Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Distributed ADH Heuristics for Multicast Tree Construction

Luca Gatani, Giuseppe Lo Re

RT-ICAR-PA-03-02 **settembre 2003**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Distributed ADH Heuristics for Multicast Tree Construction

Luca Gatani[2], Giuseppe Lo Re[1]

Rapporto Tecnico N.2: **RT-ICAR-PA-03-02**

Data: **settembre 2003**

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo, Viale delle Scienze edificio 11 90128 Palermo
[2] Università degli Studi di Palermo, DINFO, Viale delle Scienze 90128 Palermo

Distributed ADH Heuristics for Multicast Tree Construction

L. Gatani, G. Lo Re

September 2003

Contents

Abstract

This paper proposes two distributed algorithms for the heuristic solution of the Steiner Tree Problem in Networks (SPN). The problem has a practical application in the construction of a minimum cost distribution tree for multicast transmission. Multicast transmission represents a necessary lower network service for the wide diffusion of new multimedia network applications. Currently, given the lack of efficient distributed methods, the existing protocols build the multicast distribution tree using some selected central node. The proposed distributed algorithms allow the construction of effective distribution trees using a coordination protocol among the network nodes. The algorithms have been implemented and tested both in simulation and on experimental active networks, and their performance values are presented.

1 Introduction

Several future multimedia networking applications such as distance education, remote collaboration, video-on-demand services and teleconferencing will become widespread, relying on the ability of the network to provide multicast services effectively and efficiently. Multicasting refers to the simultaneous transmission of data to multiple destinations, and can be seen as the generalized concept of one-to-one unicasting and one-to-all broadcasting.

Trees isolated over the network topology are adopted by multicast routing for data transmission, in order to achieve a resource usage minimization by the simultaneous sharing of links when transmitting from one source to many destinations. In this context, an underlying specific multicast routing algorithm should determine, with respect to certain optimization objectives, a multicast tree connecting source (or sources) and group members. Data belonging to the source flow will reach their destinations, traversing tree edges once only and being replicated at branching points. One of the main goals of multicast routing is to minimize the overall tree cost. Tree cost refers to the amount of network resources needed to transport packets over the tree, and, consequently, minimizing tree cost is equivalent to the efficient use of network resources. Determining the optimal (i.e., minimum cost) multicast tree connecting all the members of a group is a difficult problem: it can be modeled as the Steiner problem in networks (SPN) which has been proved to be NP-complete in its decisional version [9]. The NP-complete nature of the problem means that the computation of explicit solutions in large networks is prohibitively expensive. For example, two popular explicit algorithms, the Spanning Tree Enumeration Algorithm (STEA) and the Dynamic Programming Algorithm (DPA), present time complexities of $O(p^2 2^{(n-p)} + n^3)$ and $O(n3^p + n^2 2^p + n^3)$, respectively, where n is the number of nodes in the network and p is the number of multicast members [25]. Excellent, inexpensive centralized heuristics for approximate Steiner trees have been proposed in the literature [11], [25], [20], [19]. Most produce solutions whose cost has been analytically demonstrated to be less than twice the cost of the optimal

solution. However, experimental observations indicate that in most cases they are capable of singling out near optimal solutions with reasonable speed.

Most of the proposed heuristics are centralized in nature. In the centralized approach, a central node that is aware of the state of the whole network computes the tree. The computation is generally easy and fast. However, the overhead of maintaining, in a single node, coherent information about the state of the entire network can be prohibitive. As a consequence, centralized algorithms are not practical for large networks where complete knowledge of their state is difficult to collect. In a distributed approach, on the other hand, each node of the network actively contributes to the algorithm computation. Distributed routing algorithms become indispensable when the network nodes cannot have a complete knowledge of the topology and state of the network. In this case, a distributed route computation would presumably take place only on those nodes owning the resources, and the resulting decisions would become effective immediately at those nodes. A distributed algorithm is a collection of asynchronous, independent algorithms that run in each node of the network. A multicast routing distributed algorithm establishes multicast connections in a decentralized manner, by exchanging messages among the nodes involved, which in turn carry out specified portions of the algorithm. A distributed approach can be slower and more complex than the centralized one, but it does not need to maintain the entire network state in each node.

In this paper we introduce two distributed algorithms as an alternative to those proposed by Bauer and Varma in [3]. The proposed algorithms are more effective than both Bauer and Varma's and previous algorithms in terms of the quality of the computed solution, and they also yield good average case behavior in terms of running time and number of messages exchanged.

The remainder of this paper is structured as follows. After a review of some previously presented distributed algorithms for the Steiner tree problem in Section 1.1, Section 2 discusses the importance of the efficient utilization of network resources, as well as the frequently considered optimization criteria. A network model is also briefly introduced. Section 3 deals with the Steiner tree problem and with its application to communication networks. This section also provides a short description of the centralized algorithms on which the proposed distributed algorithms are based. Two new distributed algorithms are presented and analyzed in Section 4, and the results of performance measurements are given in Section 5. The Appendix contains the formal specification of the proposed algorithms according to I/O automaton formalism.

1.1 Background

Distributed algorithms for the Steiner tree problem in networks (SPN) have received very little attention over the last few years. Most of the versions proposed in the literature [10], [5] are based on reducing the SPN to the minimum spanning tree (MST) problem, and on using a distributed MST algorithm such the one described by Gallager *et al.* [6]. The Steiner tree is then created by pruning unnecessary leaves and branches from the resulting minimum spanning tree.

However, distributed Steiner heuristics based on a pruned MST algorithm suffer from a significant drawback: the theoretical upper bound on their competitiveness (the ratio of the costs of the trees achieved, respectively, by the heuristic algorithm and by an optimal algorithm) has been shown to be equivalent to $s + 1$ [22] (where s is the number of non-multicast nodes), which means that competitiveness decreases with the size of the multicast group. On the other hand, the equivalent upper bound for several well known centralized heuristics (such as DNH, SPH, K-SPH, ADH) is $2(1 - (1/p))$ [25] (where p is the number of multicast nodes). Distributed Steiner algorithms based on MST computation present a further shortcoming, since they require all the nodes in the network must participate in their execution.

Bauer and Varma [3] presented two distributed algorithms for the Steiner problem in networks based on the centralized heuristics SPH and K-SPH. The latter has a similar structure to the algorithm in [6], but, since the multicast group is typically a subset of all network nodes, the spanning tree achieved will only cover a portion of the network. This distributed algorithm involves the behavior described below. Initially, each multicast node starts out as a fragment, and each fragment attempts to merge with its closest neighboring fragment. The closest fragment is located in two steps: a discovery step and a connection step. In the discovery step, the fragment leader sends a flood message to the other nodes in the fragment. In order to locate nearby fragments, each node of the fragment sends query messages to neighboring non-fragment nodes. The fragments discovered are notified to the leader, which in turn determines its preferred closest fragment. In the connection phase, the fragment leader sends a request message to its closest preferred fragment G. If G is in the discovery phase, it sends a busy response back and F will send out the request message again. If F is not the preferred fragment of G, then G sends a reject message. Otherwise, if both F and G are the preferred fragments of each another, they merge to form a larger fragment. The process continues until a single fragment remains in the network. The distributed SPH is a special case of distributed K-SPH, even though it is a serial algorithm in nature. In its execution, only one fragment grows, connecting members to itself until all the multicast members are part of the same fragment. In this way, it concentrates much of the work at the source, whilst distributed K-SPH allows multiple fragments of the tree to combine in parallel. This allows distributed K-SPH to provide lower convergence times without substantially increasing the number of messages. However, as reported in [3], the convergence time for distributed K-SPH is significantly higher than those of the pruned MST algorithm. Singh and Vellanky [18] proposed a modified version of distributed K-SPH that adopts some ideas from the algorithm in [6] to make the fragment combination mechanism more efficient than that in the protocol in [3]. In particular, the method suggested tries to avoid sending reject messages as much as possible in order to avoid new discovery phases. As opposed to the "waving technique" adopted in [3] and [18], Novak *et al.* [14] presented a distributed table-passing algorithm for SPN which is based on the centralized heuristic SPH.

It is important to notice that all existing distributed algorithms suffer draw-

backs such as heavy communication costs, long connection setup times, and poorer quality of the solutions produced as compared with centralized heuristics. This paper introduces two distributed Steiner algorithms based on the strategy adopted by the Average Distance Heuristic (ADH) which are shown to yield good performance in terms of the quality of the computed solution, the running time, and the number of messages exchanged.

