JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 20, 425-447 (2004)

Extending the Lifetime of Multicast Trees in WANETs*

BIN WANG AND SANDEEP K. S. GUPTA
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, U.S.A.
E-mail: {Bin.Wang, Sandeep.Gupta}@asu.edu

This paper presents a distributed algorithm called L-REMiT for enhancing the lifetime of a source-based multicast tree in wireless ad hoc networks (WANET). The lifetime of a multicast tree is the duration from the formation of the tree to the time when the first node fails due to battery energy exhaustion. L-REMiT assumes that the energy consumed to forward a packet is proportional to the forwarding distance and channel characteristics, and that WANET nodes can dynamically adjust their transmission power. The task of extending the lifetime of a multicast tree is formulated as the task of extending the lifetime of bottleneck nodes in the tree. The number of multicast packets that a bottleneck node can forward, as determined by its residual battery energy and the distance of its farthest child node, is the minimum over all the nodes in the multicast tree. The lifetime of a bottleneck node is improved by reassigning its farthest children to other nodes in the tree with the goal of improving the lifetime of the multicast tree. Nodes only require information from their neighbors for refining the tree in a distributed manner. Simulation results show that L-REMiT has low overhead and performs better than the BIP/MIP and EWMA algorithms.

Keywords: ad hoc network, wireless network, multicast tree, broadcast tree, distributed algorithm, tree lifetime

1. INTRODUCTION

Multicasting enables a single node in a network to communicate efficiently with multiple nodes in the network. A multicast service is needed for many distributed applications such as distributed resource allocation and replicated file systems. In wireless ad hoc networks (WANETs), including sensor networks, all the nodes in the network co-operate to provide networking services to various distributed tasks. In such networks, nodes are usually powered by a limited source of energy. As opposed to a wired network, the availability of limited energy at the nodes of a WANET has an impact on the design of multicast protocols. Specifically, the set of network links and their capacities in WANETs is not pre-determined but depends on factors such as distance between nodes, transmission power, hardware implementation and environmental noise. This is one of the basic differences between wireless and wired networks.

The energy-efficient broadcasting/multicasting tree problem was presented in [2]. Wieselthier et al [2] proposed a "node-based" elastic model for wireless multicast and the

Received October 15, 2003; accepted November 15, 2003.
Communicated by Ten-Hwang Lai, P. Sadayappan, Yu-Chee Tseng and Yi-Bing Lin.
* The preliminary version of this paper is our ICPP-03 paper [1].

concept of the *wireless multicast advantage*. The problem of constructing the optimal energy-efficient broadcast/multicast tree is NP-complete [3], several heuristic algorithms for constructing source-based energy-efficient broadcast/multicast trees have been developed. Wieselthier et al. have proposed two centralized algorithms, BIP/MIP [2] and two distributed versions of the BIP algorithm, Dist-BIP-A and Dist-BIP-G [4], to build source-based broadcast/multicast trees. These two distributed algorithms have slightly worse performance than their centralized counterparts. Cagalj et al. [5] presented an Embedded Wireless Multicast Advantage (EWMA) algorithm to reduce the energy consumption of a source-based broadcast trees. They also described a distributed version of the EWMA algorithm. Wang and Gupta have presented two distributed algorithms – G-REMiT [6] and S-REMiT [7] to minimize energy consumption for group-shared and source-based multicast tree, respectively.

As all of the multicast data traffic must go through the intermediate nodes, the above algorithms can result in rapid depletion of energy at intermediate nodes, possibly leading to network partitioned and interruption of the multicast service. New approaches are therefore needed to extend the network lifetime. The problem of maximizing the network lifetime was studied in [8]. Both Wieselthier et al. [9] and Kang et al. [8] extended BIP/MIP by using residual battery energy in their energy metric to extend the broadcast/multicast tree lifetime. Throughout this paper, we will use MIP and L-MIP to denote MIP without the consideration of residual battery energy and MIP with the consideration of residual battery energy, respectively.

In this paper, we will focus on source initiated multicasting of data in WANETs. Our main objective is to extend the lifetime of a *source-based multicast tree*. A source-based multicast tree is rooted at a multicast source node and covers all the other multicast group members who are receivers. We define the **lifetime** of a multicast tree as the time duration starting from the beginning of multicast service and continuing until the first node in the multicast tree fails due to battery energy exhaustion. We will propose a distributed protocol called L-REMiT that is part of a suite of protocols called REMiT (Refining Energy efficiency of Multicast Trees), which we are designing to achieve various energy-efficiency goals related to multicasting in WANETs. REMiT protocols are distributed protocols which refine the energy-efficiency of a pre-existing multicast tree using local knowledge at each node. The REMiT protocols can be categorized as energy-metric dimension (minimizing energy-consumption or maximizing the lifetime) or multicast-tree type dimension (source based or group-shared trees) protocols. For example, we earlier presented G-REMiT [9] and S-REMiT [10], which minimize energy-consumption for group-shared trees and source-based trees, respectively. In this paper, we present L-REMiT, that uses a *minimum-weight spanning tree* (MST) as the initial tree and improves its lifetime by switching (reassigning) children of a "bottleneck" node to another node in the tree. A bottleneck node is one that currently has the minimum energy level among all the multicast tree nodes. Each such switching step is called a "refinement." A multicast tree is obtained from the "refined" MST (after all possible refinements have been performed) by pruning the tree to reach only multicast group nodes. The L-REMiT algorithm is distributed in the sense that each node has only got local view of the tree and can independently switch its parent as long as the multicast tree remains connected. Our simulation results show that L-REMiT outperforms the most prominent proposals in the literature: BIP/MIP and EWMA.

The rest of the paper is organized as follows. Section 2 describes the system model and some notations used with L-REMiT. The problem of maximizing the multicast tree lifetime is described in section 3. Section 4 describes the L-REMiT algorithm. Section 5 presents simulation results obtained using L-REMiT. We discuss how to recover L-REMiT tokens when communication is unreliable and other issues in section 6. We give some concluding remarks in section 7.

2. SYSTEM MODEL AND ASSUMPTIONS

We assume that each node in a WANET with N nodes has a unique identifier i, $1 \le i \le N$. Each node has only local view of the network and determines the distance between itself and its neighbor nodes by using some distance estimation method [11]. The connectivity in the network depends on the transmission power of the nodes. Each node can dynamically change its transmission power level. A node may use a different power level for each multicast tree in which it participates. For simplicity, we assume that all data packets are of the same size. Let $E_{i,j}$ be the minimum energy needed for a link between nodes i and j for a data packet transmission. We assume the following model for $E_{i,j}$ [12]:

$$E_{i,j} = E_{elec} + K(r_{i,j})^{\alpha}, \tag{1}$$

where $r_{i,j}$ is the Euclidean distance between nodes i and j[1], E_{elec} is a distant-independent constant that accounts for the overheads of electronics and digital processing, K is a constant dependent upon the properties of the antenna and α, called the propagation loss exponent, is a constant that is dependent on the propagation losses in the medium. For **long range radios**, $E_{elec} << K(r_{i,j})^{\alpha}$, so $E_{i,j} \approx K(r_{i,j})^{\alpha}$. On the other hand, for **short range radios**, E_{elec} is not negligible since it can substantially exceed the maximum value of $K(r_{i,j})^{\alpha}$ [12].

Compared to wired networks, WANETs have a "wireless multicast Advantage" [2], which means that all nodes within communication range of a transmitting node can receive a multicast message with only one transmission if they all use omni-directional antennas. We assume the same capability in our model. Further, every node (say node i) has two coverage areas, **Control coveRage area** (CR_i) and **Data coveRage area** (DR_i), such that $DR_i \subseteq CR_i$. These coverage areas depend upon the transmission power selected by node i to transmit its control and data packets, respectively. For example, in Fig. 1, the radius of CR_{10} is 3.2; i.e., node 10's control message may reach nodes 6, 7, and 9, but if $DR_{10} = 2.75$, its data message may only reach only nodes 6 and 9, not node 7.

