Advances in Wireless Networks and Mobile Computing
D.-Z. Du and G. Xue (Eds.) pp. – - –

SPLAST: A Novel Approach for Multicasting in Mobile Wireless Ad Hoc Networks

Yehuda Ben-Shimol
Communication Systems Engineering Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, P.O.B. 653, Israel
E-mail: benshimo@bgu.ac.il

Amit Dvir
Communication Systems Engineering Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, P.O.B. 653, Israel
E-mail: azdvir@bgu.ac.il

Michael Segal
Communication Systems Engineering Department
Ben-Gurion University of the Negev, Beer-Sheva 84105, P.O.B. 653, Israel
E-mail: segal@bgu.ac.il

Contents

Abstract

A Mobile Ad Hoc Network (MANET) is a network architecture that can be rapidly deployed without relying on pre-existing fixed network infrastructure. Group communication is a common application where a single source sends identical information concurrently to multiple destinations. Trees of special properties are required to provide efficient network management of such applications. Usually, such trees have to balance between the requirements to minimize the total tree cost (derived from energy constraints) and the requirement to minimize the maximal shortest path (derived from delay constraints). This paper presents a novel solution for efficient multicast trees that fulfill both requirements called *SPLAST*. A detailed discussion covers the development process, starting from centralized static solution, through distributed implementation to a complete distributed algorithm that cope with various scenarios that are relevant to wireless ad hoc networks by efficient management and maintenece of the underlying components of the algorithm. Simulation inquiry shows that the average performance of *SPLAST* is attractive as well.

1 Introduction

A Mobile Ad Hoc Network (MANET) is a network architecture that can be rapidly deployed without relying on pre-existing fixed network infrastructure [1]. Wireless communication is used to deliver information between nodes,

which may be mobile and rapidly change the network topology. The wireless connections between the nodes (which later will be referred as links or edges) may suffer from frequent failures and recoveries due to the motion of the nodes and due to additional problems related to the propagation channels (e.g. obstructions, noise) or power limitations.

Group communication is the basis for numerous applications in which a single source delivers concurrently identical information to multiple destinations. This is usually obtained with efficient management of network topology in the form of tree having specific properties. For example, multicast routing refers to the construction of a spanning tree rooted at the source and spanning all destinations [2, 3, 4, 5]. Delivering the information only through edges that belong to the tree generates an efficient form of group communication which uses the smallest possible amount of network resources. In contrast, with unicast routing from the source to each destination, one needs to find a path from the source to each destination and generates an inefficient form of group communication where the same information is carried multiple times on the same network edges and the communication load on the intermediate nodes may significantly increase.

Generally, there are two well-known basic approaches to construct multicast trees: the minimal Steiner tree (SMT) and the shortest path tree (SPT). Steiner tree (or group-shared tree) tends to minimize the total cost of a tree spanning all group nodes with possibly additional non group member nodes. The construction of the SMT is known to be a NP-hard problem [6, 7]. Some heuristics that offer efficient solutions to this problem are given in [8, 9, 10]. To the best of our knowledge the best solution was derived by [11].

SPT tends to minimize the cost of each path from the source to each destination. This can be achieved in polynomial time by using one of the two well-known algorithms by Bellman [12] or Dossey et al. [13]. The goal of a SPT is to preserve the minimal distances from the root to the nodes without any attempt to minimize the total cost of the tree.

The problem considered in this paper is a distributed construction and maintenance of a good multicast tree with properties that are suited for mobile ad-hoc networks. This tree is required to balance between the minimization properties of the SMT and the SPT by defining two constraints. The first one states that the cost of each path from the source to any terminal in the multicast tree does not exceed a given constant factor α from the corresponding shortest path cost in the original graph. The second constraint states that the total cost of the multicast tree does not exceed a given

constant factor β from the total cost of the Minimum Spanning Tree (MST) with truncating all non group members of degree one in a recursive fashion. The truncated tree is known to be the Minimum Spanning Tree Heuristic ($MSTH$) of the SMT problem.

The proposed algorithm is based on a combination of the Light Approximate Shortest-Path Tree (LAST) algorithm given by Khuller et al. [14] and on the concept of spider spanning graphs explained in [15]. Our novel approach utilizes these two concepts with additional ideas in order to construct and maintain a new multicast tree (which we call $SPLAST$) under nodes mobility.

This paper is organized as follows. Section 2 presents essential definitions and key terms and cite related work. Section 3 presents the building blocks of the $SPLAST$ algorithm. Section 4 presents a centralized static solution that is extended to a distributed solution in section 5. Section 6 presents and analyzes a new comprehensive $SPLAST$ solution for wireless mobile scenarios. Average-case properties of $SPLAST$ are explored by thorough simulations in section 7. Finally, conclusions and recommendations are given in section 8.

2 Definitions and Key Terms

In this section we briefly introduce all the definitions and theorems that are used in the rest of the paper and are important for the understating of the present problem and its solution. We use notations and terms of graph theory and refer to other researches and results that are relevant to our work.