2 Multicast routing and network model

Multicast routing algorithms can be classified according to three main categories:

- shortest path-based algorithms,
- Steiner-based algorithms,
- and constrained Steiner-based algorithms.

Shortest path-based algorithms are suitable for those applications, such as video conferences, which require short propagation delays between the source and each of the receivers [8]. Steiner-based algorithms are well suited to those services, such as Video on Demand, which need to consume as few network resources as possible. Constrained Steiner-based algorithms combine both types of requirements in a single multicast communication. Steiner-based algorithms solve more difficult problems (NP-complete) than shortest path-based algorithms. Furthermore, constrained Steiner-based algorithms are usually derived by introducing additional constraints into basic Steiner algorithms. Research into efficient Steiner-based algorithms is of great interest at the moment. Steiner-based algorithms are becoming very important in mobile wireless networks, where bandwidth is limited, not to say scarce. Multicast routing in wide area WDM networks is also Steiner based, as the costs of wavelength conversion at nodes and of using wavelengths on links have to be considered.

Some basic terminology and assumptions are introduced here. Given an undirected graph $G = (V, E)$, with V the node set and E the edge set, let Z be the subset of multicast nodes, and $S = V - Z$ the set of non-terminal nodes. Let $n := |V|$, $m := |E|$, $p := |Z|$, whilst $s := |S|$. $P_{i,j}$ is the shortest path between nodes i and j, $d_{i,j}$ is the distance from node i to node j, which means the cost of the shortest path between them, and $c(T)$ is the cost of the tree T (the sum of T's edge weights). The distance between a node and a tree is the cost of the shortest among all paths between that node and any node in the tree. Likewise, the distance between two trees is the distance of the shortest among all paths between any node in one tree and any node in the other tree. We assume that each node i has a routing table that provides the shortest distance $d_{i,j}$ to any other node j. In addition, for each destination j, it also stores the next hop in the path from i to j. Using this information, a node is able to send messages via the shortest path (as indicated by the routing tables) to any destination.

We assume that this information is provided by an underlying network layer protocol such as a distance vector protocol (e.g., RIP [8]).

3 The Steiner Tree Problem

The formal definition of the (minimum) Steiner tree problem in networks can be stated as follows:
Definition.

Let $G = (V, E)$ be an undirected connected graph of the communication network, where V is the node set and E the connection set, with positive weights being associated with the connections. In this graph we consider a set $Z \subseteq V$ of destination nodes, called the multicast group. The Steiner tree problem is defined as finding a minimum cost sub-graph of G, such that there exists a path in the sub-graph between every pair of destination nodes.

Since the edge weights are positive, a solution involves isolating a subset S' disjoint of Z, which provides an optimal tree linking all nodes in Z. The non-destination nodes in S' are called Steiner nodes.

As the computation of an optimal solution of the Steiner tree problem is NP-complete [9] and thus not suitable for real-time applications, multicast routing algorithms are based on heuristic methods, some of which have been found to perform well [23]. The effectiveness of these heuristics can be measured in terms of the ratio between the cost of the solutions they are able to identify, and the cost of the optimal solution. This measure is known as cost competitiveness. Only a subset of Steiner tree heuristics have the properties that make them suitable for distributed implementation in real networks, where nodes have limited routing information. That is, to be suitable for distributed implementation, heuristic methods have to satisfy four criteria. They must:

1. use the existing routing information available at each node in the network, as provided by underlying unicast protocols;
2. use minimal computational and network resources;
3. require minimum coordination between nodes in the networks;
4. require a limited amount of computation by the non-member nodes.

An exhaustive overview of SPN centralized heuristics can be found in [25]. Among the centralized path-distance heuristics, the following four heuristics, in our view, seem to represent the best candidates for distributed implementation: the Shortest Path Heuristic (SPH)[20], the Kruskal-based Shortest Path Heuristic (K-SPH)[2], the repetitive Shortest Path Heuristic (SPH-Z) [25], and the Average Distance Heuristic (ADH) [17]. A short summary of heuristics K-SPH and ADH is given below. Another attractive heuristic, the Average Distance Heuristic with Full connection (ADHF), is fully described because it is not well known.

K-SPH: The Kruskal-based Shortest Path Heuristic starts with the forest F of multicast group member nodes. It repeatedly joins the two closest multicast group member sub-trees in F until a single tree spanning all multicast group members remains. The algorithm is described as follows:
1. Begin with the collection F (forest) of single vertex trees consisting of the p Z-vertices.
2. Connect the two closest sub-trees, T_1 and T_2, in F by their shortest path forming a new tree T'. Let $F := (F - \{T_1, T_2\}) \cup \{T'\}$.
3. If F contains at least two trees, then go to step 2; otherwise the single tree in F is the solution T_{K-SPH}.
The competitiveness bound of the K-SPH heuristic is also 2. Its run-time bound, $O(z{\cdot}n^2)$, is dominated by shortest path calculations between multicast members.

ADH: The minimum Average Distance Heuristic is a generalization of K-SPH. Like K-SPH, ADH starts with the forest of multicast group member nodes. It repeatedly finds the most central node, v^*, to the current set of sub-trees, v^* being defined as the node with the shortest average distance to a subset of neighboring sub-trees. The closest and second closest sub-trees to v^*, along with all edges and nodes on the shortest paths from v^* to these sub-trees, including v^*, are then combined to form a new sub-tree. The process continues until all nodes in Z are contained in a single sub-tree. The centrality measure of a node is computed as a weighted average of the distances of that node to all sub-trees sorted in a non-decreasing order.
1. Begin with the collection F (forest) of single vertex sub-trees consisting of the p Z-vertices.
2. For every vertex $v \in V$, re-label the sub-trees in the current forest, $F = \{T_1, ..., T_k\}$, such that they are in non-decreasing order of their distance from v (i.e., $d(v, T_1) \leq d(v, T_2) \leq ... \leq d(v, T_k)$) and for each r, $2 \leq r \leq k$, compute the mean distance

$$\mu(v, r) := \frac{\sum_{j=1}^{r} d(v, T_j)}{r - 1}. \tag{1}$$

Define

$$f(v) := min\{\mu(v, r) \mid 2 \leq r \leq k\} \tag{2}$$

and choose v^* minimizing $f(v)$.
3. Join the corresponding sub-trees T_1 and T_2 closest to v^* by a shortest path through v^*, forming a new sub-tree T'. Let $F := (F - \{T_1, T_2\}) \cup \{T'\}$.
4. If F contains at least two sub-trees, then go to Step 2; otherwise the single tree in F is the solution T_{ADH}.

Fortunately, as one can easily observe, in Step 2 $f(v)$ needs not to be evaluated in full for each node, taking in account only a limited range of values of r. Furthermore, some simple additional rules can be considered for resolving ties (i.e., cases when there are two or more nodes minimizing f).

The worst case time complexity of ADH is $O(n^3)$ and its competitiveness is bounded from above by $2 - 2/p$ and can tend to 2, as shown by Waxman and Imase [24]. However, based on a simulation performed by Rayward-Smith, the cost of the approximation trees produced by ADH exceeds the cost of the SPN optimal solution by less than 5%, on average [16].

ADHF: The idea of investigating versions of ADH which allow connection of multiple (and not only two) sub-trees of F at each iteration was suggested by Winter in [25]. Bern and Plasmann [4] proposed such a version of ADH in 1989 and demonstrated that it is a 4/3-approximation algorithm for the Steiner problem on full connected graphs with edge lengths 1 and 2. The proposed heuristic, known as the minimum Average Distance Heuristic with Full connection, differs from ADH only in step 3, which is replaced by:
3'. Choose $r^*, s.t.\ 2 \leq r^* \leq k$, minimizing $\mu(v^*, r)$. Join (successively) each of the r^* closest sub-trees $T_1, T_2, ..., T_{r^*}$ to v^* by a shortest path to v^* forming a new sub-tree T'. Let $F := (F - \{T_1, T_2, ..., T_{r^*}\}) \cup \{T'\}$.

The time complexity of ADHF is still $O(n^3)$. As for several other heuristics, ADHF's competitiveness is bounded by $2 - 2/p$ and, for any $\varepsilon > 0$, there is an instance of the Steiner problem such that competitiveness is greater than $2 - \varepsilon$. Notice that step 3' is less cautious than step 3. Nevertheless, Bern and Plasmann erroneously assumed that ADHF and ADH should necessarily provide solutions of the same cost. This is not true in general and Plesnik [15] showed with a counterexample that ADH can win over ADHF. However, few previous computational results indicate that the greedy approach adopted by ADHF does not significantly affect the quality of the solutions, although ADHF often requires fewer iterations than ADH.