Neighbors of node i are the nodes within CR_i. We use V_i, $V_i \subseteq CR_i$, to denote the set of **tree neighbors** of node i, i.e., those neighbors of node i which also belong to the multicast tree T. A **connected tree neighbor** j of a node i is a tree neighbor of node i

[1] The distance-based model can be easily extended to the energy model, which only considers the actual energy cost (per packet) between nodes i and j. For example, the mechanism can be simply implemented by local broadcast handshake messages at node i using different energy level. And every node j receives the handshake message responses to node i. This will make our system model independent of any particular propagation model increasing the applicability of L-REMiT algorithm.

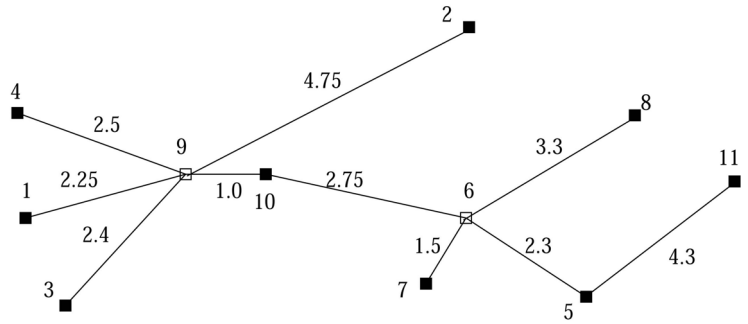

Fig. 1. A multicast tree. (Only branches are shown for clarity and because L-REMiT ignores other links. Branch labels denote the Euclidean distance between their endpoints.)

which is connected to the node by a *branch*, i.e., link $(i, j) \in T$. A **non-connected tree neighbor** j of a node i is a tree neighbor of node i which is connected to node i by more than one branch in T; i.e., the length of the unique path between i and j in T is greater than 1. We denote the set of connected and non-connected tree neighbors of node i as CTN_i and $NCTN_i$, respectively. Note that $NCTN_i = V_i - CTN_i$.

We assume that node s is the source node of the multicast tree whose lifetime is being maximized. A message to be multicasted to the group members is forwarded along the branches starting from source node s: every node on the tree which receives a new multicast message from its parent node forwards the message to all of its children nodes. Fig. 1 shows a source-based multicast tree example, where node 10 is the source node, nodes 1, 2, 3, 4, 5, 7, 8, 10 and 11 are multicast group members, and nodes 6 and 9 are non-group nodes that serve as forwarding nodes in the multicast tree.

3. PROBLEM DEFINITION

In this section, we will define the problem of maximizing the lifetime of a multicast tree. Before we do so, we will define the notions of the energy cost of a node in a multicast tree and the lifetime of a multicast tree.

3.1 Energy Cost of a Node in a Multicast Tree

In a source-based multicast tree, the energy consumption at every tree node is determined by the distance of the children nodes. For example, consider node 10's source-based multicast tree shown in Fig. 1. Node 10 will send each multicast message along the branch to nodes 6 and 9. Node 9 will forward them to nodes 1, 2, 3 and 4. Similarly, node 6 will forward them to nodes 5, 7 and 8, and so on. The energy consumed at node 9 for each multicast message from node 10, using the source-based multicast tree shown in Fig. 1, is $max(E_{9,1}, E_{9,2}, E_{9,3}, E_{9,4}) = E_{9,2}$.

Let d_i be i's maximum length between i and it's farthest children. The **energy cost** of node i in a multicast tree T, $E(T, i)$ is:

$$E(T,i) = \begin{cases} E_{elec} + Kd_i^{\alpha} & \text{if } i \text{ is the source node;} \\ E_{elec} + Kd_i^{\alpha} + E_{recv} & \text{if } i \text{ is neither the source nor a leaf node in } T; \\ E_{recv} & \text{if } i \text{ is a leaf node in } T, \end{cases} \quad (2)$$

where E_{recv} denotes the energy cost of receiving a data packet. We assume that E_{recv} is the same for every node. We use function f to denote the r.h.s. of Eq. (2), i.e., $E(T, i) = f(d_i, E_{elec}, E_{recv})$.

3.2 Multicast Lifetime Metric

The lifetime of a node in a multicast tree, given its current battery energy level, is *the maximum number of multicast packets that can be transmitted by the node*, assuming that the node does not participate in any other packet transmission. If the residual battery energy at node i is R_i, the maximum number of packets that node i can transmit is $R_i/E(T, i)$. Hence, for a node i in tree T, we define the node i's **multicast lifetime** as

$$LT(T,i) = \frac{R_i}{E(T,i)}. \quad (3)$$

The lifetime of a multicast tree is *the maximum number of packets that can be transmitted over the multicast tree*, assuming that all the nodes belonging to the multicast tree do not participate in any other packet transmissions. Thus, the lifetime of a multicast tree T is the minimum lifetime of any node in T:

$$LT(T) = \min_{\forall i \in T}\{LT(T,i)\} = \min_{\forall i \in T}\frac{R_i}{E(T,i)}. \quad (4)$$

We call the node with minimum multicast lifetime in a multicast tree to be its **bottleneck node**.

Therefore, the *problem of maximizing the lifetime of a multicast tree* becomes the problem of maximizing the lifetime of the tree's bottleneck node. This problem is similar to the "load balancing" problem where tasks need to be sent to one of the many servers available so that the response time is minimized; – this is known to be an NP-complete problem [13].

4. L-REMIT ALGORITHM

L-REMiT is a heuristic algorithm, which tries to improve the lifetime of a bottleneck node in the initial multicast tree by changing the bottleneck node's children node so that the tree's lifetime is higher. It uses MST as the initial tree since MST is useful for various purposes, such as broadcasting. Further, MST performs quite well when applied to our problem based on our experimental results.

4.1 Refinement Criterion

We call the difference of the multicast lifetime of the trees before and after branch exchange the lifetime *Gain*. Formally, the lifetime gain for the entire tree is:

$$Gain = LT(T') - LT(T),$$

where T is the initial tree and T' is the refined tree. A positive gain for a multicast tree indicates that the lifetime of that tree has increased. In our heuristic, the notion of the lifetime *Gain* is used as the criterion for changing the parent of a node; this refinement is performed only if it is expected that $Gain > 0$.

We use the operator $Change_i^{x,j}$ to refer to the refinement step in which (bottleneck) node x's child node i switches its parent from node x to node j. Let T be a multicast tree, and let T' be the resulting graph after refinement $Change_i^{x,j}$ is applied to T. Formally, if T and T' are the trees before and after performing $Change_i^{x,j}$, respectively, then

$$T' = T - (i, x) + (i, j).$$

Note that T' may not necessarily be a tree, and that the lifetime of T' may not necessarily be larger than that of T. Hence, following are two basic problems that need to be solved for our multicast tree refinement approach to work:

1) determining whether $Change_i^{x,j}$ will result in another multicast tree;
2) ensuring that $Change_i^{x,j}$ will result in a positive lifetime *Gain*.

The first problem involves identifying those non-connected tree neighbors of node i which will be valid candidates for node i's new parent node. The second problem involves taking into account the impact of the $Change_i^{x,j}$ operation on the lifetime of the initial multicast tree and determining whether there will be an overall positive lifetime *Gain* as a result of $Change_i^{x,j}$.

Let $A_{i(x)}$ and $B_{i(x)}$ denote the subtree created as a result of deletion of the branch between node i and its parent node x from the multicast tree. Further, we assume that node i is in subtree $A_{i(x)}$.

Lemma 1 The tree remains connected after $Change_i^{x,j}$ if and only if node j is in subtree $B_{i(x)}$.

Proof: If node j is in $B_{i(x)}$ then the addition of branch (i, j) will reconnect subtrees $A_{i(x)}$ and $B_{i(x)}$ resulting in a new subtree. Conversely, if node j is in subtree $A_{i(x)}$, then adding (i, j) will simply result in a cycle in subtree $A_{i(x)}$ without reconnecting the two subtrees. ❑

We have the following important corollary to Lemma 1.

Corollary 1 The tree remains connected after $Change_i^{x,j}$ if and only if node j is not a descendant of node i.

Corollary 1 is useful in identifying non-connected tree neighbors of node i, any of which could serve as i's new parent node.

