitiveness) of $\lceil \log(|Z_i|) \rceil$, where i is the number of add requests and $|Z_i|$ is the number of nodes in the destination group after these requests are handled. If both add and remove requests are allowed, no upper bound on the worst-case performance ratio exists. Both claims are proven in [14].

A. Worst-Case Performance Analysis

In the following we prove that any bound that applies to the performance of DGA on every sequence of only add requests applies to R-DGA as well. Moreover, such a bound applies to R-DGA even if the sequence also contains remove requests. Recall that since we employ the unicast routing as the underlying layer of the multicast routing, R-DGA can use only the shortest path between two nodes, regardless of its cost. In order to compare the performance of these algorithms we shall assume throughout this section that the cost function is proportional to the delay function. This implies that the shortest path between two nodes is also the cheapest path. A private case of this assumption would be to consider equal delay and cost functions.

Lemma 1: When requests are restricted to additions only, and the networks are restricted to complete graphs that satisfy the triangle inequality, the cost of a tree generated by R-DGA is equal to the cost of a tree DGA may generate.

Proof: We prove by induction on the length of the request sequence that the same tree generated by R-DGA *can* be generated by DGA as well. For an empty sequence of requests, the claim is obviously correct. When a new node v joins the tree, both algorithms will connect it to the closest node in the tree. Because the trees are, so far, identical, DGA has the *option* of selecting the same node u selected by R-

DGA, though there might be other nodes in the tree having the same minimum cost/distance to v. Since the graph is complete and it satisfies the triangle inequality, the shortest path from v to u is the single edge connecting them. No nonmember nodes are in this path and, therefore, the two trees continue to be identical. □

Lemma 2: When requests are restricted to additions only, any bound on the worst-case ratio of DGA as a function of the number of requests ($|Z_i|$) is a bound for R-DGA's worst-case ratio as well.

Proof: Consider a network $G(V, E)$ with N nodes joining the multicast tree in some order. Suppose that R-DGA builds a tree with cost L. Let $G'(V, E')$ denote the complete distance graph of G. For every two nodes u and v, G' has an edge $u \rightarrow v$ whose cost is equal to the cost of $u \rightsquigarrow v$ in G (i.e. the cost/distance of the shortest path from u to v), and an edge $v \rightarrow u$ whose cost is equal to the cost of $v \rightsquigarrow u$ in G. The optimal solutions for G and G' are of identical cost OPT, since any tree in G spanning a set of nodes can be transformed to a tree in G' spanning the same nodes and having the same cost, and vice versa. Also, running R-DGA on G' with the same sequence of add requests yields the same cost L. This is because R-DGA needs as input only the distance matrix of the graph nodes rather than the entire set of edges. Since the cost of a path from v to u in G' is the same as in G, G and G' are identical as far as R-DGA is concerned. G' is a complete graph satisfying the triangle inequality. Hence, according to Lemma 1, the cost of the tree generated by R-DGA in G is equal to the cost of a tree that DGA *may* generate in G'. Since any bound on the worst-case ratio of DGA as a function of the number of requests ($|Z_i|$) is applicable for any graph, and in particular for G', the lemma is proven. □

The logarithmic bound on DGA's performance holds only if remove requests are not allowed [14]. We now prove that R-DGA maintains the same bound even when remove requests are allowed.

Lemma 3: Any tree generated by R-DGA following an *add/remove* sequence of requests and spanning the set of nodes Z can be generated by R-DGA with a sequence of $|Z|$ *add* requests only.

Proof: We consider the output of the algorithm on the following two sequence requests:

1) $(v_1, \text{add}), \cdots, (\text{v}_\text{n}, \text{add}), (\text{v}_\text{i}, \text{remove})$;
2) $(v_1, \text{add}), (\text{v}_2, \text{add}), \cdots, (\text{v}_{\text{i}-1}, \text{add}), (\text{v}_{\text{i}+1}, \text{add}), \cdots, (v_n, \text{ add})$.

We will prove that for every possible output of running the algorithm on sequence 1) (there might be different outputs due to arbitrary choices the algorithm is allowed to do), running the algorithm on sequence 2) can yield the same tree. From this follows that the tree created after a single *remove* request can be created by a sequence of only *add* requests. Extending this claim by induction to any number of *remove* requests is straightforward.

Let $\sigma(v, S)$ denote the nodes of S which are closest to v. Let $P_a(v)$ and $P_b(v)$ denote the parent of v in the tree created by the end of sequence 1) and in the tree created by the end of sequence 2), respectively. Finally, let $P'_a(v)$ denote the parent of v in the tree created after processing only the n *add* requests of sequence 1) before v_i is removed. We shall now see that for every $1 \leq j \leq n$, where $j \neq i$, the following holds:

$$P_a(v_j) \in \sigma(v_j, \{s, v_1, v_2, \cdots, v_{j-1}\} - \{v_i\}). \quad (1)$$

Note that for each node v_j that joins the tree created by sequence 2), the algorithm is free to choose any node from $\sigma(v_j, \{s, v_1, v_2, \cdots, v_{j-1}\} - \{v_i\})$ as a parent. Thus, from (1) it follows that when running on sequence 2), the algorithm may choose as a parent the same node chosen for sequence 1). This would result in identical trees and it proves the lemma.