Bauer e Varma [3] proposed a distributed implementation of SPH and K-SPH. Heuristic SPH-Z fails our criterion, because it requires as many as p copies of component information to be stored at each node and a great number of network messages to be passed before convergence. The ADH heuristic, on the other hand, looks very attractive because of its good competitiveness values (in the literature, it is reportedly the best performing heuristic among the single step ones). In this paper we propose two distributed algorithms respectively based on the ADH heuristic and on its version with full connection (ADHF). We also present a simple additional technique limiting the execution of these algorithms only to a subset of the nodes in the network.

4 Proposed heuristics

The proposed algorithms are designed as a set of cooperative, asynchronous, independent processes running one for each node in the network. We assume that:

1. the network is connected;

2. each node in the network is a router;

3. each node knows its shortest path (i.e., distance and first hop) to all other nodes in the network, via the routing table computed by an underlying unicast protocol;

4. no topology changes occur during the execution of the algorithm;

5. every node has a unique identifier (UID).

4.1 Distributed ADH

As in its centralized version, distributed ADH (D-ADH) starts with a forest of multicast members (Z-nodes) and connects them into successively larger sub-trees until a single multicast tree has been set. We refer to the sub-trees as *fragments*. During algorithm execution, each node in the network is either part of a fragment or has not yet been included in the multicast tree. It should be noticed that every Z-node is always a fragment node and every non-member node is initially a non-fragment node. When two or more fragments merge, the nodes in these fragments and those lying on the interconnecting paths become the new merged fragment nodes. In order to uniquely identify fragments, each has a fragment leader and is identified by the same index (UID) as the leader. Initially each multicast member is the leader of its own one-node fragment. When two fragments merge, the node which starts the merging process assumes the leadership (see below). Distributed ADH processes running on the network nodes exploit the shortest path information, which is available on local nodes (i.e., the routing information computed by an underlying unicast protocol), as well as information about the multicast forest exchanged via messages. The algorithm is structured in rounds. At every round, one node in the network acts as a root node (initially, the root node is the one which starts the multicast tree setup). The root node broadcasts some information about the multicast forest (for example, the set of node-fragment indexes pairs) over a spanning tree (set in a distributed manner, during the first round). Having received this information, each node can calculate its proximity measure to the external fragments according to equations 1 and 2. Next, the minimum value of f is reported towards the root node, starting from the leaves of the spanning tree and using a converge-cast process. Eventual ties are opportunely resolved. The root node receives the information about the best value of f and sends a message asking the detected most central node to connect its closest fragments. If the most central node belongs to a fragment, it connects just one fragment (the closest external fragment). If the most central node does not belong to any fragment, it connects (via itself) the two closest fragments. In both cases, fragments are joined by minimum cost paths and the state of the nodes on these paths is modified appropriately (the intermediate nodes become Steiner nodes). Furthermore, the most central node updates the information about the forest, setting itself as the leader of the new merged fragment. When the connection step is completed, the most central node checks to see whether the forest is

connected (i.e., if all the nodes in the forest belong to the same fragment). If this is the case, the algorithm terminates and the fragment thus achieved is the resulting multicast tree, otherwise the most central node becomes the new root node and starts a new round of the algorithm. It should be noticed that each round reduces by one unit the number of fragments in the forest. Hence, after $p-1$ iterations, there is only one tree spanning all Z-vertices.
The algorithm can be described as follows:
1. *Initialization*
A node receives the list of multicast members UIDs from an external user. It becomes the root node for the first round and builds a data structure representing the multicast forest (e.g., a list of pairs, node - index of the fragment to which the node belongs), initially formed only by the Z-nodes.
2. *Construction of a spanning tree*
The root node starts the construction (via a distributed algorithm) of a network spanning tree. Each node in this tree stores a reference to every node directly attached in the tree.
3. *Broadcasting along the spanning tree*
Using the spanning tree, the root node sends in broadcast information about the multicast forest.
4. *Computing function f*
Using the information received from the root node and the locally available unicast routing table, each node calculates the f function according to equations 1 and 2. This step is achieved as follows:

a. a node which already belongs to a fragment, determines the closest external fragment (the information needed includes the fragment identifier, distance, and the node representing the tail of the minimum cost path in the selected fragment);

b. a node external to any fragment calculates its own f value. It takes into account candidate fragments in a non-descending order to determine the value of f, as well as the necessary information about the selected fragments (the identifier and node that represents the tail of the minimum cost path in each fragment).

5. *Convergecasting of the minimum value of f*
Using a converge-cast process along the set spanning tree, the information about the computed minimum value of f is reported towards the root node. Ties are opportunely resolved.
6. *Election of the most central node*
When the root node receives the information from its children on the spanning tree, it determines the best value of f, and sends a notification message to the node v^* that computed this value.
7. *Merging of target fragments*
After the notification, the node v^* becomes the most central node for the current round and starts the merging process. This is carried out in the following way:

a. if v^* belongs to a fragment, it connects to itself the closest external fragment, via the minimum cost path;

b. if v^* does not belong to any fragment, it connects the two closest fragments, via the minimum cost paths.

During the merging process, the state of the nodes along the connecting paths and the information about the multicast forest are opportunely updated.
8. *Election of the new root node*
If all Z-nodes are in the same fragment (i.e., the forest is already connected), the algorithm terminates; if not, the node v^* becomes the new root node and starts a new round (go to step 3).

4.2 Distributed ADHF

Distributed ADH is to some extent a serial algorithm since there are only two fragments connecting together. However, this makes it attractive, because it is relatively simple and does not require much coordination between the nodes in the network. Moreover, compared to centralized ADH, the distributed version has a shorter running time since the function f is computed in parallel by all the network nodes. Furthermore, in order to improve the convergence time and the number of transmitted messages, we propose a variant of the distributed ADH, which is based on the centralized heuristic ADHF. The idea underlying this further version of the algorithm, distributed ADHF (D-ADHF), is that at each round, if the most central node does not belong to any fragment, instead of connecting only the two closest fragments, it connects all the r^* closest fragments, where r^* is the number of addends in the sum of equation 1 that gives the best value of $\mu(v, r)$. This version differs from the previous one only in step 7 which is here replaced by:
7'. *Merging of target fragments (with full connection)*
After notification, the node v^* becomes the most central node for the current round and starts the merging process. This is carried out as follows:

a. if v^* belongs to a fragment, it connects to itself the closest external fragment, via the minimum cost path;

b. if v^* does not belong to any fragment, it connects the r^* closest fragments, via the minimum cost paths; r^* is the index which minimizes $\mu(v^*, r)$, according to equation 1.

During the merging process, the state of the nodes along the connecting paths and the information about the multicast forest are opportunely updated.

4.3 Reduced node set technique

The distributed heuristics ADH and ADHF are capable of isolating efficient suboptimal multicast trees using only a minimal amount of computational and

network resources. Namely, the design guideline was aimed to limit the computational effort for the non-multicast nodes. However, inefficiencies may result when multicast nodes represent just a small portion of the network nodes, since the algorithms do not entail only those nodes directly involved in the multicast session, but they require an active participation of all network nodes. In order to cope with this drawback, we introduce a further version, enhancing both the proposed algorithms, and reducing the algorithm execution to a subset of network nodes. The underlying idea is that spanning trees generally do not need to reach all network nodes but only those network areas where multicast nodes are located. Furthermore, it is a common case that in a multicast session all the nodes involved are located only in one area (or few areas). For example, let us consider a videoconference session. The conference is most likely to directly involve a minority of nodes in the network, probably located in a limited geographical area. In such cases, the adoption of the proposed technique makes more practical the use of the heuristics D-ADH and D-ADHF. The technique involves that in the initial phase of the first round, the root node (i.e., the multicast node that starts the tree construction) computes a dispersion measure of the multicast group nodes around itself. Then, the root node builds a tree, which will span the network area where multicast nodes are located. The tree, which is built, is made persistent through the whole execution. A reasonable choose is to compute the maximum distance between the root node and any other multicast node, and to construct a shortest path tree with source in the root node and depth equal to (or, slight greater than) the maximum distance computed. The broadcasting and converge-casting processes, and also the computation of f, are thus limited to the nodes in the tree. In such a way, only the multicast nodes and the nodes within their neighborhood are required to participate to the algorithm execution, without significantly affecting the competitiveness of the produced solutions.