Lemma 2 Nodes j and x are the only nodes in the tree whose multicast lifetimes may be affected by $Change_i^{x,j}$.

Proof: Let k be a node which is neither node j nor x. The links between node k and its children will not be affected by $Change_i^{x,j}$. Then d_k is the same before and after $Change_i^{x,j}$. Based on Eq. (2), $E(T, k) = E(T', k)$. Further, based on Eq. (3), $LT(T, k) = LT(T', k)$. ❑

Corollary 2 If node x is the bottleneck node in T, then the lifetime *Gain* after $Change_i^{x,j}$ is

$$Gain_i^{x,j} = LT(T') - LT(T) = \min_{\forall i \in T'} \{LT(T', i)\} - LT(T, x).$$

4.2 Performing Refinement at a Bottleneck Node

Following are the steps involved in a tree refinement, $Change_i^{x,j}$. First, find the bottleneck node, say node x. Second, identify the farthest child of node x, say node i (based on Eqs. (2) and (3), the lifetime of node x is determined by its farthest child). Third, compute the set S_i, which is a subset of $NCTN_i$, such that $S_i = \{k \mid k \in NCTN_i \cap k \notin \text{subtree of } i\}$. Selection of the new parent of node i from among nodes in S_i guarantees that no cycle will be formed or, equivalently, that the tree will not be fragmented as a result of $Change_i^{x,j}$. Fourth, select a node j from set S_i with the highest positive estimated *Gain* ($\widehat{Gain_i^{x,j}} = \min\{LT(T', i), LT(T', x), LT(T', j)\} - LT(T, x)$); and, finally, switch node i parent to be node j instead of node x.

For example, consider the multicast tree shown in Fig. 1, which is node 10's source-based multicast tree. If node 9 is the bottleneck node in this tree, we will show how node 9's children can change their parent to increase the lifetime of the multicast tree. In this example, we will consider how node 2 decides to change its parent from node 9 to node 6. We will refer to this change event as $Change_2^{9,6}$. To simplify the following explanation, we will assume that $K = 1$, $\alpha = 2$, $E_{elec} = 0$, $E_{recv} = 1$, and $R_i = 100$ units, $i = 1, 2, \ldots, 11$. Using Eq. (3), node 2 will estimate the change in the lifetime at nodes 2, 9 and 6 if it makes $Change_2^{9,6}$. We will use T and T' to denote the multicast tree before and after $Change_2^{9,6}$. First, node 2 will estimate the current lifetime at nodes 2, 6 and 9: $LT(T, 2) = 100/1 = 100$; $LT(T, 6) = 100/(r_{6,8}^2 + 1) = 8.41$; $LT(T, 9) = 100/(r_{9,2}^2 + 1) = 4.24$. Similarly, node 2 can estimate the new lifetime at nodes 2, 9, and 6 after $Change_2^{9,6}$: $LT(T', 2) = 100/1 = 100$; $LT(T', 6) = 100/(r_{6,2}^2 + 1) = 7.16$; $LT(T', 9) = 100/(r_{9,3}^2 + 1) = 6.17$. After $Change_2^{9,6}$, the *estimated Gain* ($\widehat{Gain_2^{9,6}}$) of the tree obtained by switching at node 2 from node 9 to node 6 is

$$\widehat{Gain_2^{9,6}} = \min\{LT(T', 2), LT(T', 9), LT(T', 6)\} - LT(T, 9) = 6.17 - 4.24 = 1.93.$$

Likewise, node 2 can compute the estimated *Gain* in lifetime if it switches to node 10 and node 8: $\widehat{Gain_2^{9,10}} = 1.64$ and $\widehat{Gain_2^{9,8}} = 2.92$, respectively.

After comparing the gains, node 2 selects the node with the highest positive lifetime gain as the new parent. Thus, node 8 is selected as the new parent of node 2. Node 2 selects node 8 as its parent node in the multicast tree and disconnects from node 9. Thus in Fig. 1, the branch between nodes 2 and 9 is deleted, and the branch between nodes 2 and 8 is added to the multicast tree. Because DR_9 does not need to cover node 2 any more, the radius of DR_9 decreases to $r_{9,4}$. DR_8 should be changed to cover node 2; hence, the radius of DR_8 will increase from 0 to $r_{8,2}$.

A node uses the locally computable *estimated Gain* (instead of *Gain*) as a criterion for tree refinement. In general, the estimated gain $\widehat{Gain_i^{x,j}} = \min\{LT(T', i), LT(T', x), LT(T', j)\} - LT(T, x)$. Note that $\widehat{Gain_i^{x,j}}$ may not be equal to $Gain_i^{x,j}$ since there may be multiple bottleneck nodes in T or some other node's lifetime smaller than nodes i, j and x's new multicast lifetime in T'. This means that after one bottleneck node refinement, L-REMiT needs to find a new bottleneck node in tree T', and the new bottleneck node may not be nodes i, j or x. We call one bottleneck node refinement a round of the L-REMiT algorithm. In each round, a Depth-First Search (DFS) algorithm is used to pass an L-REMiT token to the nodes one by one. The L-REMiT token is used to exclude parallel single refinements at two nodes so that the information at all of the nodes is consistent. In section 4.4, we will discuss how to pass the L-REMiT token and how to select a new bottleneck node after a round. L-REMiT performs refinement steps until the estimated gain from the refinements continues to be positive.

4.3 Local Data Structure and Messages Types

Before describing a node's local data structure and the message types used by our distributed protocol, we will introduce the following notations. Let d'_i be the second maximum length of link between i and its children. We denote the triplet (d_i, d'_i, R_i) as l_i. Further, let node j be a neighbor of i, $j \in V_i$. We will use the notation $Data_k$ to denote the data associated with node k:

- $LT(T, k)$: this is the multicast lifetime of k in the tree T.
- B_k: a list of bottleneck nodes in k's subtree (there may exist several tree nodes with the same minimum multicast lifetime); also we use b_k to denote one of the nodes in B_k. Using B_k, we can extend the lifetime of a multicast tree with multiple bottleneck nodes.
- $CTNT_k$: this is a list of records of the type (i, l_i, b_i), and $\forall i$ is k's child.
- $NCTNT_k$: this is a list of records of the type (i, l_i), $\forall i \in NCTN_k$.

L-REMiT uses the following message types:

- *TOKEN*$(x, i, b_k, LT(T', b_k), flag)$: this is sent to the bottleneck node x and returned to the source node s along the tree branches. *flag* is a boolean value used to show whether the refinement was successful or not. This message is important and is used throughout the second phase of L-REMiT, so it needs reliable passing between nodes.
- *JOIN_REQ*(i, j): this is sent by node i to node j, requesting that j become its parent. This message is used in Step II.4 by node i to make $Change_i^{x,j}$.

- *JOIN_REP*(i, j): this is sent by j to reply node i's *JOIN_REQ*(i, j). This message is used in Step II.4 by node j to make $Change_i^{x,j}$.
- *LEAVE*(i, x): this is sent by node i to leave parent node x. This message is used in Step II.4 by node i to make $Change_i^{x,j}$ and in Step II.7 by node i to leave the tree when i is a leaf node and non-group node.
- *ELECTION_REQ*(s, i): this is sent by node s to node i, requesting election from node i. This message is used in Step I.2 by node s to request all the leaf nodes for bottleneck node election. This message is also used in Step II.6 by node s for bottleneck node election.
- *ELECTION*(b_i, $LT(T', b_i)$): this is the bottleneck node election result sent by node i to its parent node. This message is used in Step I.2 by node i to submit its subtree's election result.
- *NEIGHBOR_UPDATE*(l_i): this is sent by node i to nodes in V_i notifying new l_i after refinement. This message is used in Step II.4.

4.4 Distributed Protocol

L-REMiT consists of two phases: 1) multicast tree construction and 2) lifetime refinement. The **first phase** includes the following two steps:

I.1 Building initial multicast tree: All nodes run a distributed algorithm proposed by Gallager et al. [14] to build a MST in the wireless network, which is used as the initial multicast tree T. We require that after building T, each node in the multicast tree knows its parent and children nodes with respect to the source node s. Further, each node i, $i \in T$, has all local information l_k ($\forall k \in V_i$).