A *communication network* is usually defined as an undirected, connected weighted graph $G(V, E)$ where V is the set of n nodes and E is the set of undirected edges of cardinality m. Each edge $e(u, v) \in E$ connects two nodes $u, v \in V$. Every edge $e(u, v) \in E$ is assigned a non negative real value (cost) $c(e)$. A sequence of edges (path) that connects two nodes $u, v \in V$ is represented by $P(u, v)$ with total path cost $|P(u, v)| = \sum_{e_i \in P(u,v)} c(e_i)$. The cost of a spanning tree T of G is defined as $|T| = \sum_{e_i \in T} c(e_i)$. A Minimum Spanning Tree ($MST(G)$) of graph G is defined as a tree spanning all nodes in the graph with a minimum total cost [16]. Let $s \in V$ be the source node of the graph (named as *root*) and let $M \subseteq V$ be a subset of nodes that are called *terminals* (or multicast group). *Multicast Tree* refers to any tree spanning the root, all multicast members and possibly additional non

multicast members of degree larger then 1 that serve as intermediate nodes.

The *Steiner Minimal Tree* (*SMT(G)*) is the multicast tree with the minimal total cost. One difference between the SMT and the multicast tree is the special communications role of the source node in the multicast tree. Usually one would limit the distance between source and the multicast members in the multicast tree, a property which is not considered in the construction process of the Steiner tree. Therefore, in addition to the computational problems of constructing the SMT, the worst-case end-to-end path length of a SMT is not bounded ([6, 7]) and it may be as long as the longest path within the graph.

Let $d_G(v, u)$ be the minimal path cost from node $v \in V$ to node $u \in V$ in a given graph G and let $d_G(v)$ be the minimal path cost from node $v \in V$ to the root $s \in V$ in G. A *Shortest Path Tree* (*SPT(G)*) [12] T_{sp} of graph G with source $s \in V$ is defined as a tree spanning all nodes in the graph with a minimum path cost to the root such that for each node $v \in V$ in $T_{sp} : d_{T_{sp}}(v) = d_G(v)$ and the total cost of T_{sp} is not constrained. A *Short-Path-Tree* T_s of graph G with source $S \in V$ is defined as a tree spanning all nodes in G with $d_{T_{sp}}(v)/d_{T_s} \leq \delta$.

Several researches have been working on multicast tree problem. We briefly discuss some previous work here.

Kortsarz and Peleg [17] considered a *d-MST* problem, which finds a minimum weight spanning tree of a given subset of the vertex set, with diameter no more than d. Khuller et al.[14] construct a Light Approximate Shortest Paths Tree ($LAST(G)$) T which is a spanning tree of G rooted at s. For $\alpha > 1$ and $\beta \geq 1$ T is called an (α, β)- $LAST$ rooted at s if: a) for each vertex v, the distance between s and v in T is at most α times the shortest distance from s to v in G,that is $d_T(v)/d_G(v) \leq \alpha$; and, b) the total weight of T is at most β times the weight of a minimum-spanning tree of G (i.e., $|T|/|MST(G)| \leq \beta$).

Wu et al. [18] construct a light approximate routing cost spanning tree ($LART(G)$), which is at most a constant factor larger than the $MST(G)$. In addition the path cost to any vertex u from the source s is not larger than a constant factor of the routing distance of G which is defined as $c(G) = \sum \alpha_{ij} \cdot d_G(i, j)$, where α_{ij} is the requirement between nodes i and j. Only the special case where all the requirements are set to 1 was discussed in [18].

We formulate the problem of *multicast routing* as follows: given an undirected, simple, weighted graph $G(V, E)$, a group of terminals $M \subseteq V$ and a multicast root $s \in M$, find and maintain in a distributed fashion a mul-

ticast tree $T'(V', E')$, $T' \subseteq G$ and $V' \subseteq V$, $E' \subseteq E$ such that T' spans all the nodes in M and satisfies the following conditions: a) the path length in the multicast tree between the source and each node from the multicast group is as small as possible; and, b) the total weight of the multicast tree is also as small as possible. This property should be preserved efficiently under dynamical changes in edges weights. The length of unicast routing paths between the root and any other node from the multicast group are minimized in T' (under the other constraint given above). Therefore, our multicast tree efficiently preserves energy for both unicast and multicast transmissions.

In practice the weight of an edge $e(u, v)$ is defined as a function of the power transmission level of the nodes u and v that is required to establish a connection between them. Proximity changes between nodes may lead to an increase/decrease in transmission power which is then treated as a change in the edge's weight.

3 Fundamental SPLAST Techniques

We use several techniques and algorithms in order to accomplish our task of building the multicast tree. First we build a tree that is at most larger by a constant factor than the $MST(G)$, with the length of the path to any vertex u from the source s not larger by a constant from $d(u)$. Next, we decompose the obtained tree into smaller parts (which we call *spiders*) and connect them in an efficient way. In what follows we describe each step of our algorithm in more details.