This additional technique introduces the following modification in respect to the base versions D-ADH and D-ADHF. Step 2 is replaced by:
2'. *Construction of a tree which spans only the multicast nodes*
Using the locally available unicast routing information, the root node computes the maximum distance between itself and any other multicast nodes. The root node starts the construction (via a distributed algorithm) of a shortest path tree with source in the root node and depth equal to c times the maximum distance computed (where c is a parameter whose value is equal to, or slightly higher than, 1). Each node in this tree stores a reference to every node directly attached in the tree.

4.4 Complexity Analysis

Having described the distributed ADH and ADHF in the previous subsections, we can derive some simple asymptotic bounds on the number of messages and on convergence time. Distributed ADH merges only two fragments at each round. As a consequence, if the number of multicast nodes is p, $p-1$ rounds are needed before a solution is found. We now assume that a single spanning tree

is constructed during the first round and used throughout algorithm execution. The construction of this spanning tree, using a simple distributed algorithms, is characterized by an $O(m)$ as the number of messages and $O(diam)$ (where $diam$ is the network diameter) as convergence time. At each round the combined process of broadcasting and convergecasting involves $O(n)$ messages and $O(h)$ time to converge (where h is the height of the spanning tree), whereas the merging process involves $O(diam)$ both as the number of messages and as convergence time. In this way, the overall number of messages exchanged for distributed ADH is $O(m + p \cdot n)$ and the convergence time is $O(p \cdot diam)$. Since, in the worst case, distributed ADHF performs as distributed ADH, the two algorithms share the same asymptotic bounds on the number of messages and on convergence time. As demonstrated in the following section, the experimental results confirm this analytical behavior for both heuristics proposed here. Although in the worst case, the reduced node set technique does not modify the asymptotic bounds previously derived, the empirical evidence indicates that the number of nodes involved in the process of multicast tree construction is significantly reduced, so thus the number of messages exchanged and the convergence time required.

5 Performance Analysis

In order to evaluate the proposed heuristics we performed extensive testing, both in simulation and on real topologies, which have been deployed on a cluster of active nodes. Active Networks are a novel approach to network architecture in which the switches of the network perform customized computations on the messages flowing through them. This supports faster service innovation by making it easier to deploy new protocols and network services, even over wide areas [21]. The decision to use two different testing environments is designed to investigate the performance of the algorithms from different perspectives. On the one hand, simulations allow the analysis of the cost competitiveness of distributed ADH and ADHF on a large set of randomly generated test networks characterized by a large number of nodes and different topology models. On the other hand, tests on active networks deployed on a cluster of real nodes allow us to study communication complexity and convergence time better, by taking into account some important characteristics of real networks (such as concurrent access to resources, sharing of network resources, etc.). However, the experimental activity on active networks, since it involves tests on several topologies and the management of complex distributed algorithms, is a very complex task, especially when the complexity of the topology grows. In order to cope with these problems, we used a software framework for Active Networks and Active Applications Management [1] that makes it easier and faster to carry out experiments, relieving the applications programmer from secondary details which are specific to the Execution Environment.

In the following subsections we present the results of these performance experiments. We first describe the network model, then define the performance

Group	Net. Nr.	Net. size	M.cast Nodes	Generation Criterion	Comp. Term	Metric
A	100	1000	20%	BRITE	best	$cost \in R^+$
B	100	1000	20%	BRITE	best	$cost \in R^+$
C	100	150	10%	BRITE	optimal	$cost \in R^+$
D	100	1000	10%	Internet subgraph	best	$hop\ count$
E	100	2000	10%	Internet subgraph	best	$hop\ count$

Table 1: Summary of experimental sets of sample networks

metrics and, finally, present and discuss the results.

5.1 Network Model

We compared Steiner heuristics on both randomly constructed test networks, and sub-networks extracted from a complete Internet topology using an "oil-spot technique". For the first group of experiments we adopted the BRITE (Boston university Representative Internet Topology gEnerator) network generator [13], and for the latter we used a simple extraction method on the map of Internet obtained by the project Mercator in 1999 [7].

We considered several test groups, each containing 100 sparse networks, that is, networks where the number of edges is less than twice the number of nodes. The networks have 10% or 20% of their nodes in the multicast group. We believe that these choices describe plausible multicast applications in wide area networks (WAN), where the topology graph is likely to be loosely interconnected, and transmission is likely to involve only a minority of the nodes in the network. Moreover, these topological configurations represent the most difficult cases of the Steiner problem. In section 5.3, we also discuss how the proposed heuristics scale with increasing multicast membership and network size.

5.2 Evaluation Metrics

The metrics we used for comparison are competitiveness, convergence time and the number of messages transmitted. Competitiveness is the ratio between the heuristic tree cost and that of the optimal solution. For large networks where explicit algorithms capable of finding optimal solutions are prohibitively expensive, we used the best solution obtained by any heuristic considered rather than an optimal solution. Convergence time is the time elapsed from the beginning of the execution to the time at which last message reaches its destination. The number of messages is the total number of messages exchanged between nodes before convergence.

5.3 Experimental Results

Simulations have been carried out on five different groups of networks each containing 100 randomly generated or extracted topologies. On these networks, we compared the cost competitiveness of the heuristics proposed (distributed

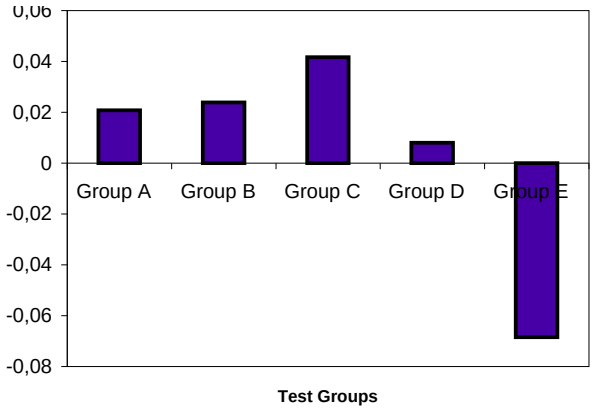

Figure 1: *D-ADHF Solution Cost Percentage Deviation from D-ADH*

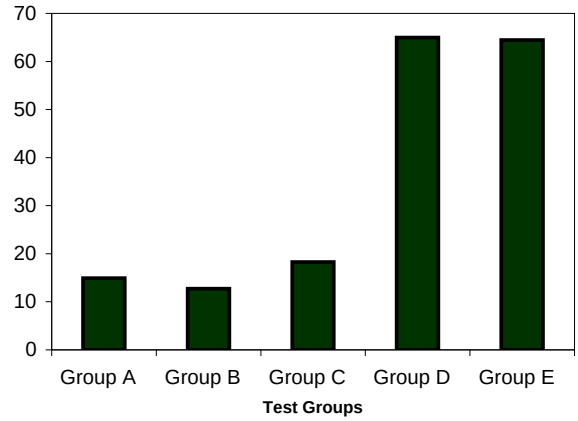

Figure 2: *D-ADHF Round Number Percentage Deviation from D-ADH*

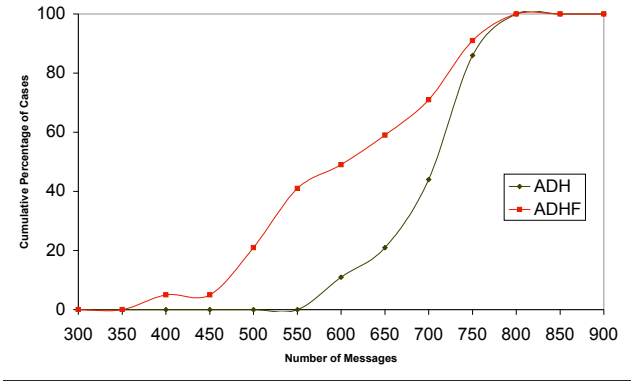

Figure 3: *Cumulative percentage of networks solved within a given number of messages*

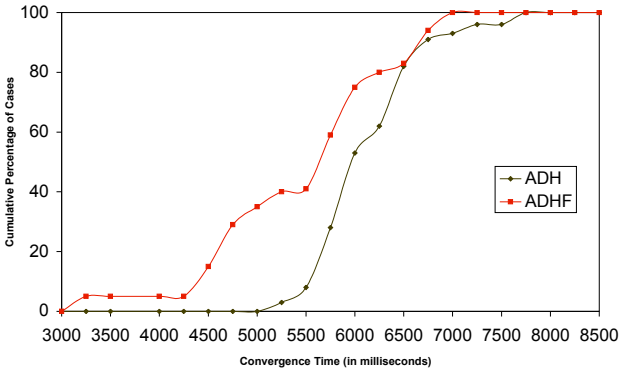

Figure 4: *Cumulative percentage of networks solved within a given time*

ADH and ADHF) with some classical heuristics (DNH, SPH and K-SPH). Table 1 summarizes the most important features of these network groups. The first result can be considered as a correctness proof of the proposed algorithms, since distributed ADH and ADHF provided solutions that have the same level of competitiveness when compared to their centralized versions. This is also a performance result, since, as reported in [3], the distributed version of SPH and K-SPH heuristics may provide worse solutions compared to their centralized versions. The reason for this drawback can be attributed to the lack of global topology information. However, it is not present in our algorithms given the sharing of critical information by means of broadcasting and converge-casting along the spanning tree.