I.2 Bottleneck node election: The source node s requests all of the nodes in T to elect the minimum multicast lifetime node in a bottom-up manner from leaf nodes to the source node s. If i is a node on the tree, i needs to first find b_i, a bottleneck node in i's subtree. Then node i informs its parent node of the tuple (b_i, $LT(T, b_i)$) as the election result of node i's subtree. If node i is a leaf node of T, then $b_i = i$. An intermediate tree node delays the computation until it obtains the election results from all its children, before sending its election result to its parent node. Also, node i records each of its child subtree's bottleneck node information in $Data_i$. This information is used in Step II.2 for selection of the new bottleneck node after a refinement. Note that B_i obtained by this election may not include all of the bottleneck nodes in node i's subtree. But B_i obtained in the election is good enough for L-REMiT to proceed and does not affect the results of the L-REMiT algorithm.

The **second phase** proceeds in rounds coordinated by the source node s. Based on the bottleneck node election results, node s selects a bottleneck node x from B_s. Node s passes the L-REMiT token to node x, and then node x lets its farthest child, say node i, switch its parent to increase x's multicast lifetime. We use node j to denote the new parent of node i. After refinement $Change_i^{x,j}$, x passes the token back to node s along the tree path from node x to node s. Similar to Step I.2, node k (k is an ancestor of x) will re-elect bottleneck node b_k of Tree T' (T' is the multicast tree after $Change_i^{x,j}$) when

node k is forwarding the token which sent by x. After node s gets back the token, it requests node i to re-elect the new bottleneck node b_s from node i in a bottom-up manner in Tree T'. After $Change_i^{x,j}$, b_x and b_j may be changed in Tree T', and the bottleneck node needs to be re-elected at all of node x and i's ancestors. So that node s can locate a new b_s in Tree T' for the next round refinement. Node s terminates the L-REMiT algorithm when no lifetime can be gained in the current refinement step.

Following are the steps used to improve the lifetime of the multicast tree in the second phase (see Fig. 2 for illustrations of these steps):

II.1 Bottleneck node selection: Node s selects a node x from B_s.

II.2 Token passing and farthest children selection: Node s gives $TOKEN(x, -, -, -, false)$ to node x. Node x selects a node i from the list of x's farthest children list (x may have several farthest children nodes at the same time). If node i does not exist, x returns $TOKEN(x, -, -, -, false)$ to node s along the tree path from x to s and goes to Step II.6; otherwise, node x forwards $TOKEN(x, i, -, -, false)$ to node i and goes to the next step.

II.3 New parent selection: Once node i gets the L-REMiT token, and i selects a new parent node j which is not node i's descendant node[2] with the highest positive estimated $Gain$: $\widehat{Gain_i^{x,j}} = \min\{LT(T', i), LT(T', x), LT(T', j)\} - LT(T, x)$. If there is no such node j available, node i constructs the token as $TOKEN(x, -, -, -, false)$ and then goes to Step II.5; otherwise, goes it to the next step.

II.4 Make $Change_i^{x,j}$ and V_x, V_j notification: Node i makes $Change_i^{x,j}$ and notifies nodes in V_i. Node i constructs the token as $TOKEN(x, i, -, -, true)$. Based on *Lemma* 2, only l_j and l_x may be changed in the new multicast tree. So nodes j and x update V_j and V_x using *NEIGHBOR_UPDATE* message for new l_j and l_x.

II.5 Return token to x: Node i returns the token to node x. Node x updates B_x and returns $TOKEN(x, i, x, LT(T', x), true)$ or $TOKEN(x, -, -, -, false)$ to node s along the tree path from x to node s and then goes to the next step.

II.6 Update bottleneck node list at s: Node s gets back the L-REMiT token. If $flag$ = $false$, then it goes to the next step; otherwise, node s requests that i do bottleneck election which is similar to Step I.2. After node s gets back the election results, it updates B_s and goes to Step II.1.

II.7 Pruning the Tree: Node s requests that all of the tree node prune the redundant transmissions that do not need to reach the members of the multicast group from the tree. Then, node s terminates L-REMiT protocol.

Following are three examples to illustrate the second phase of the L-REMiT algorithm: 1) bottleneck node election; 2) single refinement at a node; and 3) the process flow of refinements in all of the rounds in the multicast tree. In these three examples, we use the multicast tree shown in Fig. 1.

[2] Following is the scheme used to find out whether node j is node i's descendant node in a distributed manner. Node i requests node j to send message to node j's ancestors. If node i gets the message, then i knows that node j is one of its descendant node. Otherwise, the message will finally come to node s, which it will forward to node i. − informing it that node j is its descendant node.

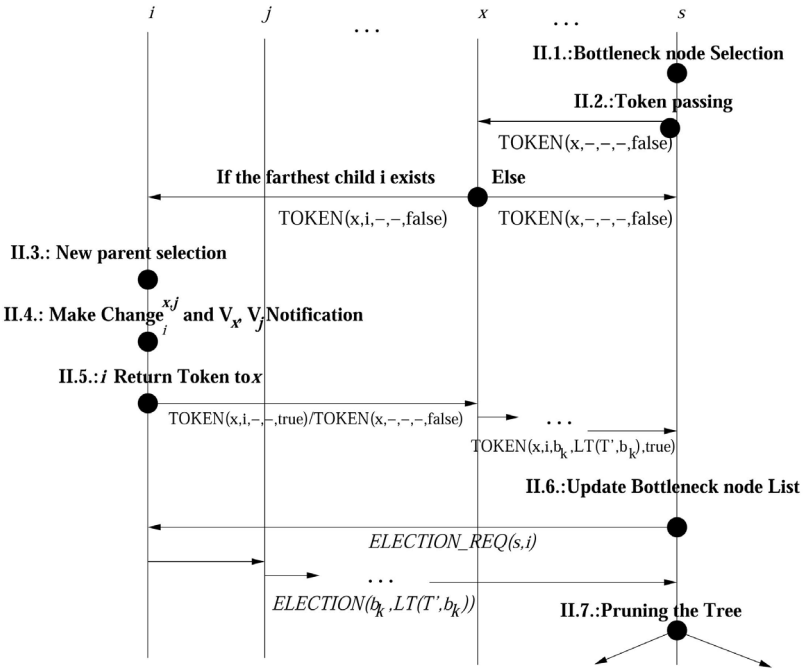

Fig. 2. Overview of second phase of L-REMiT.

Example 1: This example illustrates how to elect the bottleneck node. Node 10 requests that all the nodes elect the bottleneck node. Because nodes 1, 2, 3, 4, 7, 8, and 11 are leaf nodes, they will submit their own multicast lifetime to their respective parent nodes. Once node 9 obtains all of the submissions from nodes 1, 2, 3, and 4, it compares these lifetime values with its own multicast lifetime. It finds that the bottleneck node of its subtree is itself. Then node 9 submits the identifier of itself and the lifetime value to its parent node, node 10. Similarly, node 6 finds out that its subtree's bottleneck node is node 6. Then node 6 submits this election result to node 10 also. Now, node 10 obtains the election results from all of its children nodes. Thus, node 10 finds that node 9 is the only bottleneck node in the tree. ❑

Example 2: This example illustrates a single refinement step. Based on Example 1, node 9 is the bottleneck node. Node 10 passes the L-REMiT token to node 9. In turn, node 9 passes the token to its farthest child node 2. Once node 2 gets the L-REMiT token, node 2 performs the following steps:

1) Node 2 calculates *estimated gains* as explained previously in this paper and finds that $\widehat{Gain_2^{9,8}}$ is the highest positive value.
2) Node 2 makes $Change_2^{9,8}$ and notifies the nodes in V_9 and V_8.
3) Finally, node 2 passes the L-REMiT token back to its previous parent node 9. Node 9 passes the *TOKEN*(9, 2, 9, *LT*(*T'*, 9), *true*) back to node 10. Then, node 10 requests that node 2 do bottleneck election. Similar to Example 1, this bottleneck election goes through nodes 2, 8 and 6. Finally, node 10 gets back the election result from node 6.

Thus, node 10 updates its bottleneck node list B_{10} and finds that node 6 is the only new bottleneck node of tree T'. ❑

Example 3: Following are the steps that are performed in the multicast tree in the second phase of L-REMiT.