To prove (1), note that $P'_a(v_j) \in \sigma(v_j, \{s, v_1, v_2, \cdots, v_{j-1}\})$ must hold. If $P'_a(v_j) \neq v_i$, then $P'_a(v_j) \in \sigma(v_j, \{s, v_1, v_2, \cdots, v_{j-1}\} - \{v_i\})$. Since the *remove* request at the end of sequence 1) affects only v_i and its sons, $P_a(v_j) = P'_a(v_j)$ and (1) holds. In the other case, where $P'_a(v_j) = v_i$, the *remove* request requires v_j to select a new parent from $\sigma(v_j, \{s, v_1, v_2, \cdots, v_{j-1}\} - \{v_i\})$. Hence, (1) holds in this case as well. □

Theorem 1: Consider an execution of R-DGA on a sequence α of i add/remove requests. Let Z_i be the set of nodes in the multicast group following this sequence. Let $\text{R} - \text{DGA}(\alpha)$ the cost of the tree created by R-DGA and let OPT be the optimal tree spanning Z_i. Then

$$\frac{\text{R} - \text{DGA}(\alpha)}{\text{OPT}} \leq \lceil \log(|Z_i|) \rceil.$$

Proof: The theorem follows from [14], where this bound is proven for an execution of DGA on a sequence of only add requests, from Lemmas 2 and 3. □

It is interesting to note that although R-DGA can be viewed as a restricted version of DGA, applying to DGA the R-DGA approach for handling remove requests would not retain the logarithmic bound. Since in DGA a child is not necessarily a member of the multicast group, we will examine two natural approaches. The first approach is to reconnect the children of the removed node back to the tree in the same way as in R-DGA. The second approach is not necessarily to reconnect every immediate child of the removed node, but only the first descendants in every subtree of the removed node that is either a member of the multicast group or has degree three or more. The following example shows that both approaches do not retain the worst-case performance ratio of $\lceil \log(|Z_i|) \rceil$. Consider the graph depicted in Fig. 3(a). Suppose that all edges have a cost and delay of one except the long one between v and s whose cost and delay is two. Let s be the root of the multicast tree marked by the dark lines. Such a tree could have been created by adding the dark nodes from bottom to top. Suppose now that node x leaves the multicast group. In both approaches, node w must reconnect to the tree since it is a nonmember node of degree three. It has two possible shortest paths, and after the one that goes through x is chosen, the tree looks as depicted in Fig. 3(b). Next node z leaves the tree and node y joins it [Fig. 3(c)]. Following the sequence of add/remove requests described so far for the nodes in the topmost square, node x was removed from the tree without changing the long path connecting s to v. This process can be repeated in the other squares all the way down, resulting in the

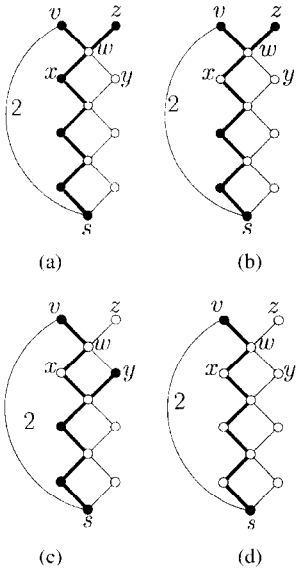


Fig. 3. Example of node removal in possible modifications of DGA.

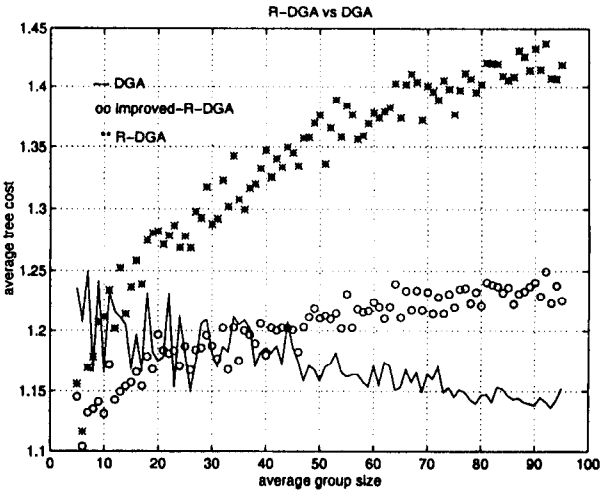


Fig. 4. Average cost of R-DGA versus DGA.

tree shown in Fig. 3(d). Since this example can be extended to any number of squares, and since the optimal tree connecting v to s is of cost two, the performance ratio is $O(N)$, where N is the number of add/remove requests. Since $|Z_i| = 1$ in the end, the performance ratio cannot be bounded by any function of $|Z_i|$.