3.1 Light Approximate Shortest Paths Tree (LAST)

One of the key components of our proposed algorithm is the Light Approximate Shortest Paths Tree ($LAST$) algorithm [14, 19]. The results given in [14, 19] show that a single tree or graph can balance between the minimizations of both MST and SPT trees. The $LAST$ algorithm consists of several steps: a) Build the *MST(G)* and the *SPT(G)*. b) Perform a Depth First Search (*DFS(G, s)*) walk on graph G starting with the root s. During this walk construct another graph H that holds the required solution and is set initially to $MST(G)$. The DFS walk ensures that distance from each node v to the root s is compared against $\alpha \cdot d(v, s)$. If this distance is higher than $\alpha \cdot d(v, s)$, edges from $SPT(G)$ are added to H. c) At the last step the minimal spanning tree of H is calculated and gives the (α, β) - $LAST$

approximation T.

Theorem1 **[14]:**

Let G be a graph with n nodes and m edges of non-negative edge weights. Let s be a vertex of G and α, $\beta \geq 1$ and $\beta = 1 + 2/(\alpha - 1)$. Then G contains an (α, β) - $LAST$ tree rooted at s. $LAST$ can be computed in linear time given a $MST(G)$ and a $SPT(G)$, and in $O(m + n \log n)$ time otherwise.

3.2 Spider Decomposition

Definition **1[15]:** A *spider* is a tree with at most one node of degree greater than two. A *center* of a spider is a node from which there are edge disjoint paths to the leaves of the spider. A *foot* of a spider is a leaf, or, if the spider has at least three leaves, the spider's center.

Note that if a spider has at least three leaves, its center is unique and every spider contains disjoint paths from its center to all its leaves. A *nontrivial spider* is a spider with at least two leaves.

Let G be a graph, and M be a subset of its nodes. A *spider decomposition* of M in G is a set of node disjoints nontrivial spiders in G such that the union of the feet of the spiders in the decomposition contains M.

A spider decomposition of M in G may be found as follows [15]. Let T be any rooted spanning tree of G. The depth of a node in T is defined as the distance of the node from the root. Choose a node v of maximum depth in the tree such that the sub tree rooted at v contains at least two nodes in M. Ties are broken arbitrarily. By choice of v, all the paths from the nodes in M to the node v to in the subtree of v are node-disjoint, and together with v form a nontrivial spider centered at v. We now delete the sub tree rooted at v from the tree (or the graph). If no node in M remains in the tree, we are done. If the tree contains two or more nodes in M, then we can find a spider decomposition of these nodes recursively. Otherwise, there is exactly one node in M remaining in the tree. In this case, we add the path in the tree from this node in M to the spider centered at v. This leaves a spider centered at v and we are done.

4 The Static Algorithm

In this section our Spider based LAST ($SPLAST$) multicast tree is presented.

The construction of static $SPLAST$ algorithm is given in algorithm 1. Notice that we use a slightly modified $LAST$ algorithm in step 1; we don't

iterate through all nodes of G, but rather from one described in [14, 19] only through the nodes of M.

Algorithm 1: *SPLAST* Construction

Input: *Graph $G(V, E)$, set of terminals $M \subseteq V$, source node $s \in V$ and two real numbers α and β such that $\alpha > 1$ and $\beta \geq 1 + 2/(\alpha - 1)$.*

Output: *SPLAST Graph $H'(\alpha, \beta)$, SPLAST Tree $T'(\alpha, \beta)$.*
Step 1: *Run an (α, β) - LAST algorithm on G (with our minor change). Produce graph $H(\alpha, \beta)$ and tree $T(\alpha, \beta)$.*
Step 2: *Find a spider decomposition D of M in $T(\alpha, \beta)$*
Step 3: Connect the spiders from D with the source s based on the paths of *T. Produce tree T'.*
Step 4: *Delete from graph H all the nodes that do not belong to R, where R is the set of nodes of T'. Produce Graph H'*

Theorem 2

For a given graph G, a multicast group M, root s, $\alpha \geq 1$ and $\beta \geq 1 + 2/(\alpha - 1)$ the algorithm correctly produces sub graph $H'(\alpha, \beta)$ and tree $T'(\alpha, \beta)$ such that: for node v, the distance between s and v in $T'(\alpha, \beta)$ and in $H'(\alpha, \beta)$ is at most α times the shortest distance from s to v in G; the total weight of $T'(\alpha, \beta)$ and the total weight of $H'(\alpha, \beta)$ is at most β times the weight of a minimum-spanning tree of G' where G' is the sub graph of G only with the nodes that belongs to R .