A) In the first round, because node 9 is the only node in the list of bottleneck nodes of the tree, node 10 selects node 9 as the bottleneck node to refine.
B) Node 10 passes the L-REMiT token to node 9, and node 9 passes the token to its farthest children node 2 as shown in Fig. 3.
C) Node 2 gets the token, and it conducts refinement in Example 2. After refinement, node 2 makes $Change_2^{9,8}$. Node 2 returns token to node 9. Node 9 updates its bottleneck node list B_9. Then node 9 passes the L-REMiT token back to node 10 and piggyback the information of new bottleneck node in node 9's subtree as shown in Fig. 3.
D) Node 10 gets back the token and requests new bottleneck node election from node 2. After obtained the new election results, node 10 updated its bottleneck node list B_{10}. Node 10 finds that the current bottleneck node is node 5. Then, node 10 begins the second round of refinement; node 10 passes the L-REMiT token to node 5, and node 5 passes the token to its farthest child node 11 as shown in Fig. 4.
E) Similar to step C, node 11 performs refinement and passes token back to node 10. And node 10 will start new bottleneck node election from node 5 and 11. Fig. 4 shows a snapshot of the multicast after the refinements.
F) Node 10 gets back the token and updates its bottleneck node list with new election results. In the third round, node 10 finds the current bottleneck node is node 6 and passes the token to node 6. Node 6 passes the L-REMiT token to node 8 as shown in Fig. 5.
G) Similar to steps C and E, node 8 calculates all of the possible lifetime gains. But node 8 cannot find any positive estimated gains. Node 8 will pass the token back to node 10 and piggyback the "no changes happened" information as shown in Fig. 5.
H) Node 10 gets back the L-REMiT token from node 8 and finds that it is not possible to improve the bottleneck node's lifetime. Thus, node 10 terminates the L-REMiT algorithm. Fig. 6 illustrates the final multicast after refinement. ❑

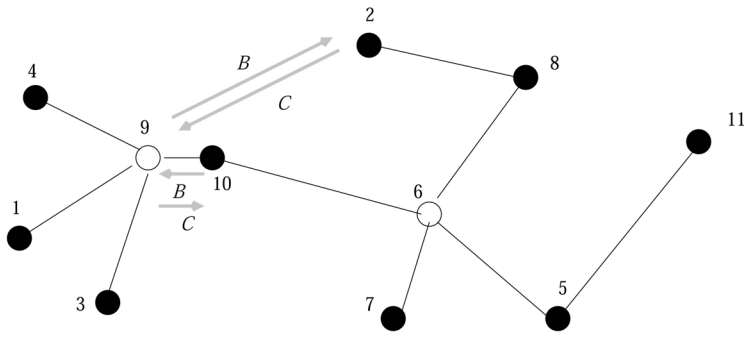

Fig. 3. Snapshot after steps A, B and C of L-REMiT.

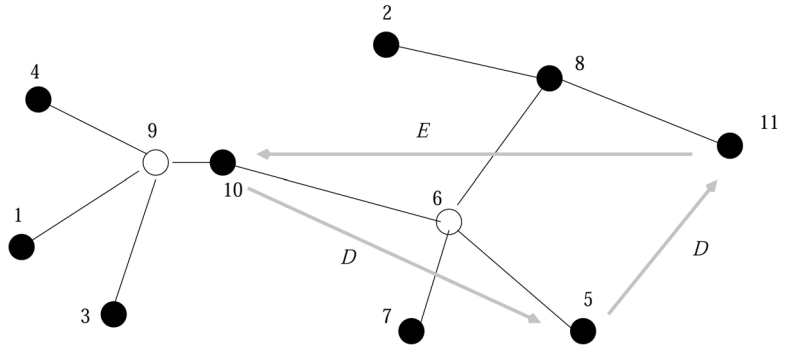

Fig. 4. Snapshot after steps D and E of L-REMiT.

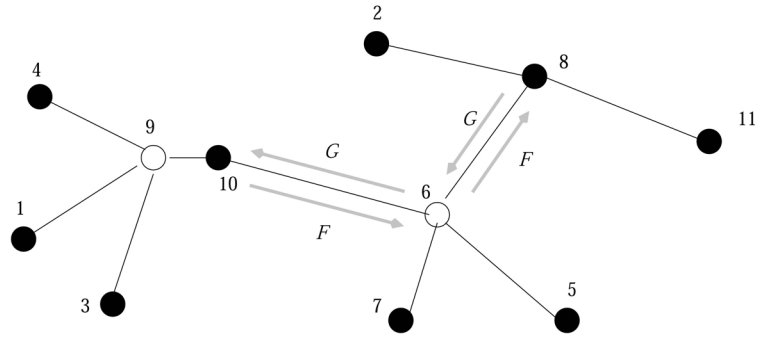

Fig. 5. Snapshot after steps F to G of L-REMiT.

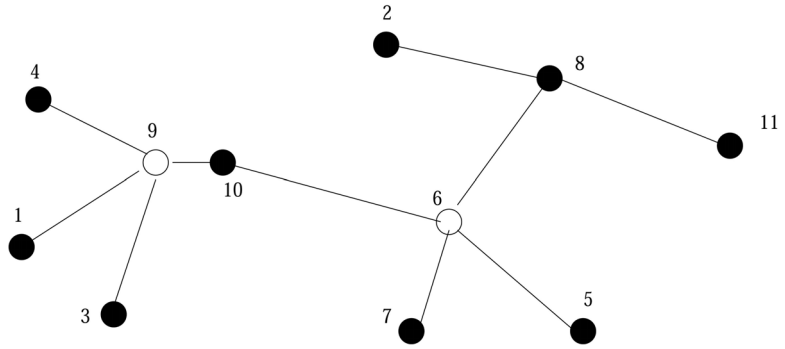

Fig. 6. Snapshot after step H of L-REMiT.

4.5 Worst Case Complexity Analysis

The message complexity of bottleneck node election is $O(N)$, where N is the number of nodes in the network. The message complexity of changing a node's parent is $O(1)$. The message complexity of a round in which a tree refinement is performed is $O(\delta_{max} + 2H)$, where δ_{max} is the maximum number of neighbors in any node's control coveRage

area(CR), and H is the diameter of the network. Hence, the message complexity of L-REMiT is $O(N + R(\delta_{max} + 2H))$, where R is the number of rounds performed. The computational complexity of one refinement is $O(\delta_{max})$. Therefore, the computational complexity of L-REMiT is $O(R\delta_{max})$. The space complexity of L-REMiT for each node is $O(\delta_{max})$ since the size of V is $O(\delta_{max})$.

5. SIMULATION RESULTS

We used simulations to evaluate the performance of the L-REMiT algorithm. We compared our algorithm with MIP, L-MIP, MST, and EWMA-Dist (the distributed version of the EWMA algorithm). Because the EWMA-Dist algorithm is used to build broadcast trees, we extended the EWMA-Dist algorithm for multicasting by pruning the redundant transmissions that do not need to reach the members of the multicast group from the broadcast tree produced by the EWMA-Dist algorithm. The simulations were performed using networks of four different sizes: 10, 40, 70, and 100 nodes. The distribution of the nodes in the networks and the residual battery energy at nodes were randomly generated. Every node was within the maximum transmission range of at least one other node in the network, i.e., the network was connected. We used two different E_{elec} values to represent the long range radio and short range radio. Based on the experiment data in [12], we decided to use $E_{elec} = 0$ to represent the long range radio and $E_{elec} = 4r^{\alpha}$ to represent the short range radio. We ran 100 simulations for each simulation setup consisting of a network of a specified size to obtain the average $LT(T)$ with 95% confidence, and the propagation loss exponent α varied from 2 to 4. For each simulation setup, we used *normalized Lifetime* as the performance metric:

$$normalized\ \ Lifetime = \frac{LT(T_{alg})}{LT(T_{best})},$$

where $LT(T_{best}) = \max\{LT(T_{alg})\}$, $alg \in A = \{L\text{-}REMiT, MST, MIP, L\text{-}MIP, EWMA\text{-}Dist\}$.