The second result is that, when comparing competitiveness, distributed

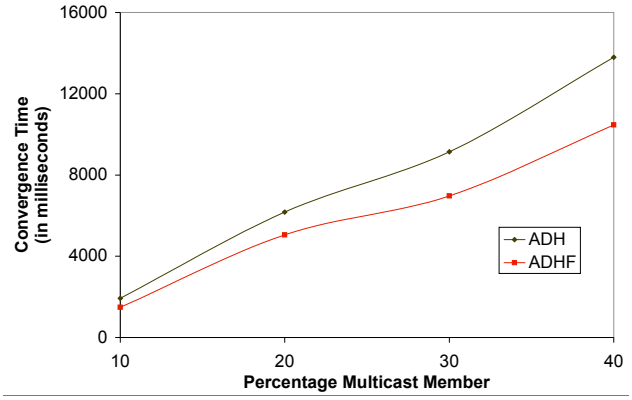

Figure 5: *Average convergence time when the multicast group size is varied from 10% to 40%*

heuristics ADH and ADHF consistently outperformed the centralized heuristics DNH, SPH and K-SPH. Moreover, from the comparison of the number of rounds used by D-ADH and D-ADHF, it emerges that D-ADHF uses significantly fewer rounds. This is mainly due to the greedy approach adopted by D-ADHF in merging the fragments. D-ADH reduces the forest fragmentation connecting only two fragments at each round, whereas D-ADHF allows multiple fragments of the forest to be combined in a single round. This allows D-ADHF to provide both lower convergence time, and fewer messages, since the values of these evaluation metrics grow with the number of rounds, still maintaining the competitiveness results. Figures 1 and 2 show the D-ADHF percentage deviations from the D-ADH for solution values and for the numbers of rounds respectively. It should be noted that in the first case, the deviation represents an incremental percentage cost, whilst in the second case, the percentage refers to the reduction in the number of rounds. As clearly shown by the experiments reported above, the performance of the two algorithms is almost identical in terms of competitiveness, but the ADHF saving in the number of rounds represents a noticeable improvement, especially if we consider the distributed nature of the algorithms.

In the following experiments, we carried out a set of measurements on the active network test-bed which allows us to cope with more realistic features, as described at the beginning of this section.
In the following experiment, we compared the performance of both the heuristics on a hundred instances of topologies, each with fifty nodes, 20% of which were set in the multicast group. Figure 3 shows the cumulative percentage of networks solved within a given number of messages for both of the algorithms proposed. Likewise, chart 4 plots the cumulative percentage of networks solved

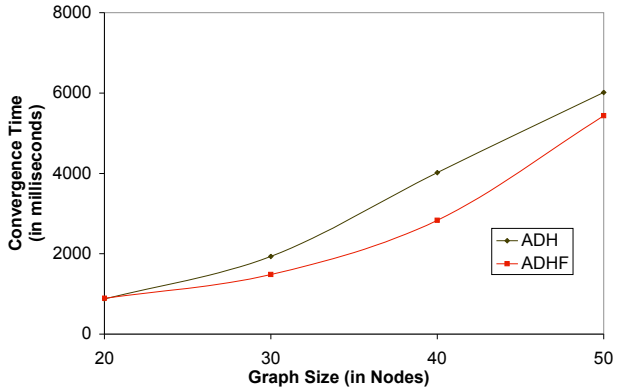

Figure 6: *Average convergence time when the network size is varied from 20 nodes to 50 nodes*

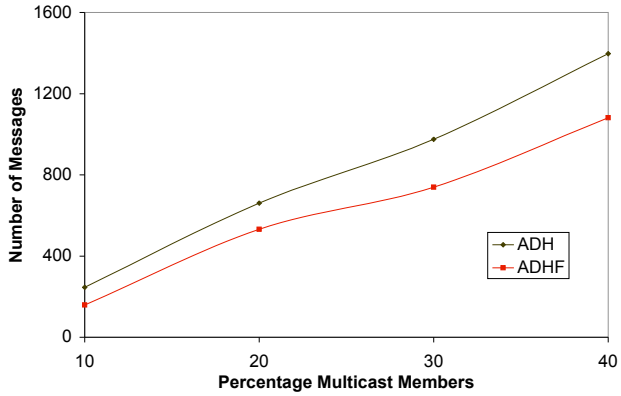

Figure 7: *Average number of messages transmitted when the multicast group size is varied from 10% to 40%*

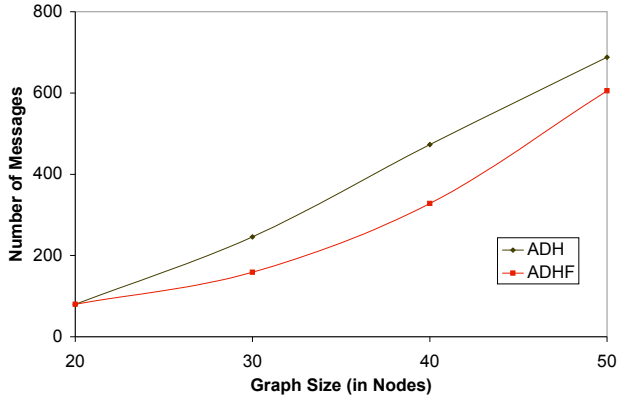

Figure 8: *Average number of messages transmitted when the network size is varied from 20 nodes to 50 nodes*

within a given convergence time. Both the number of messages and the convergence time for D-ADHF fall within a more limited range as compared to the results produced by D-ADH. Although both heuristics share identical theoretical upper bounds, this result is consistent with the observed reduction in the number of rounds used.

In order to investigate the scaling capability of the two proposed distributed heuristics with the size of network and multicast group, we performed additional tests summarized by the following figures. Figures 5 and 6 show the average convergence time for both ADH and ADHF, when either the multicast group size is varied between 10% and 40% of the total number of nodes, or when network size is varied between 20 and 50 nodes. Each point in the graph indicates the average value for 40 test networks. Similarly, Figures 7 and 8 summarize the average number of messages transmitted, when membership and network size, respectively, are varied.

In order to evaluate the performance of the algorithms when the *reduced node set* technique, which has been described in section 4.3, is adopted, we report the results of the D-ADH execution on four further groups of sample networks. We chose to analyze the behaviour in such cases where the basic algorithms show more inefficiences, that is when the multicast nodes are located only in a limited area of the network. All the four groups are constituted by one hundred topologies generated by the *Brite* generator. Topologies in groups A_R and C_R are populated by 1000 nodes with respectively 10% and 20% in the multicast groups, whilst those in groups B_R and D_R are populated by 2000 nodes with analogous percentages in the multicast groups. Figure 9 shows the D-ADH incremental percentage cost due to the adoption of the *reduced node set* technique, whilst the chart in Figure 10 plots the D-ADH percentage reduction in the num-