Proof

The *LAST* algorithm [14,19] and step 3 in algorithm 1 ensures that the desired α factor is achieved. Let T be any spanning tree of G with a root s and let $z_1, z_2, z_3 \ldots, z_k$ be any k nodes of T. From [19] it is known that

$$\sum_{i=1 \ldots k} d_T(z_{i-1}, z_i) \leq 2 \cdot c(T) \tag{1}$$

Let T' be the *MST* of H'. Every node that belongs to T'_m also belongs to minimum spanning tree T_m of G, so the path $p(z_{i-1}, z_i)$ between z_{i-1} to z_i in T'_m $(z_{i-1}, z_i \in T'_m)$ is the same in T_m as well. Theorem 2 [19] proves

that $\sum_{i=1...k}(\alpha-1)\cdot d(z_i) \leq d_{T_m}(z_{i-1},\, z_i) = d_{T'_m}(z_{i-1},\, z_i)$. Therefore, from *equation*1 it follows that $\sum_{1...k} d(z_i) \leq 2\cdot c(T'_m)/(\alpha-1)$ and the total cost of H' is $|H'| = |T'_m| + \sum_{1...k} d(z_i) \leq |T'_m|\cdot(1+2/(\alpha-1)$ From [14] we can see that if H' is $(\alpha,\, \beta)$ $LAST$ so T' is $(\alpha,\, \beta)$ $LAST$ too.

5 The Distributed Algorithm

Notice that *SPLAST* algorithm can be viewed as a combination of several algorithms for example *MST*, *SPT* and *DFS*. First, we shortly explain the distributed version of above algorithms. Then we show how to combine them into a unified distributed *SPLAST* algorithm.

5.1 Distributed LAST Algorithm

We begin our discussion with the distributed $LAST$ algorithm. At the first stage we use the algorithm proposed in [20, 21] that builds an *MST* (named T_m) in a distributed fashion (we assume that all nodes have unique ID's). Next we proceed to computing *SPT* by running an improved distributed Bellman-Ford algorithm that is presented in [22]. All these operations can be done in $O(n^2)$ time and messages in the worst case. In what follows, we use the distributed *DFS* algorithm given in [23, 24] in order to produce the *DFS* walk. We note that the output of each of the algorithms (*MST*, *SPT* and *DFS*) is produced in a distributed fashion, that is, each node knows its neighbors in the corresponding tree. In addition, after *MST* construction each node v knows its distance from the root $s, d_{T_m}(v,\, s)$, and after the SPT computation each node v knows its $d(v)$. This can be accomplished by a simple broadcast process from root s towards the leaves in the corresponding tree. In Algorithm 2 we present a pseudo code for distributed $LAST$ followed by additional explanations.

Algorithm 2 - Distributed $(\alpha,\, \beta)$ **Light Approximate Shortest Paths Tree**

At the beginning of algorithm 2 each node is aware of its distance from the root s in the *MST* T_m and the distance from s in the *SPT* T_{sp} We construct the required graph H when initially $H = T_m$. We perform a distributed *DFS* walk on H [23, 24]. When node v has been activated during this walk, v checks whether it belongs to the multicast group. If not, v sends its distance $d_H(v,\, s)$ to the next node on the walk. If node v is a member of the multicast group, v checks if $d_H(v,\, s) > \alpha\cdot d(v)$. If

Input: *Weighted graph $G(V, E)$, root s and $\alpha > 1, \beta \geq 1 + 2/(\alpha - 1)$.*
Output: *Graph $H(\alpha, \beta), T(\alpha, \beta)$-LAST.*
Step 1: *Find a $MST(G)$ T_m by employing distributed algorithm [20,21]; Find a $SPT(G)$ T_{sp} by employing distributed algorithm [22] with start node s;*
Step 2: *Find a preorder numbering of T_m using s as the start node by employing the distributed DFS algorithm [23,24];*
Step 3: $H = T_m$;
For each node $v \in M$ in the preorder sequence of T_m do
Find a shortest s - v path P in H;
If $c(P) > \alpha \cdot d(v)$
Then add all the edges in a shortest s - v path in G to H;
Update $d_H(v, s)$ and send to next node in the walk;
End; (if)
End; (for)
Step 4: *find a SPT of H with start node s by the employing distributed algorithm in [22]. Produce T.*

so, v sends a "*connect*" message to its parent u_1 in T_{sp}(shortest path tree) and adds the edge (v, u_1) to H. Node u_1 sends the same message to its parent u_2 in T_{sp} and adds the edge (u_1, u_2) in H. This process propagates upwards towards the root s and stops after the "*connect*" message reaches s. In parallel, v sends its new distance from the root in H to the next node in the DFS walk. Each node w upon receiving a message with the updated distance $d_H(v, s)$ from its neighbor, updates $d_H(w, s)$ if necessary. Note that every node may participate in the *DFS* walk several times.