5.1 Short Range Radios

For short range radios, the performance is shown in Figs. 7 and 8. One can see the average *normalized Lifetime* (show on the vertical axis) achieved by the algorithms on networks of different sizes (the horizontal axis). The figures show that the solutions for multicast trees obtained by L-REMiT have, on average, higher *normalized Lifetimes* than do the solutions obtained by L-MIP, MIP, EWMA-Dist, and MST, when 100% and 50% of the nodes are group members with different reception energy cost (This is also true for $\alpha = 3$, which is not shown in the figure.)

5.2 Long Range Radios

For long range radios, the performance is shown in Figs. 9 and 10. In the figures, we can see that the multicast trees produced by the L-REMiT algorithm have, on the aver

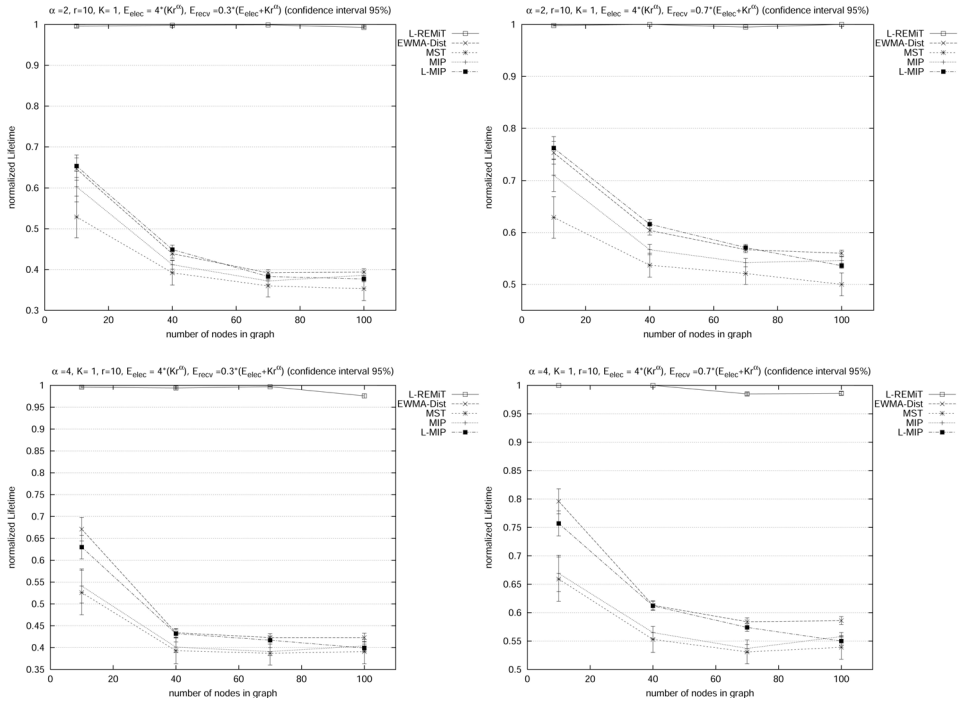

Fig. 7. *Normalized Life* (short range radios, 100% nodes are in multicast group and $\alpha = 2$ (above two) and $\alpha = 4$ (below two)).

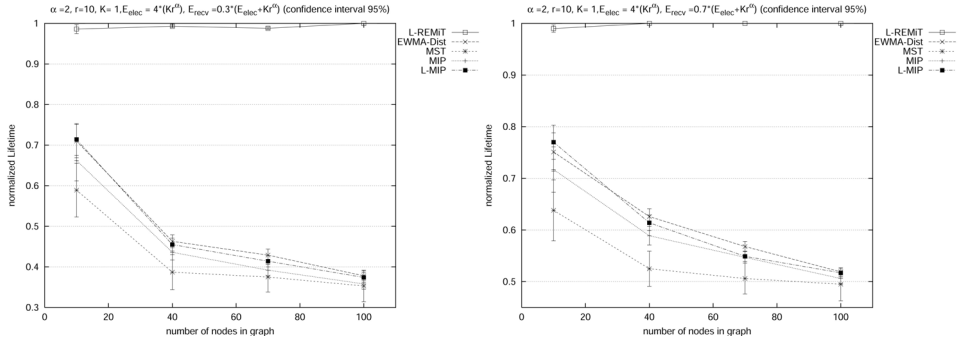

Fig. 8. *Normalized Life* (short range radios, 50% nodes are in multicast group and $\alpha = 2$).

age, higher *normalized Lifetimes* than do those obtained by L-MIP, MIP, EWMA-Dist, and MST, when 100% and 50% of the nodes are group members with propagation loss exponent $\alpha = 2$ and 4. However, one can see that for the propagation loss exponent $\alpha = 4$, L-MIP and L-REMiT have very similar performance. Thus, the figure also reveals that the difference in performance decreases as the propagation loss exponent increases when 100% of the nodes are group members. The main reason for such behavior is that when the propagation loss exponent increases, the lifetimes of the trees that use longer links

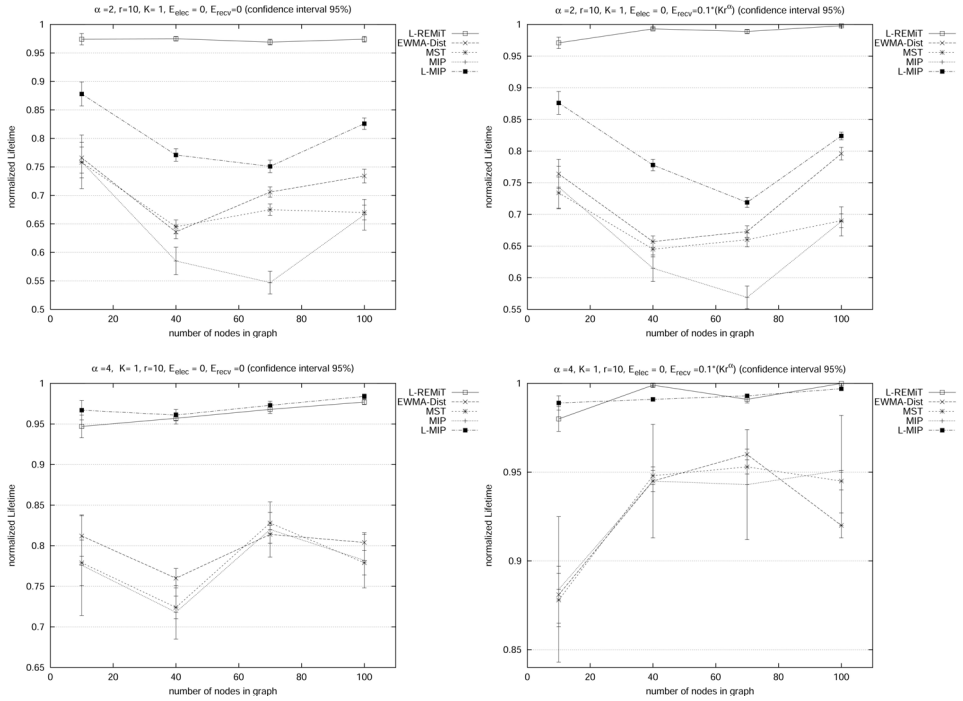

Fig. 9. *Normalized Life* (long range radios, 100% nodes are in multicast group and $\alpha = 2$ (above two) and $\alpha = 4$ (below two)).