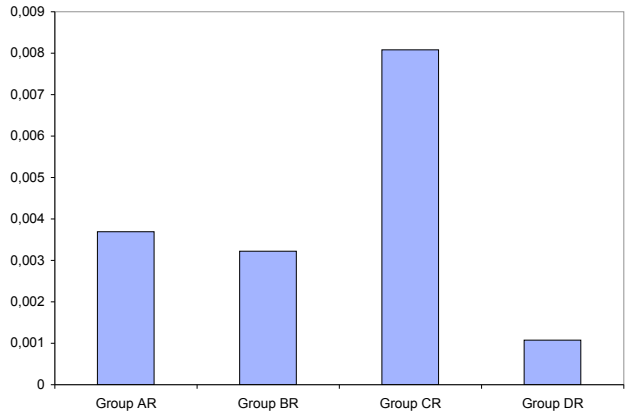

Figure 9: *Incremental cost percentage with the reduced node set technique*

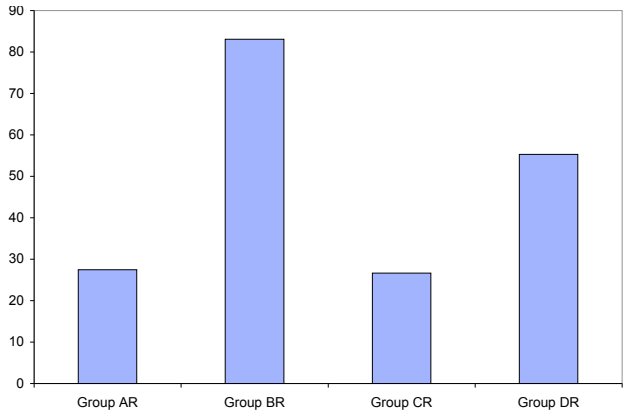

Figure 10: *Node percentage reduction with the reduced node set technique*

ber of nodes involved in the multicast tree construction. It is straightforward to observe that the cost competitiveness remains almost unchanged, but in the considered scenarios the proposed technique allows an interesting saving in the number of network nodes that participate in the execution of distributed algorithms.

6 Concluding Remarks

The development of distributed algorithms for the Steiner Problem in Networks has received very little attention over the past few years. Very few proposals of distributed algorithms are to be found in the literature in clear contrast with the quantity of centralized versions. In this paper, we study distributed versions of ADH and ADHF centralized heuristics which are capable of determining the best approximating solutions for the SPN. We designed and implemented the distributed versions in such a way that they maintain all the properties of their centralized versions. The experimental results confirmed our design conjecture. Furthermore, we designed a simple technique which allows to reduce the number of network nodes involved in the algorithm execution. For the ADH algorithm, the overall number of messages exchanged has been demonstrated to be $O(m + p \cdot n)$, with convergence time being $O(p \cdot diam)$. Since, in the worst case, distributed ADHF performs as distributed ADH, the two algorithms share the same asymptotic bounds on the number of messages and on convergence time. However, the experimental results highlighted an advantage of distributed ADHF, since it allows a noticeable saving in the number of rounds, although it still maintains comparable performance in terms of competitiveness.

References

[1] A. Barone, P. Chirco, G. Di Fatta, and G. Lo Re. A management architecture for Active Networks. In *Proc. of AMS 2002 - 4th Annual Int.nl Workshop on Active Middleware Services*, pages 41–48. IEEE, July 2002.

[2] F. Bauer and A. Varma. Degree-constrained multicasting in point-to-point networks. In *Proc. IEEE INFOCOM, Boston*, pages 369–376, April 1995.

[3] F. Bauer and A. Varma. Distributed algorithms for multicast path setup in data networks. *IEEE/ACM Trans. on Networking*, 4(2):181 – 191, April 1996.

[4] M. Bern and P. Plasmann. The Steiner problem with edge lengths 1 and 2. In *Information Sciences*, number 74(1-2), pages 73–96, October 1993.

[5] G. Chen, M. Houle, and M. Kuo. The Steiner problem in distributed computing systems. *"Inform. Sciences"*, 74(12):73–9, October 1993.

[6] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66 – 77, Jan. 1983.

[7] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *Proc. of Infocom 2000*, pages 1371–1380. IEEE, March 2000.

[8] C. Hedrick. Routing Information Protocol. In *Request for Comments*, number 1058. IEEE, June 1988.

[9] R. M. Karp. Reducibility among combinatorial problems. In *Miller and Thatcher (Eds.), Complexity of Computer Computations*, pages 85–103. Plenum Prest, New York, 1972.

[10] V. Kompella, J. Pasquale, and G. Polyzos. Two distributed algorithms for the constrained Steiner tree problem. In *Proc. Comput. Commun. and Netw.*, Jun. 1993.

[11] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15(2):141–145, 1981.

[12] N. Lynch and M. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, September 1989.

[13] A. Medin, A. Lakhina, I. Matta, and J. Byers. *BRITE Universal Topology Generator User Manual*. BU-CS-TR-2001-003. April 05,, 2001.

[14] R. Novak, J. Rugelj, and G. Kandus. A note on distributed multicast routing in point-to-point networks. *Computers and Operations Research*, 28:1149–1164, 2001.

[15] J. Plesnik. Worst-case relative performances of heuristics for the Steiner problem in graphs. *Acta Math. Univ. Comenianae*, 60(2):269–284, 1991.

[16] V. J. Rayward-Smith. The computation of nearly minimal Steiner trees in graphs. *Int. J. Math. Educ. Sci. Technol.*, 14:15–23, 1983.

[17] V. J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.

[18] G. Singh and K. Vellanki. A distributed protocol for constructing multicast tree. In *Proc. of International Conference on Principles of Distributed Systems*, pages 41–48. IEEE, Dec. 1998.

[19] M. Smith and P. Winter. Path distance heuristics for the Steiner problem in undirected networks. *Algorithmica*, 7(23):309–327, 1992.

[20] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24(6):573–577, 1980.

[21] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, and G. J. Minden. A survey of Active Network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[22] S. Voss. Steiner's problem in graphs: heuristic methods. *Discrete Applied Mathematics*, 40:45–72, 1992.

[23] S. Voss. Worst-case performance of some heuristics for Steiner's problem in directed graphs. *Information Processing Letters*, 48:99–105, 1993.

[24] B. Waxman and M. Imase. Worst-case performance of Rayward-Smith's Steiner tree heuristics. *Inform. Proc. Lett.*, 29:283–287, 1988.

[25] P. Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987.

A Appendix

A.1 The I/O Automaton Model

In the following, we present a formal specification of the D-ADH algorithm, using the *Input/Output (I/O) automaton model* and we describe the transition relation in a *precondition-effect* style. The Input/Output automaton model, developed by Lynch and Tuttle [12], is a labelled transition system model for components in asynchronous concurrent systems. The actions of an I/O automaton are classified as input, output and internal actions, where input actions are required to be enabled all the time. An I/O automaton has *"tasks"*. In a fair execution of an I/O automaton, all tasks are required to get turns infinitely often. The behavior of an I/O automaton can be described in terms of *traces*, or, alternatively, in terms of *fair traces*. Both types of behavior notions are compositional. Sections A.2.1-A.2.5 of the Appendix report the I/O automaton components. In section A.2.6, we give descriptions of some of the utility functions we assume in the formal description.

A.2 The distributed ADH algorithm

An *I/O automaton* A, often simply referred to as *automaton*, consists of five components:

- *sig(A)*, a *signature*
- *states(A)*, a (not necessarily finite) set of *states*
- *start(A)*, a nonempty subset of $states(A)$ known as the *start states* or initial *states*
- *trans(A)*, a *state-transition relation*, where $trans(A) \subseteq states(A) \times act(sig(A)) \times states(A)$. This must have the property that for every state s and every input action π, there is a transition $(s, \pi, s') \in trans(A)$
- *tasks(A)*, a *task partition*, which is an equivalence relation on $local(sigA))$ having at most countably many equivalence classes

In our case, the description of the D-ADH algorithm involves the definition of a single automaton, as each node in the network should be able to perform all the actions that are necessary to achieve the multicast transmission tree. For this reason, we designed a single automaton whose instances are replicated on each node in the network. Moreover, a simple I/O automaton models the reliable FIFO message channels.
The first component of an I/O automaton consists of the set of actions that can be performed by the automaton. This set is known as its *"signature"* and it is partitioned in the subsets of *input*, *output* and *internal actions*.