5.2 Distributed Spider Algorithm

Here we explain the distributed spider decomposition algorithm. The algorithm works as follows: every leaf in the $LAST$-T initiates the process by sending to its parent in T a message *spider(k, prune)*, where k plays a role of an accumulator: initially $k = 1$ if node $v \in M$ or $k = 0$ otherwise, and *prune* is a boolean variable: initially $prune = 0$ if $v \in M$ or $prune = 1$ otherwise. When a node w receives messages from all its children in T it sums the total value of k values derived from the received messages plus its own k value,

obtaining k_w. If the sum k_w is equal or greater than 2, w declares itself as a center of a spider and sends a message *spider(0, 0)* to its parent u in T. Otherwise, it sends a message *spider*(k_w, $k_w \oplus 1$). Each node w removes the children that sent the message *spider(0, 1)* from its neighborhood list. If a node z receives a message *spider*(*1/0* , *0*) from one of its children it must propagate this message to its parent with '0' in the *prune* field. The algorithm terminates when the root receives messages from all its children and removes nodes if needed. After the algorithm terminates each center is connected to the root and only the relevant paths connecting terminals to the root s remain in the tree.

5.3 Distributed SPLAST Algorithm

The steps of the distributed *SPLAST* algorithm are based on the corresponding steps static *SPLAST* algorithm. Each step follows a distributed approach analogous to the corresponding centralized approach as presented in algorithm 1.

6 Mobile SPLAST Algorithm

Our main challenge is to maintain the resulting *SPLAST* under dynamic changes of the network. We assume that each node has a limited battery power, and therefore required increase in transmission power which cannot be supported by the battery correspond to edges failures. Similarly edges recoveries may be related to a decrease in the required transmission power which now can be supported by the remaining energy of the battery. Recoveries and failures of network edges may also happen due to obstructions and other phenomena related to physical properties of the radio channels. Weight modifications are functions of changes in the required transmission power that can be supported by the battery. In the following we describe how to deal with each one of these updates and how to adjust the underlying algorithms (i.e., *MST*, *DFS* and *SPT*) to fit mobile scenarios.

6.1 Maintaining the Minimum Spanning Tree

A primary requirement from a distributed *MST* algorithm for mobile networks is a quick respond to topological changes that are caused by failures, recoveries and weight modification of the edges. Without loss of generality, we assume that at any given time only one topological change may occur.

The solution presented in [25] is suitable for cases of edge failures and recoveries but not for modifications of edges' weights. We modify that algorithm in order to provide a comprehensive solution for this case as well. The idea is that each node periodically checks the weight of its adjacent edges. If the weight of any edge in the *MST* T_m is increasing or the weight of some non-tree edge is decreasing, both end point nodes of this edge will update their neighborhood list by deleting this edge with its old weight and follow the failure delete algorithm presented in [25]. Afterwards these nodes insert the delete edge with modified weight by updating their neighborhood list and follow the recovery algorithm of [25]. This technique provides a way to maintain a distributed mobile MST.

The algorithm uses $O(n + m)$ messages. In contrast, if one uses an algorithm (such as GHS [20]) to reconstruct the tree after every failure or recovery, the message complexity changes to $O(m + n \log n)$.

6.2 Maintaining the Shortest Path Tree

Changes in the network topology trigger the execution of the algorithm that produces an updated Short-Path-Tree where δ equals 3. In the proposed algorithm we utilize the observation, given in [26].

Observation 1:
When an edge $e(v, u)$ fails, a new *SPT* may be found by replacing the failed edge with a new edge of graph G, thus producing a new tree with the paths length up to 3 times bigger than the optimal.

6.2.1 Edge Failure

When an edge $e(v, u) \in T_{sp}$ fails, T_{sp} is split to two disjoint subtrees S_u and S_v such that, without loss of generality S_u contains root s, node u, and S_v contains node v where $d(u) \leq d(v)$. First, each node in the graph should be informed about the failed node and mark itself to the appropriate subtree. In order to deal with this case, node u sends to root s a $failed(e(u, v), up)$ message regarding the failed edge. When root s receives $failed(:, :)$ message it broadcasts to all nodes in S_u a $failed(e(u, v), down)$ message.

Each node in S_u that receives this message marks itself as part of S_u. Node v broadcasts to all nodes in S_v a $failed(e(v, u), up)$ message. Each node in S_v that receives this message marks itself as part of S_v tree and sends a *search* message on its outgoing non-*SPT* edges. Each node in S_v

that receives a *search* message replies with an *ignore* message on the same edge.