B. Simulation Results

So far we have shown that the worst-case performance of R-DGA is at least as good as of DGA. In order to compare the average performance we have tested DGA, R-DGA, and a third algorithm, referred to as improved R-DGA, on 100 randomly generated networks. Improved R-DGA allows a wider selection for a node v, whose parent is deleted from the tree. Instead of sending v a list of the nodes that have joined the group in the last time before v and have not left since then, the source sends v (via v's old parent) the entire list of nodes in the tree except those in the subtree of v. Recall that the average number of sons of a deleted node is less than one, but if a deleted node has several sons N, say, then in order to avoid the creation of cycles the root arranges the sons in some arbitrary order $v_1, v_2, \cdots, v_N$ and allows v_i to connect to any node in the tree except those in the subtree of v_j for every $i < j \leq N$. Note that implementing the improved R-DGA would require the source to know the exact structure of its tree, namely, the identity of the parent of every destination node. R-DGA, in contrast, needs to know only the order according to which the destination nodes have joined the tree. This new requirement of the improved R-DGA leads to a small modification of the multicast protocol presented in Section III as follows. A node v that selects a parent, either because v has just joined the multicast group or because its previous parent has left the tree, needs to inform s about its new parent.

The simulated networks were created as in [6]. One hundred nodes were randomly distributed on a $[0 \cdots 400][0 \cdots 400]$ grid. An edge was then added between every pair of nodes u and v with probability $\beta \cdot \exp(-(d_{u,v}/576\alpha))$, where $d_{u,v}$ is the Euclidean distance between u and v. The values selected for α and β are 0.1 and 0.4, respectively. The network was then made connected by randomly selecting nodes from distinct components of the graph and connecting them with an edge. The cost and length of each edge are set to the Euclidean distance between the edge nodes.

The add/remove sequence of requests was created by adding a nonmember node with probability P_{in} and removing a member node with probability P_{out}. These values determine the average density of the multicast group. We tried densities ranging from 5% to 95% (a density of $p\%$ means an average of $p\%$ of the network nodes were members of the multicast group at any given time). The created trees were tested after sequences of 500 requests.

The metrics used for comparison is the competitiveness of each heuristic. Since calculating the optimal tree is NP-complete, we considered the cost of the tree generated by the static MPH as the optimal tree. Thus, for all of the following simulations, we define competitiveness of an algorithm A as the ratio between the cost of the tree constructed by A and the tree constructed by MPH.[4]

The simulation results are depicted in Fig. 4. The graph reflects 2400 executions on different networks and different group sizes. For each group size, the average performance of 25 executions is presented. The results show that the extra cost that R-DGA pays, due to the restriction of connecting new nodes only to nodes in the group or to the source, is more than desirable. For instance, when the density of the multicast group is larger than 30%, the competitiveness of R-DGA gets larger than 1.3, while DGA's competitiveness drops below 1.2. This is because the penalty for putting only group members in the junctions of the multicast trees becomes heavier as the group gets more dense. The improved R-DGA yields much better results, which are similar to those gained by DGA. When the density of the multicast group is smaller than 20%, the results of the improved R-DGA become distinctly

[4]Recall that MPH has a worst-case performance of two times the optimum [5]. However, simulations show [21] that its actual performance is only 5% worse than the optimum.