A.2.1 Signature

For each process automaton P_i, the following actions are defined (where M is the set of :

Input :
$init(m)_i, m \in M$
$receive(m)_{j,i}, m \in M, j \in nbrs$

Output :
$send(m)_{i,j}, m \in M, j \in nbrs$

Internal :
$compute_i$
$report_i$
$root-report_i$
$reset-state_i$

A.2.2 States:

For each process automaton P_i, the states in $states(P_i)$ consist of the following components:

- *status*, a variable with values in $W = \{z, s, o, u\}$, initially u. It represents the node state (with z multicast group node, s Steiner node, o node external to the multicast tree, and u node with a still undefined state);

- *frgs-list*, a list of nodes, associated with the corresponding leaders, with values belonging to type (host, host), initially empty (*nil*). It represents the current forest of fragments[1];

- *f-adh*, with values belonging to type (host, float, host list), initially $(null, \infty, nil)$. It represents a node, with a computed value of f, and the list of all the border nodes along the different paths towards the fragments considered in the summation that has yielded the value of f;

- *best-tgts-list*, a list with values of type host, initially empty (*nil*). It represents the list of all the border nodes along the different paths towards the fragments considered in the summation that has yielded the *best* value of

[1] To make the description clearer, we here suppose that the data structure containing the information about the multicast forest is a list of ordered pairs, where the first component is a node UID and the second one is the UID of the associated fragment leader. We also assume that the node UIDs are values of type host, with a special value, *null*, not associated to any node.

f;

- *values*, a list with values of type (host, float, host list), initially empty (*nil*). It represents the information reported through the convergecast process (this information is used by each node to determine the best value of f in its own subtree);

- *parent*, with values in $nbrs \cup \{null\}$, initially *null*. It represents the parent node in the spanning tree, during the current round;

- *root*, a boolean, initially *false*. It shows that the node is the source (root) of the spanning tree, during the current round;

- *computed*, a boolean, initially *false*. It shows that the node has completed the computation of its own value of f;

- *reported*, a boolean, initially *false*. It shows that the node has reported to its parent the best known value of f, through the convergecast process;

- *completed*, a boolean, initially *false*. It shows that the algorithm has terminated;

- *first-round*, a boolean, initially *true*. It shows that the algorithm is in its first round of execution;

- *converged*, a subset of *nbrs*, initially $\emptyset$. It represents the set of neighbor nodes which an acknowledgement has been received from;

- *rip-table*, a table with values of type (host, host, int). It represents the network routing table, as computed by an underlying unicast routing protocol (e.g., RIP);

- *nbrs-in-spanning-tree*, a list with values of type host, initially empty (*nil*). It represents the set of neighboring nodes in the spanning tree;

- *nbrs-in-mc-tree*, a list with values of type host, initially empty (*nil*). It represents the set of neighboring nodes in the multicast tree achieved;

- *nbr-to-tgt*, with values of type host, initially *null*. It represents the next node along the path towards a target node;

- for every $j \in nbrs$:
  $send(j)$, a FIFO queue of messages in M, initially empty. It represents the outgoing message queue associated with the neighbor node j.

A.2.3 Start:

The set of start states $start(P_i)$ consists of the single state defined by the initializations.

A.2.4 Transitions:

/* The automaton receives an input value ("*mcast*", l) from the external environment, where l is the list of multicast group nodes. The automaton assumes the role of root node ($root := true$) for the spanning tree, during the first round, creating from l the list *frgs-list*, containing the information about the multicast forest (nodes and respective leaders). It sets its own *status* (node in the group or not), and adds the message to be sent in broadcast to the queues associated with each neighbor node. */

$init($"$mcast$"$, l)_i$
  Effect:
    $root := true$
    $frgs - list := expand - list(l)$
    if $is - z(frgs - list) = true$ then
      $status := z$
    else
      $status := o$
    for all $k \in nbrs$
      add ("$bcast$"$, frgs - list$) to $send(k)$

/* The automaton receives an input value ("*bcast*", l) from a neighbor node j, where l is the list containing the information about the current fragment forest. If the automaton is in its first round of execution (i.e., $first - round = true$), then, if it has not yet received a message of this type (i.e., $frgs - list = null$), then it stores the information contained in the list l, adds j to the list of neighbor nodes in the spanning tree (setting j as its *parent* node), sets its *status*, and adds the message received to the queues associated with each neighbor node in the network, except j; otherwise (i.e., $if\ frgs - list \neq null$), it adds an acknowledgement message ("$bcast - ack''$) to the queue associated with j, in order to notify it that it is not its child in the spanning tree. If the automaton

is not in its first round of execution (i.e., $first - round = false$), then it stores the information contained in the list l, sets j as its *parent* node during the current round, and adds the message received to the queues associated with each neighbor node in the spanning tree, except j */

$receive(\text{"}bcast\text{"}, l)_{j,i}$
Effect:
  if $first - round = true$ then
    if $frgs - list = null$ then
      $frgs - list := l$
      $parent := j$
      add $parent$ to $nbrs - in - spanning - tree$
      if $is - z(frgs - list) = true$ then
        $status := z$
      else
        $status := o$
      for all $k \in nbrs - \{j\}$
        add ("$bcast$", $frgs - list$) to $send(k)$
    else
      add "$bcast - ack$" to $send(j)$
  else
    $frgs - list := l$
    for all $k \in nbrs - in - spanning - tree - \{j\}$
      add ("$bcast$", $frgs - list$) to $send(k)$

/* The message m is at the front of the queue associated with the neighbor node j. The automaton removes the message m from the queue. */

$send(m)_{i,j}, m \in M$
  Precondition:
    m is first in $send(j)$
  Effect:
    remove first element of $send(j)$

/* The automaton receives an input value *"bcast-ack"* from a neighbor node. The automaton updates the set, *converged*, containing the neighbor nodes from which it has already received a convergecast message.*/

$receive(\text{"}bcast - ack\text{"})_{j,i}$
  Effect:
    $converged := converged \cup \{j\}$

/* The automaton has already received the information about the multicast forest ($frgs - list \neq null$), but it has not yet computed its own value of f ($computed = false$). It computes its own value of f, adds this information to the list $values$ (together with its own UID and the list of border nodes of the shortest paths towards the fragments considered in the summation that has given the best value of f), and it marks itself to demonstrate completion of the current computation ($computed := true$). */

$compute_i$
  Precondition:
    $frgs - list \neq null$
    $computed = false$
  Effect:
    $fadh := f(frgs - list, rip - table, status)$
    add $fadh$ to $values$
    $computed := true$

/* The automaton, that is not the root of the spanning tree ($root \neq true$) during the current round, has already been inserted in the tree ($parent \neq null$), has already computed its own value of f ($computed = true$), and has already received a convergecast message from all its neighbor nodes in the spanning tree except its parent ($converged = nbrs - in - spanning - tree - \{parent\}$) - or, in the first round, from all its neighbor nodes except its parent in the tree ($converged = nbrs - \{parent\}$) - , but it has not yet performed the convergecasting towards the root node ($reported = false$). The automaton determines the information associated to the best value of f ($best\text{-}cand$), adds a message containing this information to the queue associated to the $parent$ node, marks itself showing the performance of convergecasting ($computed := true$), and removes any stored information about the current round of execution ($frgs - list := null$, $computed := false$, $converged := null$). */

$report_i$
  Precondition:
    $root \neq true$
    $parent \neq null$
    $computed = true$
    $converged = nbrs - \{parent\}$ (if $first - round = true$)
    $converged = nbrs - in - spanning - tree - \{parent\}$ (if $first - round = false$)
    $reported = false$
  Effect:
    if first-round = true then
      first-round = false
    else
      ()

let $best - cand := not - greater(values)$
in
add ("$candidate$", $best - cand$) to $send(parent)$
$reported := true$
$parent := null$
$frgs - list := null$
$computed := false$
$converged := null$

/* The automaton receives an input value ("$candidate''$, c) from a neighbor node j. If the automaton is in the first round of its execution, then it adds j to the list of neighbor nodes in the spanning tree. In any case, the automaton adds c to the list *values*, and updates the set *converged* of the nodes from which it has received a convergecast message,*/

$receive($"$candidate''$, $c)_{j,i}$
Effect:
if $first - round = true$ then
add j to $nbrs - in - spanning - tree$
else
()
add c to $values$
$converged := converged \cup \{j\}$

/* The automaton, which is the root of the spanning tree ($root = true$) during the current round, has already computed its own value of f ($computed = true$) and received a convergecast message from all its neighbor nodes in the spanning ($converged = nbrs - in - spanning - tree - \{parent\}$) - or, in the first round, from all its neighbor nodes ($converged = nbrs - \{parent\}$) - , but it has not yet determined the node with which the best value of f is associated ($reported = false$). The automaton determines the information associated with the best value of f in the whole network (*winner*), adds a message with the list *frgs-list* containing the information about the multicast forest and the list *tgt-list* of border nodes of the shortest path towards the fragments considered in the summation that has given the best value of f. It marks itself showing the determination of the most central node ($reported := true$), and removes any stored information about the current round of execution ($root := null$, $frgs - list := null$, $computed := false$, $converged := null$). */