Each node w in S_u that receives a *search* message via edge $e(x, w)$, $x \in S_v$ sends back on the same edge a message *distance*(*value*), where *value* is equal to $d(w) + c(x, w)$ *such* that e(x, w) connects between S_u and S_v. If a leaf z in S_v tree receives *distance*(*value*) messages from all its non-*SPT* edges, it chooses the minimum *value*(*min_val*) and sends a *minimum_length(min_val, z)* to its parent in the S_v tree. Every node k in S_v that receives a *minimum_length(min_val, z)* messages from all its children in S_v and also receives a message *distance*(*value*) from all its non-*SPT* edges sends to its parent a message *minimum_length*(*min_val*, d) where *min_val* is the minimum between all *min_val* and *value* that node k receives and d is the node that sent this value. This process ends when node v receives *minimum_length(min_val, d)* messages from all its children in S_v and *distance*(*value*) messages from its non-*SPT* edges between S_u and S_v. Afterwards, node v chooses the minimum between all *min_val* and *value* that it receives and sends a *connect*(d) message to node d that corresponds to this minimum distance. When node d receives the *connect*(d) message from v it sends a *connect_trees* message to the node in S_u thus reconnecting the two trees S_u and S_v .Node d broadcasts the new distance from root to all nodes in S_v in order to provide a consistent update of distance from nodes of S_v to root s.

6.2.2 Edge Recovery

When an edge $e(u, v)$ recovers, a cycle appears in the *SPT* T_{sp}. The proposed LCA mechanism from [25] enables to find the heaviest edge $e(x, y)$ in the cycle in a distributed fashion. After we isolate the heaviest edge $e(x, y)$, we delete it from the tree and start the process of a failed edge as explained above. Notice that we are using the recovered edge $e(u, v)$ in the failed process as a non-*SPT* edge.

6.2.3 Edge Modification

In the case edge $e(u, v)$ changes its weight, notice that we determine the type of $e(u, v)$ before we perform any update. The changes we deal with are a non-*SPT* edge that decreased its weight or an edge in the *SPT* that changes its weight (notice that in all cases $d(u) \leq d(v)$). In all cases we follow the same procedure below. First we check whether the new edge improves the

path length from v to the root s. If not, this process is terminated otherwise, three cases may occur:

Case 1: A non-SPT $e(u, v)$ edge weight decreased. Node v sends a *search* message to node u to check whether the new weight improves its path length. Node u receives via $e(u, v)$ the *search* message and replies to v a *distance*(*value*) where $value = d(u)+c(u, v)$. If *value* improved $d(v)$, node v switches between the improved edge $e(u, v)$ and the edge that connected node $v$to its parent in the SPT.

Case 2: An edge weight increase in $e(u, v) \in SPT$. Node v deletes the edge $e(u, v)$ with the old weight and starts the failed edge process for this edge.

Case 3: An edge weight decrease in $e(u, v) \in SPT$. Node v broadcasts a message *new_dist*($d(v)$) to all its children in the SPT regarding the new distance from the root.

6.3 Maintaining the Depth-First-Search Tree

The main idea of this algorithm relies on the distributed approach of constructing the *DFS* tree [23].

6.3.1 Edge Failure

When edge $e(u, v)$ fails the original *DFS* tree is split to two disjoint subtrees S_u and S_v such that, without loss of generality S_u contains root s, node u, and S_v contains node v where in the *DFS* walk starting from s, u is reached before v. Each node in the graph should be informed about the failed node and mark itself to the appropriate subtree. This task can be accomplished by simple broadcast from each one of the roots of S_u and S_v where v is the root of S_v and knows about the failure and s is the root of S_u and is not aware of the failure. Therefore node u should inform s. Before s issues an instruction to u to start reconnecting S_u and S_v we perform two convergecast processes (one in S_u and one in S_v) in order to guarantee that each node in the converged subtrees has marked itself as belonging to the correct subtree. Afterward, root s starts the connection processes by choosing node w in S_u with the smallest id and then send it a *connect* message. When node w receives this message it tries to connect S_u with S_v with its outgoing non-*DFS* tree edges. In case of success this process terminates, otherwise, node w sends to the root an *unsuccessful* message. When root s receives this message it picks the next smallest node in S_u and send it a *connect* message.

This node in turn tries to connect in the same way as w did. This process continues until one of the nodes success to connect the two subtrees.

6.3.2 Edge Recovery

When edge $e(u, v)$ recovers from a failure, node u sends to its parent in the *DFS* tree a message *failed*(u). Node v sends to its parent in the *DFS* tree a *failed*(v) message. Each node w that receives one of these messages appends itself to the message and passes to its parent *failed*(v, w) or *failed*(u, w) accordingly. When the root s receives both messages it compares them. If they are not equal it means that u and v are not from the same branch of the *DFS* tree, therefore the root deletes the edge that connects the root to the first node in the v branch (the last node in v message). If some nodes in the two sequences are equal, it means that u and v are from the same branch of the *DFS* tree, therefore the root sends a *delete*(z) message where z is the first node that is not equal in the sequence (the first node in the sequence of u that is not contained in v sequence when searching from right to left). When z receives the *delete* message it deletes the edge that connects it to v. If node u receive a message from v, it means that u and v are from the same branch of the *DFS* tree and the path from v to root pass in u so the new edge does not change the *DFS* tree.