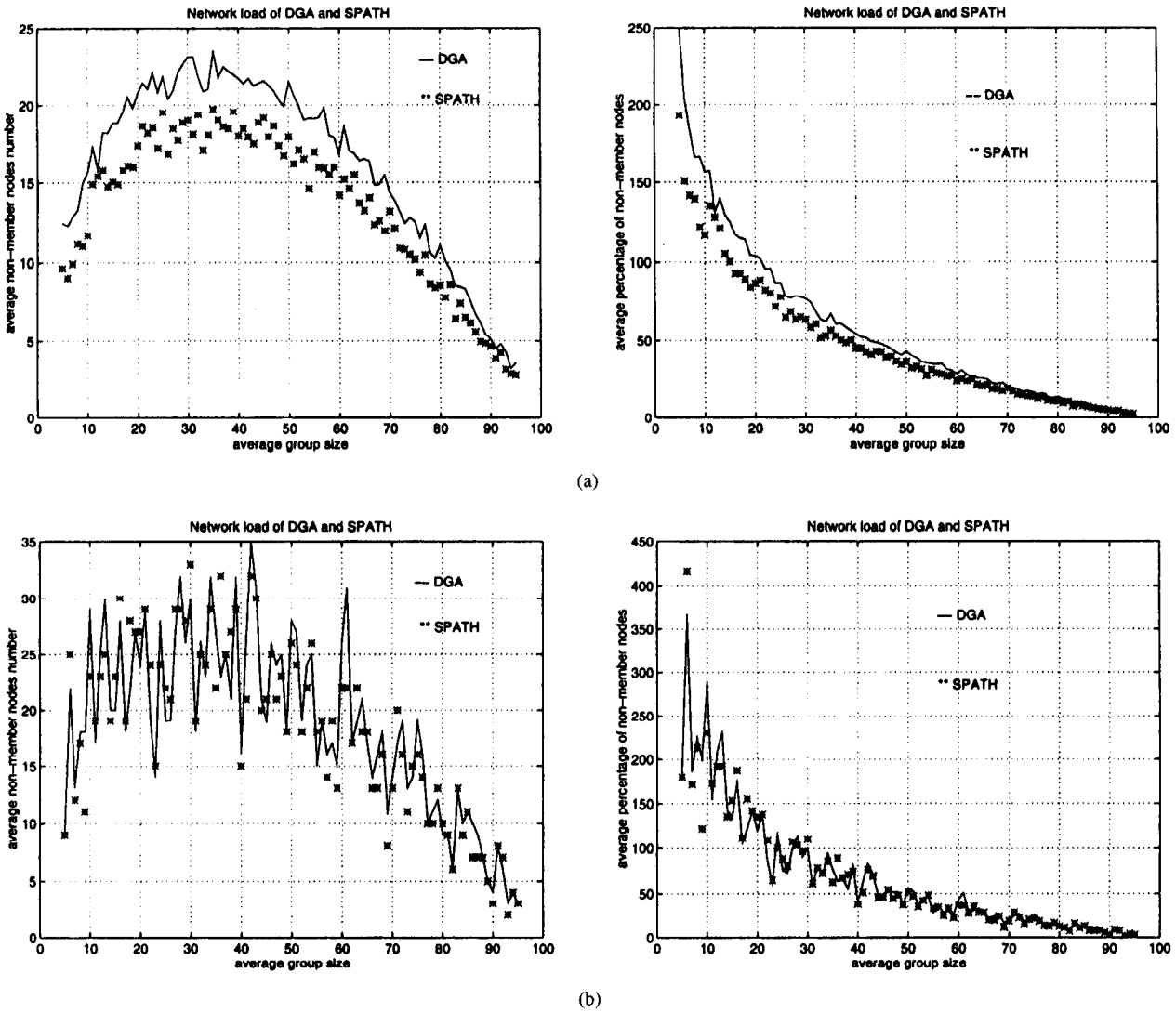


Fig. 5. Number of nonmember nodes involved in multicast routing in DGA and SPATH protocols (in R-DGA this number is always 0). (a) Average degree of network graph is eight. (b) Average degree of network graph is three.

better than those of DGA, whereas for larger groups it retains a reasonable competitiveness. The reduction in cost, even compared to DGA, is due to the smart mechanism employed by improved R-DGA when remove requests are received.

In order to view the advantage of R-DGA, Fig. 5 shows the number of nodes in the multicast tree that are *not* members of the multicast group, under DGA and SPATH, when the average degree of a node is eight [Fig. 5(a)] and three [Fig. 5(b)]. The left graphs show absolute numbers while the right ones show these numbers relative to the group size. These graphs show that for almost all sizes of multicast groups, the multicast tree created by SPATH or DGA contains 10–25 network nodes which are not members of the group. All of these nodes have to maintain and update state information in any case, and routing information in the case where the concept of tunneling is not employed, regarding multicast trees for which they serve as intermediate nodes only. Of course, when the network size increases, these numbers increase as well. For instance, for a 500-node network, an average tree with 40–60 destination nodes has about 60–70 nondestination nodes.

From Figs. 4 and 5, it follows that improved R-DGA is most effective when the destination group consists of no more than 10%–30% of the total network nodes. In such a case the cost of the improved R-DGA algorithm is *smaller* than the cost of DGA, and the number of nodes that need to encounter routing and maintenance burden is reduced by 50%.

Another potential scalability issue is the processing burden laid on the source, since it handles all add and remove requests. We believe that this will not pose any problem for the following reasons. Firstly, having a single node monitoring all additions and deletions from the tree may be required anyway for purposes such as admission control. Secondly, R-DGA is expected to be employed as an exterior multicast algorithm. Hence, each participating node is expected to be a border gateway rather than an end user. Consequently, the number of add/delete requests received by the source grows much more slowly than the number of end users. An add request issued by some end user will be forwarded by the border gateway to the source only if the border gateway does not yet belong to the tree. Similarly, a delete request issued by some end user

will be forwarded to the source only if no additional user in the same autonomous network belongs to the tree. Thirdly, when multiple sources are associated with the same multicast group, only one tree has to be established and only one source needs to maintain this tree. Finally, as previously discussed and is evident from Fig. 5, R-DGA significantly reduces the average processing burden imposed on the nodes. Hence, each node may be able to spend more processing power on those multicast trees for which it functions as a root.

V. Delay Constrained Multicast Trees

In the previous section we considered only the cost of the multicast tree and ignored the distance between the source and each destination node. However, for many future multicast applications, it will be desirable to bound the latency between the source and each member of the destination group. In what follows we analyze the problem of minimizing the cost of a multicast tree satisfying such a latency constraint.

The problem, referred to as *the constrained Steiner tree problem*, can be formulated as follows [16]. As before, the network is represented by a graph $G(V, E)$. Each edge e is associated with two values: its *cost* $c(e)$ and its *delay* $d(e)$. Delay constraint Γ is given. A tree T satisfies the delay constraint if the *delay* along each path from the root to a leaf is not more than Γ. The cost of the tree is, as before, the sum of the costs of the edges of the tree. The problem is given a source $s \in V$ and a destination group $Z \subseteq V$ to find the cheapest tree spanning $Z \cup \{s\}$ and satisfying the constraint.