$root - report_i$
Precondition:
$root = true$
$computed = true$

$converged = nbrs$ (if $first - round = true$)
$converged = nbrs - in - spanning - tree$ (if $first - round = false$)
$reported = false$
Effect:
if first-round = true then
first-round = false
else
()
$best - tgts - list :=\#3(not - greater(values))$
let $winner := fst(not - greater(values))$
in
add ("$winner$", $frgs - list, best - tgts - list$) to $send(winner)$
$reported := true$
$root := false$
$frgs - list := null$
$computed := false$
$converged := null$

/* The automaton receives an input value ("$winner''$, l, t) from the node that is the root of the spanning tree during the current round, where l is the list containing the information about the multicast forest and t is the list of border nodes of the shortest paths towards the fragments that have to be connected. The automaton, if associated with a node not yet included in the multicast forest, updates its own status ($status := s$) and the information about the forest (including the node in the forest). The automaton determines the border node, *tgt*, along the shortest path towards the first fragments that have to be connected, and determines the first node, *nbr-to-tgt*, along the path towards *tgt*. It adds *nbr-to-tgt* to the list of neighbor nodes in the multicast tree and adds a message ("$merge''$, (l, tgt, i)) to the queue associated with the neighbor node *nbr-to-tgt*. */

$receive("winner", l, t)_{j,i}$
Effect:
if $status = o$ then
$status := s$
$l := add - me(l, i)$
else
()
$best - tgts - list := t$
$tgt := hd(best - tgts - list)$
$best - tgts - list := tl(best - tgts - list)$
$nbr - to - tgt := next - hop(tgt, rip - table)$
add $nbr - to - tgt$ to $nbrs - in - mc - tree$
add ("$merge$", (l, tgt, i)) to $send(nbr - to - tgt)$

/* The automaton receives an input value ("$merge$", $(l, k, leader)$), form a neighbor node j, where l is the list containing the information about the multicast forest, k is the UID of the border node along the path towards the fragment that has to be connected and *leader* is the UID of the leader node of the new fragment (i.e., the most central node in the current round). The automaton adds j to the list of neighbor nodes in the multicast trees. The automaton, if associated with a node different from the node k, updates its own status (if necessary) and the information about the multicast forest (the associated node is inserted in the forest). It determines the first node, *nbr-to-tgt* , along the path towards k, adds *nbr-to-tgt* to the list of neighbor nodes in the multicast tree, and adds a message ("$merge$", (l, k, i)) to the queue associated to the neighbor node *nbr-to-tgt*. Otherwise, (i.e., if the automaton is only associated with node k), it updates the information about the multicast forest (performing the merging process of target fragment), it determines the first node, *nbr-to-tgt* , along the path towards the most central node and it adds a message ("$merg - ack$", $l, leader$) to the queue associated to the neighbor node *nbr-to-tgt*. */

$receive$("$merge$", $(l, k, leader)$)$_{j,i}$
  Effect:
  add j to $nbrs - in - mc - tree$
    if $k \neq i$ then
      if $status = o$ then
        $status := s$
      else
        ()
      $l := add - me(l, leader)$
      $nbr - to - tgt := next - hop(k, rip - table)$
      add $nbr - to - tgt$ to $nbrs - in - mc - tree$
      add ("$merge$", $(l, k, leader)$) to $send(nbr - to - tgt)$
    else
      $l := merge - my - frg(l, leader)$
      $nbr - to - tgt := next - hop(leader, rip - table)$
      add ("$merg - ack$", $l, leader$) to $send(nbr - to - tgt)$

/* The automaton receives an input value ("$merg - ack$", $l, leader$) from a neighbor node j, where l is the list containing the information about the multicast forest and *leader* is the UID of the leader node of the new fragment (i.e., the most central node in the current round). The automaton, in the case where it is running on a node different from the node *leader*, determines the first node, *nbr-to-tgt*, along the path towards the most central node and adds a message ("$merg - ack$", $l, leader$) to the queue associated with the neighbor node *nbr-to-tgt*. Otherwise, (i.e., if the automaton is the one on the most central node), if there are fragments that have still to be connected ($best - tgts - list \neq nil$), it determines the border node, *tgt*, along the path towards the

next fragment that has to be connected, determines the first node, *nbr-to-tgt*, along the path towards tgt, adds *nbr-to-tgt* to the list of neighbor nodes in the multicast tree, and adds a message ("$merge$", (l, tgt, i)) to the queue associated with the neighbor node *nbr-to-tgt*. Otherwise, (i.e., if there are not any more fragments to be connected in the current round), if the whole forest is still not connected ($check{-}end(l) = false$), the automaton starts a new round, assuming the role of root node ($root := true$) and adding the message, ("$bcast$", $frgs - list$), with the information about the forest that has to be broadcast to the queues associated with all neighbor nodes in the spanning tree. Otherwise, (i.e., if the forest is already connected) the algorithm terminates. */

$receive($"$merg - ack$", $l, leader)_{j,i}$
Effect:
  if $i \neq leader$ then
    $nbr - to - tgt := next - hop(leader, rip - table)$
    add ("$merg - ack$", $l, leader$) to $send(nbr - to - tgt)$
  else
    if $best - tgts - list \neq nil$ then
      $tgt := hd(best - tgts - list)$
      $best - tgts - list := tl(best - tgts - list)$
      $nbr - to - tgt := next - hop(tgt, rip\_table)$
      add $nbr - to - tgt$ to $nbrs - in - mc - tree$
      add ("$merge$", (l, k, i)) to $send(nbr - to - tgt)$
    else
      if $check - end(l) = false$ then
        $root := true$
        $frgs - list := l$
        for all $k \in nbrs - in - spanning - tree$
          add ("$bcast$", $frgs - list$) to $send(k)$
      else
        $completed := true$

A.2.5 Tasks:

/* The task partition contains one task for any action $send_{i,j}$, for each $j \in nbrs$, and one task for all the *compute*, *report*, *root-report*, and *reset-state* actions. */

$\{compute_i, report_i, root{-}report_i, reset{-}state_i\}$
for every $j \in nbrs$:
  $\{send(m)_{i,j}, m \in M\}$

A.2.6 Utility functions

We assume that the following functions are defined:

As explained in section 4.1, the algorithm assumes a specific data structure which represents the multicast forest. The function:

$$expand{-}list : hostlist \rightarrow (host, host)list$$

given the list of multicast group nodes, creates the data structure representing the multicast forest.

During the first round, the function:

$$is{-}z : (host, host)list \rightarrow bool$$

checks to see whether the node belongs to the multicast group, assuming as parameter the list representing the multicast forest.

The cost metric of the D-ADH algorithm is determined by means of the function:

$$f : ((host, host)list, (host, host, itf)list, W) \rightarrow \\ \rightarrow (host, float, hostlist)$$

The f function, given the lists which represent the multicast forest, the routing table and a variable W which codes the status of the node, computes the value of f according to equations 1 and 2, and returns the node UID, the computed value of f and the list of nodes that are tails of minimum cost paths on the target fragments.

$$not - greater : (host, float, hostlist)list \rightarrow \\ \rightarrow (host, float, hostlist)$$

given a list of values of type (host, (int, int), host list), where the first component is the node UID, the second is its own value of f, and the third is the list of the border nodes along the paths towards the fragments considered in the summation that gives the best value of f[2]), it determines the information about the best value of f;

$$next{-}hop : (host, (host, host, int)list) \rightarrow host$$

given the UID of a target node and the list containing the unicast routing information, it determines the UID of the first node (next-hop) along the shortest

[2]The list of the border nodes refers to all the fragments considered in the summation in the variant with full connection (D-ADHF), while in the base version (D-ADH), it refers only to the closest fragment or to the two closest fragments.

path towards the target node;

$add{-}me : ((host, host)list, host) \rightarrow (host, host)list$
given the list containing the information about the multicast forest and the UID of a leader node, it updates the multicast forest list adding a new pair, with the node as the first component and the given leader as the second one;

$$merge{-}my{-}frg : ((host,host)list, host) \rightarrow \\ \rightarrow (host, host)list$$

given the list containing the multicast forest and the UID of a leader node, it updates the multicast forest list, setting the new leader UID in all the pairs which have the same leader of the node as their second component;

$check{-}end : (host, host)list \rightarrow bool$

given the list containing the information about the multicast forest, it checks if the forest is connected (i.e., if all the nodes in the forest have the same leader).