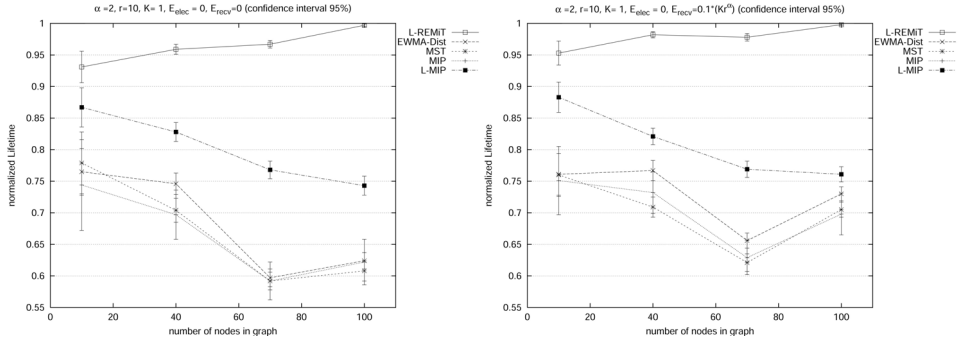

Fig. 10. *Normalized Life* (long range radios, 50% nodes are in multicast group and $\alpha = 2$).

decrease. Consequently, L-MIP and L-REMiT set their transmitting nodes to transmit at lower power levels. Hence, L-REMiT and L-MIP's broadcast trees (because 100% of the nodes are group nodes) become similar when α increases. (This is also true for different group sizes, which is not shown in the figure.) Further, our simulation results show that the energy overhead of L-REMiT is always below 1% of the total energy consumption of all the nodes in the multicast tree within the lifetime of the tree.

Based on our simulation results, we find that L-REMiT achieves better performance than L-MIP, MIP, EWMA, and MST for various scenarios. Because Dist-BIP-A and Dist-BIP-G [4] perform slightly worse than the BIP algorithm (BIP is the broadcast case of the MIP algorithm), L-REMiT should be better than the two distributed versions of the BIP algorithm.

6. DISCUSSION

6.1 L-REMiT Token Recovery

In a WANET, the L-REMiT token may be lost due to unreliable communication. Now, we will discuss how to recover the L-REMiT token once it has been lost. The source node s sets a time-to-live(TTL) for every L-REMiT token. If the TTL times out and the token has not been returned to s, it will regenerate a new L-REMiT token for the current round of refinement. If node s gets back the old token just after s sends out a new token, s will use the parameters in the old token to update the bottleneck node list and calculate the new b_s of T' in the next round. If node x gets more than one token before the refinement is complete, it will drop the old token. If node x gets the new token after x gets back the token from i, x will just copy the parameters from the old token to the new token and return the new token to s. If node k ($\forall k$ such that k is x's ancestor) gets the two tokens at the same time, k will drop the old token and forward the new token. And the value of the L-REMiT token's TTL should be larger than the round-trip time from s to the farthest leaf node on the tree.

6.2 Delay

Certain applications may have application specific delay constraints. We will not discuss this issue in this paper. [15, 16] and [17] dealt with the delay constrained multicast tree building problem by incorporating a delay constraint into the node metric. We also can add a delay constraint parameter to the node metric, which is similar to the residual battery energy parameter in our lifetime metric. Thus, nodes running the L-REMiT algorithm can only switch from the old parent to a new parent that satisfies with the delay constraint.

6.3 Centralized vs. Distributed

A distributed algorithm is more suitable for the energy-efficient multicast tree problem in ad hoc networks [2, 5, 18]. However, only two distributed algorithms have been proposed for building minimum energy source-based broadcast/multicast trees [4, 5]. L-REMiT is the first distributed algorithm for extending multicast tree lifetimes. However, if a base station is available (e.g., wireless sensor network) or some node has more energy to work as a base station, then our algorithm can still be easily implemented in a centralized algorithm manner. In these scenarios, a centralized algorithm is better because the communication overhead can be reduced.

6.4 Sparse vs. Dense Model

L-REMiT extends a multicast tree's lifetime by switching a node's parent node in an MST and pruning the redundant transmissions. MST is good for a dense group, which means that the major nodes in the network are group members. But when a group in the network is sparse, the overhead of building MST as an initial tree may be very high. There are two possible solutions for sparse groups: 1) Limited the range to build MST so that all of the group members within boundary of the range, then use the bounded MST as the initial tree of L-REMiT; and 2) Build a MST that only includes group members, then add some non-group nodes in the tree to extend the lifetime. And the scheme is similar to L-REMiT algorithm which only trying to switch nodes' parent to other tree nodes.

6.5 Online Refinement

Now, we will discuss how L-REMiT can do online refinements. Nodes may switch a parent node to a new parent node in the L-REMiT algorithm, so every node should cache the multicast messages for some time. Thus, once node i switches from x to j, it can pull the recently multicast messages from j. In WANETs, there may exist more than one source node and unicast traffic. The multicast source nodes should start up the L-REMiT algorithm and periodically adapt to the dynamic nature of WANETs. The period of L-REMiT refinement needs further study.

7. CONCLUSIONS

In this paper, we have proposed the L-REMiT algorithm for refining a multicast tree with the goal of extending its lifetime in a WANET. L-REMiT is a distributed algorithm that employs an energy consumption model for wireless communication, which takes into account energy losses due to radio propagation as well as transceiver electronics. This enables L-REMiT to adapt a given multicast tree to a wide variety of wireless networks irrespective of whether they use long-range radios or short-range radios. We have also discussed how to recover the L-REMiT token when it is lost.

We have shown that L-REMiT outperforms other schemes proposed in the literature: MST, MIP, and EWMA. Further, the energy consumption overhead of the algorithm itself is very small compared with the sum of the energy consumption at all the nodes in the multicast tree within the lifetime of the tree.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments that helped to improve the quality of the paper. This work was supported in part by NSF grants ANI-0123980 and ANI-0196156.

APPENDIX I. FUNCTIONS OF THE MULTICAST LIFETIME FROM THE NODE'S LOCAL POINT OF VIEW

Following are the two functions used to compute new multicast lifetimes at nodes j and x after $Change_i^{x,j}$ from node i's point of view.

```
New_Lifetime_at_old_parent(x) {
  if (δ_x = 2) then
    // If x is not i's parent node any more,
    // x will become a leaf node. Using Eq. (2).
    LT(T', x) = R_x / E_recv;
  else // (δ_x > 2)
    if (r_{i,x} < d_x) then
      // r_{i,x} is the distance between nodes i and x.
      LT(T', x) = R_x / f(d_x, E_elec, E_recv);
    else // r_{i,x} ≥ d_x
      LT(T', x) = R_x / f(d'_x, E_elec, E_recv);
  return LT(T', x);
}
```

Fig. 11. Pseudocode for *New_Lifetime_at_old_parent*(x).

```
New_Lifetime_at_new_parent(x, i_j) {
  if (r_{i,i_j} ≤ d_{i_j}) then
    // r_{i,i_j} is the distance between nodes i and i_j.
    LT(T', i_j) = R_{i_j} / f(d_{i_j}, E_elec, E_recv);
  else // (r_{i,i_j} > d_{i_j})
    LT(T', i_j) = R_{i_j} / f(r_{i,i_j}, E_elec, E_recv);
  return LT(T', i_j);
}
```

Fig. 12. Pseudocode for *New_Lifetime_at_new_parent*.

APPENDIX II. PSEUDOCODE OF SECOND PHASE OF THE L-REMiT ALGORITHM

The syntax of the message in L-REMiT as follows:
Asynch-send(*dest*, *Msg_Type* < $parameter_1$, …, $parameter_n$ >)
Asynch-broadcast(*bdest*, *Msg_Type* < $parameter_1$, …, $parameter_n$ >)
Local-Broadcast(*bdest*, *Msg_Type* < $parameter_1$, …, $parameter_n$ >)
1) *dest* is the destination.
2) *bdest* is the set of nodes to broadcast.
3) *Msg_Type* is the type of the messages.
4) $parameter_i$ is the ith parameter in the message.