In the following we give lower bounds for the dynamic and static cases. Then we present a dynamic algorithm achieving the lower bound in a restricted version of the problem, where the delay of an edge is proportional to its cost. Finally, we present simulation results for that algorithm.

As stated before, a static Steiner algorithm can achieve a worst-case performance ratio of two. For the constrained Steiner tree problem, we show that a polynomial algorithm achieving constant performance ratio is unlikely to exist, by providing a reduction from the minimum set cover (MSC) problem. It has been shown in [17] that no polynomial algorithm for MSC can achieve better than logarithmic approximation factor unless $n^{\text{polylog}(n)}$ contains NP, which is thought to be as unlikely as $P = \text{NP}$.

MSC can be formulated as follows. Given a set $V = \{v_1, \cdots, v_n\}$ and a set of subsets of V, $S = \{S_1, \cdots, S_m\}$, where $\forall i\ 1 \leq i \leq m$, $S_i \subseteq V$, find the smallest subset C of S such that every element of V occurs in at least one set of C. The Δ-approximated MSC is to find such a subset whose size is not more than Δ times the smallest subset. The Δ-approximated constrained Steiner tree problem is to find a solution to the constrained Steiner tree problem whose cost is within Δ times the optimal one.

Lemma 4: There is a polynomial reduction from 2Δ-approximated MSC to the Δ-approximated constrained Steiner tree problem.

Proof: Given V and S, we build a bipartite graph $G(V \cup \{s\}, S, E)$, where E is the set of edges defined as follows: $E = \{(v_i, S_j) \mid v_i \in s_j\} \cup \{(s, S_j) \mid 1 \leq j \leq m\}$. An example of such a graph is shown in Fig. 6. We will refer to the edges connected to s as the first-level edges and to the other edges as the second-level edges. Let the first-level edges have cost and delay of n, and the second-level edges have cost and delay of one. Using a constrained Steiner tree approximation algorithm, we get a tree T rooted in s and spanning V with delay constraint $n+1$. Since the graph is bipartite, every node in V has a parent in S. The delay constraint forces every node in S that is a member of the tree to be connected directly to s in T. Let C denote those nodes in S that are members of T. Since every node in V has a parent in C and since the graph was constructed so that a neighbor of a node v in V is a set that contains v, C covers all the nodes in V. Let $C(T)$ denote T's cost. Since T has exactly $|C|$ first-level edges and exactly n second-level edges, we get $C(T) = n|C| + n$, which yields $|C| \leq (C(T)/n) - 1$. If T' is the optimal constrained Steiner tree, then by definition of the approximation problem $C(T) \leq \Delta C(T')$, $|C| \leq \Delta(C(T')/n) - 1$ holds. From the optimal solution to the set cover problem C', a tree T'' can be built by connecting the members of C' to s via first-level edges and then connecting each node in V to the subset that contains it in C'. This yields a tree with cost $C(T'') = n|C'| + n$. Since T' is the optimal tree, $C(T') \leq C(T'')$ holds and thus we get $|C| \leq \Delta|C'| + \Delta - 1 \leq 2\Delta|C'|$. □

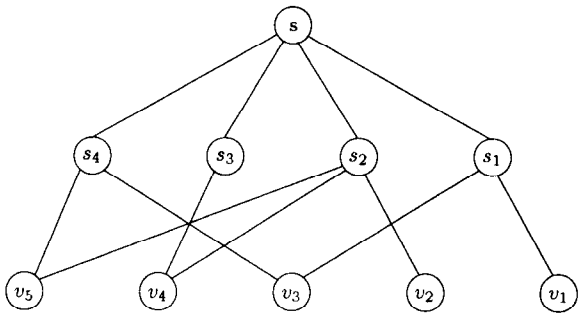


Fig. 6. An example of the reduction in the case where $V = \{v_1, v_2, v_3, v_4, v_5\}$, $S_1 = \{v_1, v_3\}$, $S_2 = \{v_2, v_4, v_5\}$, $S_3 = \{v_4\}$, $S_4 = \{v_3, v_5\}$, and $S = \{S_1, S_2, S_3, S_4\}$.

Note that in the graph built in the reduction, every edge's cost is the same as its delay; therefore, the lower bound holds for the restricted problem where the delay is proportional to the cost.

We now turn to the dynamic case. We show that no nonrearrangeable algorithm can achieve better than linear performance ratio.

Lemma 5: Any dynamic algorithm for the constrained Steiner tree problem that does not rearrange the tree after add requests has a worst-case ratio of $|Z|$.

Proof: We shall start by considering a graph for which the cost and the length of every edge are not necessarily equal. Let $G_n(V, U, E)$ be a bipartite graph defined as follows for any integer n larger than 0. $V = \{s, v_1, v_2, \cdots, v_k\}$, where $k = n^2$; $U = \{u_1, u_2, \cdots, u_t\}$, where $t = \binom{k}{n}$. An edge of cost one connects every node in U to s; an edge of negligible cost ϵ connects every node in U to exactly n nodes in V such that each node in U is connected to a different choice of n nodes from V (the nodes in V cover together all of the $\binom{k}{n}$ choices of n nodes from V). Suppose that all edges have a

delay of one and that the delay constraint is two. Every tree satisfying the constraint has a height of two levels at most. The first level is of nodes in U and the second level is of nodes in V. In any nonrearrangeable algorithm when a node $v \in V$ joins the tree, there are two possible cases. If v has a neighbor in U already in the tree, one ϵ-edge is added to the tree. Otherwise, a neighbor of v in U must be added along with its edge toward s, increasing the cost of the tree by $1+\epsilon$.