6.4 Maintaining SPLAST

Distributed SPLAST algorithm has 4 steps. The first two steps of the algorithm were explained in paragraphs 6.1 and 6.2. The rest of algorithm is essential for the maintenance of *SPLAST*, when the network experiences topology changes. Whenever the tree is updated, we deteriorate the α factor of the corresponding tree. Therefore, after a number of changes in the *SPT* we may need to reconstruct a new *SPT* from scratch.

7 Simulation Results

The theoretical bounds given before provide information on the worst case properties of the *SPLAST* tree and motivate its usefulness for multicast applications in wireless ad hoc networks. However, average properties are also important from practical networking point of view. The average properties of *SPLAST* trees were investigated with thorough simulation experiments. For many given graphs with various number of nodes and different topologies

we compared the efficiency of the *SPLAST* tree against the minimum spanning tree, the shortest path tree, the minimal Steiner tree and one heuristic of the Steiner tree constructed with the shortest path heuristic (*SPH*) [6] which is known to have good average results.

The α parameter measures the effectiveness of each tree with relation to the shortest path of each multicast destination to the source, while the β parameter measures the effectiveness with relation to the total tree cost. As was mentioned previously, the *SPLAST* algorithm tries to balance between these two requirements.

Simulation details are as follows: each point on each graph represents the average parameter of several experiments on random graphs with fixed number of nodes. The total number of nodes varied from 4 to 20 and the size of multicast group is one half of the total number of nodes.

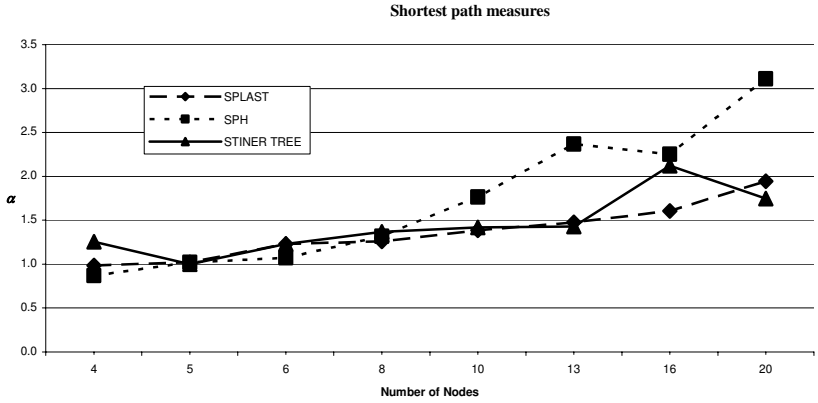

Figure 1: Comparison between the shortest distance properties of the *SPLAST*, *SMT* and *SPH* trees.

Figure 1 show the properties of the α parameter. In order to compare the results for different number of nodes, the distance from the source to each multicast destination was normalized by the shortest distance obtained from the SPT. The SPLAST tree shows good performance compared with the *SMT* and the *SPH* trees.

Figure 2 shows the properties of the β parameter. In order to compare the results for different number of nodes, the distance from the source to each multicast destination was normalized by the total weight of the MST. Again the SPLAST tree shows good performance on the average compared against the *SMT* and the SPH trees.

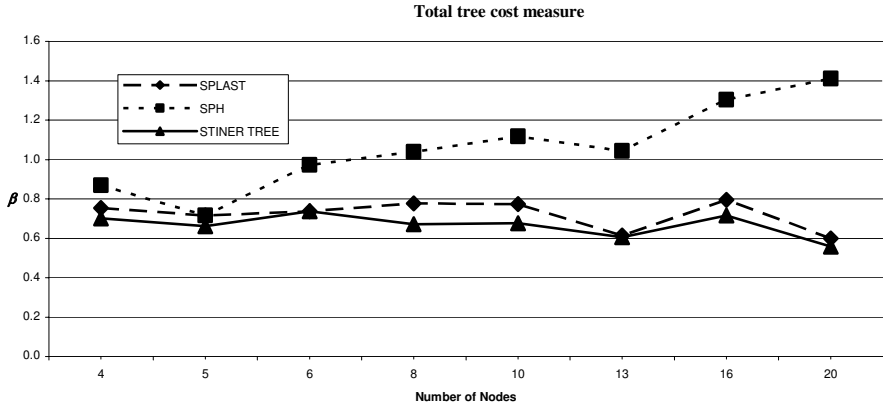

Figure 2: Comparison between the total tree cost property of the SPLAST, SMT and SPH trees.

8 Conclusion

In this paper we presented a novel approach for the construction of multicast tree in wireless ad hoc network – the *SPLAST* algorithm which combines the *LAST* algorithm and spider decomposition. *SPLAST* trees are proved to keep all the properties of the *LAST* trees expressed by the α and β parameters while offering complete solution for multicast applications in wireless ad hoc networks.