Asynch-send will unicast the message to *dest*. Asynch-broadcast will broadcast the message to every node on the multicast tree. However, node i's Local-Broadcast will local broadcast to nodes which are in V_i.

```
// Initiate L-REMiT algorithm
Asynch-broadcast(every tree node, BottleneckElection < z, F >)
// F is the field for piggybacking bottleneck node list information.

when Asynch-recv(BottleneckElection < a, F >)
Subtree.a := F
if Subtree is full of the return information from all of the children nodes then
    For All m is z's child or z do
        z_bottleneck := Select_minimum_lifetime(Subtree.m, z)
        Add z_bottleneck to the identifier list of Tree_z
        Add LT(T, z_bottleneck) to the minimum lifetime list of Tree_z
        // Note: Tree_z is the list of information of bottleneck node in the tree.
    bottleneck := Select-from_non-refined(Tree_z)
    if bottleneck ≠ NULL then
        A: Asynch-send(bottleneck, lremitToken < false, bottleneck, F >)
        // Here, F is the field for piggybacking the new lifetime of the node after refinement.
    else
        B: Asynch-broadcast(every tree node, PruneTree < >)
        exit L-REMiT algorithm

when Asynch-recv(lremitToken < flag, F >)
if flag = true then
    update bottleneck node list
    goto A
else // No more refinement needed
    goto B
```

Fig. 13. Source z's Pseudocode in the L-REMiT algorithm.

```
when Asynch-recv(BottleneckElection < a, F >)
if a is my child then
   Subtree.a := F
if Subtree is full of the returned information from all of the children nodes then
   For All m is i's child or i do
      i_bottleneck := Select_minimum_lifetime(Subtree.m, i)
   Add i_bottleneck to the identifier list of G
   Add LT(T, i_bottleneck) to the minimum lifetime list of G
   Asynch-send(i's parent, BottleneckElection < i, G >)

when Asynch-receive(lremitToken < flag, parent, F >)
if parent = i then
```

Fig. 14. Preudocode for the L-REMiT algorithm at any tree node i.

```
   if flag = true then
      update_bottleneck_node_info(F)
   l := Select_from_non-refined (farthest children list of i)
   if l = NULL then
      Asynch-send(z, lremitToken < flag, parent, F>)
   else
      Asynch-send(l, lremitToken < flag, i, F >)
else
   x := parent
   S_i := a set of i's non-connected tree neighbors which are not on i's downstream subtree
   For All i_j ∈ S_i do
      Gain_i^{x,i_j} := min{LT(T', i), LT(T', x), LT(T', i_j)} − LT(T, x)
   Gain_i^{x,j} := max{Gain_i^{x,i_j}}
   if Gain_i^{x,j} > 0 then
      Asynch-send(j, Join_REQ < i >)
   else
      flag := false
      Asynch-send(x, lremitToken < flag, parent, F>)

when Asynch-receive(PruneTree <>)
goto C

when Asynch-recv(Join_REP)
move j from NCTN_i to CTN_i
move x from CTN_i to NCTN_i
Asynch-send(x, leave < i >)
flag := true
F.i := LT(T, i); F.x := LT(T, x); F.j := LT(T, j)
Asynch-send(x, lremitToken < flag, parent, F >)

when Asynch-recv(leave < a >)
move a from CTN_i to NCTN_i
C: if i is a leaf node now and i is not a group member then
   Asynch-send(i's parent, leave < i >)
Update l_i
Local-Broadcast(V_i, Neighbor_Update < l_i >)

when recv Local-Broadcast(Neighbor_Update < a, l_a >)
update neighbor a's l_a

when Asynch-recv(Join_REQ < a >)
move a from NCTN_i to CTN_i
Asynch-send(a, Join_REP)
Update l_i
Local-Broadcast(V_i, Neighbor_Update < l_i >)
```

Fig. 14. (Cont'd) Preudocode for the L-REMiT algorithm at any tree node *i*.

REFERENCES

1. B. Wang and S. K. S. Gupta, "On maximizing lifetime of multicast trees in wireless ad hoc networks," in *Proceedings of International Conference on Parallel Processing* (*ICPP '03*), 2003, pp. 333-340.
2. J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "On the construction energy-efficient broadcast and multicast tree in wireless networks," in *Proceedings of IEEE INFOCOM 2000*, 2000, pp. 585-594.
3. A. E. F. Clementi, P. Crescenzi, P. Penna, G. Rossi, and P. Vocca, "On the complexity of computing minimum energy consumption broadcast subgraphs," in *Proceedings of 18th Annual Theoretical Aspects of Computer Science* (*STACS*), Vol. 2010, 2001, pp. 121-131.
4. J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "Distributed algorithms for energy-efficient broadcasting in ad hoc networks," in *IEEE Military Communications Conference* (*MILCOM*), 2002, pp. 820-825.
5. M. Cagalj, J. P. Hubaux, and C. Enz, "Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues," in *Proceedings of ACM International Conference on Mobile Networking and Computing* (*MOBICOM 2002*), 2002, pp. 172-182.
6. A. Misra and S. Banerjee, "MRPC: maximizing network lifetime for reliable routing in wireless environment," in *IEEE Wireless Communications and Networking Conference* (*WCNC*), 2002, pp. 800-806.
7. J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "Resource management in energy-limited, bandwidth-limited, transceiver-limited wireless networks for session-based multicasting," *Computer Networks*, Vol. 39, 2002, pp. 113-131.
8. I. Kang and R. Poovendran, "On the lifetime extension of energy-constrained multihop broadcast networks," in *Proceedings of 2002 International Joint Conference on Neural Networks*, 2002, pp. 365-370.
9. B. Wand and S. K. S. Gupta, "G-REMiT: an algorithm for building energy efficient multicast trees in wireless ad hoc networks," in *Proceedings of 2nd IEEE International Symposium on Network Computing and Applications* (*NCA '03*), 2003, pp. 265-272.
10. B. Wang and S. K. S. Gupta, "S-REMiT: an algorithm for enhancing energy-efficiency of multicast trees in wireless ad hoc networks," in *Proceedings of IEEE 2003 Global Communication Conference*, 2003, pp. 3519-3524.
11. W. C. Y. Lee, *Mobile Communication Engineering*, McGraw-Hall, 1993.
12. L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *Proceedings of IEEE INFOCOM 2001*, 2001, pp. 1548-1557.
13. S. Singh, C. S. Raghavendra, and J. Stepanek, "Power-aware broadcasting in mobile ad hoc networks," in *Proceedings of International Symposium on Personal, Indoor and Mobile Radio Communications* (*PIMRC '99*), 1999, pp. 22-31.
14. R. Gallager, P. A. Humblet, and P. M. Spira, "A distributed algorithm for minimum weight spanning trees," *ACM Transactions on Programming Languages and Systems*, Vol. 5, 1983, pp. 66-77.
15. M. Parsa, Q. Zhu, and J. J. Garcia-Luna-Aceves, "An iterative algorithm for de-

lay-constrained minimum-cost multicasting," *IEEE/ACM Transactions on Networking* (*TON*), Vol. 6, 1998, pp. 461-474.
16. S. Raghavan, G. Manimaran, and C. S. R. Murthy, "A rearrangeable algorithm for the construction delay-constrained dynamic multicast trees," *IEEE/ACM Transactions on Networking* (*TON*), Vol. 7, 1999, pp. 514-529.
17. A. Hac and K. Zhou, "A new heuristic algorithm for finding minimum-cost multicast trees with bounded path delay," *International Journal of Network Management*, Vol. 9, 1999, pp. 265-278.
18. P. J. Wan, G. Calinescu, and O. Y. Li, "Minimum-energy broadcast routing in static ad hoc wireless networks," in *Proceedings of the IEEE INFOCOM 2001*, 2001, pp. 1161-1171.

Bin Wang is a Ph.D. candidate and research assistant in Department of Computer Science and Engineering at Arizona State University, Tempe, Arizona. He received his M.E. and B.E. in National Key Laboratory of Switching Technology and Telecommunication Networks and Computer Engineering Department from Beijing University of Posts and Telecommunication, Beijing, China, respectively. His research interests include pervasive/ubiquitous computing, wireless network, mobile computing, and middleware.

Sandeep Kumar S. Gupta received the B.Tech. degree in Computer Science and Engineering from Institute of Technology, Banaras Hindu University, Varanasi, India, the M.Tech. degree in Computer Science and Engineering from Indian Institute of Technology, Kanpur, and the M.S. and Ph.D. degree in Computer and Information Science from The Ohio State University, Columbus, Ohio. He is currently an Associate Professor in the Department of Computer Science and Engineering at Arizona State University, Tempe, Arizona. He has previously served as a faculty at Duke University, Ohio University, and Colorado State University. His research interests include wireless sensor networks, mobile and pervasive computing, middleware, and embedded sensor networks for biomedical applications. His recent research papers are available at http://shamir.eas.asu.edu/~mcn. Dr. Gupta is a member of the ACM and a senior member of the IEEE.