If the cost of the tree is $i < n$, it contains at most i nodes from U. In such a case there exist no more than $n * i$ nodes in V that have neighbors in the tree, and a node in V without such a neighbor can be found and added to the group, increasing the cost of the tree by $1+\epsilon$. Inductively, we can get a tree of cost n after adding no more than n nodes of V. The cost of this tree is n times the optimal cost, since there always exists a single node in U connected to these n nodes by ϵ-edges. This example can be extended to the case where the delay and the cost of each edge are equal by setting the cost of the edges between U and s to a high cost x (rather than one), the cost of the ϵ-edges to one, and the delay constraint to $x+1$. The performance ratio in this case is $(nx+n)/(x+n)$. For x much larger than n, this ratio tends to n. It can be then extended to the case where all edges costs are equal to one by replacing each edge with a higher cost c by a path of c edges and $c-1$ new nodes. □

To address the constrained Steiner tree problem, a version of R-DGA, referred to as C-R-DGA, is presented in the following. Like R-DGA, C-R-DGA aims at finding a low-cost tree rooted at s and spanning the destination group Z such that only nodes from $\{s\} \cup Z$ are located in the tree forks. In addition, C-R-DGA ensures that the path from s to any member of the multicast group is not longer than some threshold Δ.

C-R-DGA is very similar to R-DGA, except that when a new node searches for a parent, it takes into consideration not only the cost of the path leading to its parent but also the sum of the delay of this path and the delay from the source to the parent on the tree. More formally, let $\mathcal{C}(u \rightsquigarrow v)$ be the cost of the unicast path from u to v, let $\mathcal{D}(u \rightsquigarrow v)$ be the delay of this path, and let $\delta(u)$ be the delay from s to $u \in Z$ on the multicast tree. In addition, let Δ be the delay constraint. Node v selects node $u \in \{s\} \cup Z$ as a parent if the following holds:

1) $\mathcal{D}(u \rightsquigarrow v) + \delta(u) \leq \Delta$;
2) for no $u' \in Z$, where $u' \neq u$, $\mathcal{D}(u' \rightsquigarrow v) + \delta(u') \leq \Delta$ and $\mathcal{C}(u' \rightsquigarrow v) < \mathcal{C}(u \rightsquigarrow v)$ hold.

Assuming that the delay constraint Δ is not smaller than the delay of the shortest path from s to any network node,[5] node v can always find a parent because the first requirement is fulfilled by selecting the root s as a parent.

When the parent of v is deleted from Z, v has to select a new parent. This is performed according to 1) and 2) above, with two exceptions. Firstly, the new parent is selected by v from a *subset* $Z(v)$ of Z as in R-DGA (or in the improved R-DGA). Secondly, the path from s to v through the new parent must not be longer than the path from s to v through the old parent. Without this requirement, node v may have in its subtree some nodes in Z for which the delay constraint Δ was fulfilled with the old parent but not with the new one. Again, this stricter constraint can always be guaranteed by selecting the source s as the new parent.

Note that C-R-DGA requires every node v to know the value of $\delta(u)$ for every $u \in Z$ before selecting a new parent. This information can be provided to v by the source s when the latter responds to an add request of v or to a remove request of v's parent. To this end, every node joining the destination group must inform s of the identity of its parent, like in the improved R-DGA (see Section IV). In any case, C-R-DGA is currently more applicable for ATM networks than for IP networks. This is mainly because a multicast link can be associated in ATM with a maximum delay for the entire duration of the multicast session, as C-R-DGA requires.

In the following we study the performance of C-R-DGA. We first prove that C-R-DGA has the best possible worst-case performance ratio. Again, for the analysis we assume that the cost function is proportional to the delay function.

Lemma 6: C-R-DGA has a worst-case performance ratio of $|Z|$.

Proof: Let T_{OPT} be the cheapest tree satisfying the delay constraint and let $\mathcal{C}(v, T_{\mathrm{OPT}})$ be the cost of the path from the source s to $v \in Z$ in this tree. Let $T_{\text{C-R-DGA}}$ be the tree constructed by C-R-DGA and $P_{\text{C-R-DGA}}(v)$ be the parent of $v \in Z$ in this tree; thus, $\mathcal{C}(P_{\text{C-R-DGA}}(v) \rightsquigarrow v)$ is the cost of the path connecting v to $T_{\text{C-R-DGA}}$. In addition, $\mathcal{C}(s \rightsquigarrow v)$ is the cost of the shortest path from the multicast source s to v in the network graph.