The analysis of *SPLAST* algorithm started from a discussion of its underlying components: the *LAST* algorithms, shortest path and minimal spanning trees, and spider decomposition.
A centralized static solution was presented and later extended to a distributed implementation, which is crucial for communication networks. *SPLAST* applicability to wireless ad hoc networks must be tested with various conditions which are unique for this type of networks: rapid topology changes caused by edges failures, edges recovery, and changes to edges weights. We developed a detailed solution to wireless ad hoc scenarios, and analyzed the overhead of maintaining it various components.

Supported by good worst case theoretical bounds and good average results explored by thorough simulations, the *SPLAST* algorithm suggests a good balance between the minimum spanning tree and shortest path tree with low complexity in computation time and messages. This suggests that *SPLAST* trees may be used as an uniform algorithm for both unicast and

multicast routing in wireless ad hoc networks.

REFERENCES

1. Z.J. Haas, J. Deng, P. Papadimitratos, S. Sajama. Wireless ad hoc networks, appears in *Wiley Encyclopedia of Telecommunications*, John G. Proakis, Ed. Wiley, 2002.

2. S. Deering, *Scalable Multicast Routing Protocol*, Ph.D. dissertation, Stanford University, 1989.

3. B. Awerbuch, Y. Azar. Competitive multicast routing, *Wireless Networks*, 1995; 1 :107-114.

4. J. Nonnenmacher, M. Lacher, M. Jung, G. Carl, Biersack EW. How Bad is Reliable Multicast Without Local Recovery?, *Proc. IEEE Infocomm 98*, San Francisco, CA, April 1998; 972-979.

5. S. Paul, K. Sabnani, J. Lin, S. Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE J. Selected Area Comm.*, 1997; **15**(3): 407-421.

6. F.K. Hwang, D.S. Richards, P. Winter. *The Steiner tree problem*, North-Holland, 1992; 93-202.

7. M.R. Garey, R.L Graham, D.S. Johnson., The complexity of computing Steiner minimal trees, *SIAM J. Appl. Math.*, 1977; 32: 835-859.

8. P. Winter. Steiner Problem in Networks: A Survey, *Networks*, 1987, **17**(2), 129-167.

9. F. Hwang, D. Richards. Steiner Tree Problems, *Networks*, 1992; **22**(1); 55-89.

10. R. Ravi. *Steiner trees and beyond: Approximation algorithms for network design*, Ph.D. Dissertation, Brown University, 1993.

11. G. Robins, A. Zelikovsky. Improved Steiner tree approximation in graphs, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2000; 770-779.

12. R. Bellman. *Dynamic Programming*, Princeton University Press, 1957.

13. J. Dossey, A. Otto, L. Spence, C. Eynden. *Discrete Mathematics*, Harper Collins College Publishers, 1993.

14. S. Khuller, B. Raghavachari, N. Young. Balancing minimum spanning and shortest path trees, *Algorithmica*, 1994; **120**(4): 305-321.

15. P.N. Klein, R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner tree, *Algorithms*, 1995; **19**; 104-115.

16. J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Am. Math. Soc.*, 1956; **7**: 48-50.

17. G. Kortsarz, D. Peleg. Approximating shallow-light trees, *Proceedings of the 8th Symposium on Discrete Algorithms*, 1997; 103-110.

18. B.Y. Wu, K.M. Chao, C.Y. Tang. Light graphs with small routing cost, *Networks* 2002; **39**(3): 130-138.

19. J. Cheriyan, R. Ravi. Lecture Notes on Approximation Algorithms for Network Problems, http://www.math.uwaterloo.ca/∼jcheriya/PS_files/ln-master.ps, 1998; 78-83.

20. R.G. Gallager, P.A. Humblet, P.M. Spira. A Distributed Algorithm for Minimum Weight Spanning Trees, *ACM Trans. on Program. Lang. and Systems*, 1983; **5**(1); 66-77.

21. P.K. Mohapatra, Fully Sequential and Distributed Dynamic Algorithms for Minimum Spanning Trees, http://www.arxiv.org/PS_cache/cs/pdf/0002/00020 2000.

22. L. Brim, I. Cerna, P. Krcal, R. Pelanek. Distributed shortest path for directed graphs with negative edge lengths, *Technical Report FIMU-RS-2001-01*, Faculty of Informatics, Masaryk University Brno, 2001.

23. M.B. Sharma, N.K. Mandyam, S.S. Iyangar. An optimal distributed depth-first-search algorithm, *Proceedings of the seventeenth annual ACM conference on Computer science : Computing trends in the 1990's*,1989; 287-294.

24. D. Kumar, S.S. Iyengar, M.B. Sharma. Corrections to a Distributed Depth-First Search Algorithm. *Inform. Process. Lett.*, 1990: **32**(4); 183-186.

25. C. Cheng, I.A. Cimet, S.P.R. Kumar. Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology, *Computer Communications Review*, 1988: **18**(4); 330-338.

26. E. Nardelli, E. Proietti, P. Widmayer. Swapping a Failing Edge of a Single Source Shortest Paths Tree Is Good and Fast, *Algorithmica*, 2003: **35**(1); 56-74.