First, note that $\mathcal{C}(P_{\text{C-R-DGA}}(v) \rightsquigarrow v) \leq \mathcal{C}(s \rightsquigarrow v)$, because C-R-DGA always gives v the option to choose s as its parent and to be connected to $T_{\text{C-R-DGA}}$ through $s \rightsquigarrow v$. Since the delay of $s \rightsquigarrow v$ in the network graph cannot be larger than the delay from s to v in T_{OPT}, and following the assumption that the cost function is proportional to the delay function, $\mathcal{C}(s \rightsquigarrow v) \leq \mathcal{C}(v, T_{\mathrm{OPT}})$. Consequently, $\mathcal{C}(P_{\text{C-R-DGA}}(v) \rightsquigarrow v) \leq \mathcal{C}(v, T_{\mathrm{OPT}})$ holds. Let $\mathcal{C}(T)$ be the cost of a tree T. Thus, $\mathcal{C}(T_{\text{C-R-DGA}}) = \sum_{v \in Z} \mathcal{C}(P_{\text{C-R-DGA}}(v) \rightsquigarrow v)$ holds, implying that $\mathcal{C}(T_{\text{C-R-DGA}}) \leq \sum_{v \in Z} \mathcal{C}(v, T_{\mathrm{OPT}})$. Since for every $v \in Z$, $\mathcal{C}(v, T_{\mathrm{OPT}}) \leq \mathcal{C}(T_{\mathrm{OPT}})$, then $\mathcal{C}(T_{\text{C-R-DGA}}) \leq |Z| \cdot \mathcal{C}(T_{\mathrm{OPT}})$ holds, and the lemma is correct. □

Theorem 2: No algorithm for finding the cheapest tree with delay constraint that does not rearrange the tree after add requests has a worst-case performance ratio better than C-R-DGA.

Proof: Directly from Lemmas 6 and 5. □

We conclude this section with simulation results for C-R-DGA on the network model described in the previous section. We tried two values of constraint. The first is equal to the longest distance of a node to the root, namely, the tightest possible constraint, and the second is 1.5 times larger than that distance. Each of these two values was tested on networks and group sizes similar to those in the previous section. The resulting graphs are shown in Fig. 7. In all cases the performance of C-R-DGA is compared to the performance of a simple SPATH, where the multicast tree consists of a collection of the shortest paths from the source to each member

[5] If for some v $\mathcal{D}(s \rightsquigarrow v) > \Delta$, no algorithm can create a tree that spans v and satisfies the constraint.

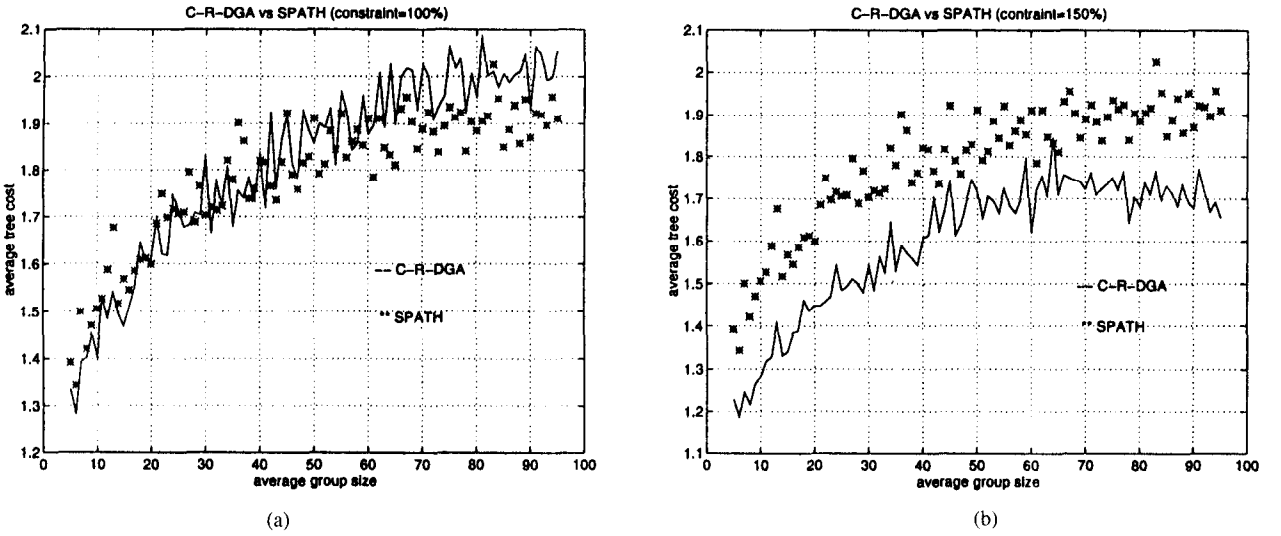


Fig. 7. Average cost of C-R-DGA versus the SPATH.

of the destination group. We use this SPATH as a benchmark because the tree it generates, which is actually the same tree generated by the MBONE DVMRP [9], obviously fulfills the delay constraint, and because this algorithm has the same worst-case performance ratio of $|z|$ (see [5]).

The graphs in Fig. 7 show that when the delay is tight, the performance of C-R-DGA and the SPATH are similar. These results show that we can apply the restriction that removes the routing and maintenance burden from all of the nodes in the tree that are not in $\{s\} \cup Z$, while imposing a tight upper bound on the delays without increasing the cost of the tree at all. For the 150% constraint, the performance of C-R-DGA is even better than that of the SPATH. This shows that C-R-DGA can trade off the cost of the tree and the maximum delay.

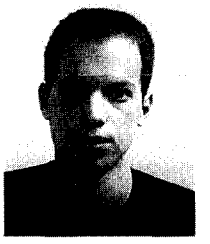
VI. Conclusion

The paper has presented an approach for solving the scalability problem of routing over, updating, and maintaining a great number of multicast trees in a datagram network. According to the presented approach, only the nodes directly related to a multicast tree, namely, the source node and destination group nodes, need to keep information related to the tree routing and maintenance. This is achieved by viewing a multicast tree as a collection of multicast paths and imposing a restriction where only the source node and destination nodes can be located in the tree junctions as end points of multicast links. Based on this restriction, two algorithms were presented: R-DGA and C-R-DGA. R-DGA aims at establishing a low-cost tree, whereas C-R-DGA aims at establishing such a tree while imposing a constraint on the distance from s to every node in the destination group. The paper has shown that both algorithms can be efficiently implemented by a multicast protocol that can guarantee fast response to changes in the destination group. The paper has also shown that despite the restriction imposed by the proposed approach, both R-DGA and C-R-DGA yield a good performance. The worst-case performance of C-R-DGA is the best that any nonrearrangeable algorithm may yield, whereas the worst-case performance of R-DGA is within two times of the best that any nonrearrangeable algorithm may yield. In terms of their average performance, both algorithms perform as well as (and, in many cases, even better than) other known and applicable algorithms that do not impose the restriction and, therefore, require all of the nodes along the data path of a tree to participate in the routing over the tree or in the maintenance of the tree.

References

[1] M. H. Ammar, S. Y. Cheung, and C. M. Scoglio, "Routing multipoint connections using virtual paths in an ATM network," in *Proc. INFOCOM*, 1993, pp. 98–105.
[2] T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees (CBT)," in *Proc. SIGCOMM*, San Francisco, CA, 1993, pp. 85–95.
[3] F. Bauer and A. Varma, Distributed algorithms for multicast path setup in data networks," in *Proc. GLOBECOM*, Singapore, Nov. 1995, pp. 1374–1378.
[4] F. Bauer and A. Varma, "ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm," in *Proc. INFOCOM*, Mar. 1996, pp. 361–368.
[5] K. Bharath-Kumar and J. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Trans. Commun.*, vol. COM-31, pp. 343–351, Mar. 1983.
[6] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Commun.*, vol. COM-34, pp. 677–691, May 1986.
[7] Y. Dalal and R. Metcalfe, "Reverse path forwarding of broadcast packets," *Commun. ACM*, vol. 21, no. 12, pp. 1040–1048, 1978.
[8] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, "Protocol independent multicast (PIM): Motivation and architecture," Internet draft, Jan. 1995.
[9] S. Deering, C. Partridge, and D. Waitzman, "Distance vector multicast routing protocol," vol. RFC-1075, Nov. 1988.
[10] M. Doar and I. Leslie, "How bad is naive multicast routing?," in *Proc. INFOCOM*, San Francisco, CA, 1993, pp. 82–89.
[11] H. Eriksson, "MBone: The multicast backbone," *Commun. ACM*, vol. 37, no. 8, pp. 54–60, Aug. 1994.
[12] M. Garey and D. Johnson, *Computers and Intractability*. San Francisco, CA: Freeman, 1979.
[13] E. Gilbert and H. Pollak, "Steiner minimal tree," *SIAM J. App. Math.*, vol. 16, pp. 1–29, 1968.
[14] M. Imase and B. Waxman, "Dynamic Steiner tree problem," *SIAM J. Discrete Math.*, vol. 4, no. 3, pp. 369–384, Aug. 1991.
[15] J. Kadirire and G. Knight, "Comparison of dynamic multicast routing algorithms for wide-area packet switched networks," in *Proc. INFOCOM*, Apr. 1995, pp. 212–219.

[16] V. Kompella, J. Pasquale, and G. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Trans. Networking*, vol. 1, pp. 286–292, June 1993.
[17] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," in *Proc. 25th ACM Annual Symp. Theory Computing*, San Diego, CA, 1993, pp. 286–293.
[18] J. Moy, "Multicast routing extensions for OSPF," *Commun. ACM*, vol. 37, no. 8, pp. 61–66, Aug. 1994.
[19] W. Simpson, "IP in IP tunneling," vol. RFC-1853, Oct. 1995.
[20] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," *Math. Japonica*, vol. 24, pp. 573–577, 1980.
[21] B. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, 1988.
[22] P. Winter, "Steiner problem in networks: A survey," *Networks*, vol. 17, pp. 129–167, 1987.



Ehud Aharoni received the B.Sc. and M.Sc. degrees in computer science from the Technion, Israel Institute of Technology, Haifa, Israel, in 1995 and 1997, respectively. Since then he has been developing products for the Internet.

Reuven Cohen (M'92), for photograph and biography, see p. 29 of the February 1998 issue of this TRANSACTIONS.