

Quality of Service Routing Tree for Wireless Ad Hoc Networks

Khaled M. Alzoubi
Saint Xavier University
Chicago, IL 60655
alzoubi@sxu.edu

Moussa S. Ayyash
Illinois Institute of Technology
Chicago, IL 60616
ayyamous@iit.edu

Abstract—Given the weighted graph $G(V, E)$, where the weight of each edge represents the bandwidth of the corresponding link. We define the *Quality of Service (QoS) path* for any pair of nodes in V as a path with the bottleneck edge maximized. We further define the *QoS tree* $G'(V, E') \subseteq G(V, E)$, as a subgraph such that for any pair of nodes u and v in G' there exist a QoS path between u and v , and all intermediate edges are a subset of E' . In this paper, we propose a distributed algorithm to construct a QoS tree, and the internal nodes of the tree form the backbone that maintains the QoS paths. Then we propose a proactive routing schemes that runs on the QoS tree. Only the nodes of the backbone are responsible for the maintenance of the routing tables. The QoS tree and its backbone are constructed using $O(n \log n)$ messages in $O(n \log n)$ time.

I. INTRODUCTION

Applications of *mobile ad hoc networks* (MANETs) occur in situations such as emergency search-and-rescue operations, and data acquisition operations in hostile terrain. In situations like battlefields or major disaster areas, MANETs need to be deployed immediately without base stations or wired infrastructures. These networks are typically characterized by scarce resources (bandwidth, power, etc.), lack of established backbone infrastructure, high error rates, and a dynamic topology [1].

As a result of the rising popularity of real-time multimedia applications and because of the potential commercial usage of MANETs, many researchers focused their interest on supporting an as of yet unavailable *Quality of Service (QoS) architecture model* [2]. Such a model concentrates on providing high-quality media services by meeting specific QoS *metrics* such as available link bandwidth and delay. Generally, the QoS problem is intricate and follows logically because of the special characteristics of the MANETs platform and its need for metric monitoring. Also, a common networking practice that leads to performance degradation, is the practice of global network flooding that leads to serious problems such as heavy contention and intense collisions.

Even though providing a QoS solution for MANETs is complicated, some promising research on QoS routing in MANETs has been done. Examples of these algorithms are: Core-Extraction Distribution Ad Hoc Routing (CEDAR) protocol [3], Quality of Service for Ad Hoc Optimized Routing (QOLSR) protocol [4], and Robust Quality of Service Routing (RQoS) protocol [5]. Both CEDAR and QOLSR propose topology control algorithms that extract nodes for network

control regardless of their stability conditions or network resources availability. In [6], RQoS defines node stability as a monitor of the variation of the set of neighbors for each node over a predefined period.

This paper devises a new distributed algorithm for constructing a QoS routing tree, which is designed to serve as a *QoS virtual backbone (QoS-VBB)* for packets routing. Advantages of the QoS-VBB approach were discussed in [7]. Several approaches have been proposed in the literature for constructing VBBs for wireless ad hoc networks [8], [9].

The unique construction approach of our QoS tree has valuable properties that added the following features to the tree: (1) Guarantees the optimal maximum bandwidth path between any pair of nodes (2) A connected VBB consists of a subset of the QoS tree nodes is responsible for routing information maintenance (3) This QoS tree is practical for both reactive and proactive routing (4) The time and message complexity for its construction is $O(n \log n)$. In addition to the QoS tree with the above mentioned features, we propose a proactive routing techniques based on the QoS tree. The maintenance process of the QoS tree is provided in [10].

The rest of the paper is organized as follows. Section II introduces the network model and presents the definitions and terminologies related to QoS tree. Section III describes the construction of the QoS tree, its analysis and correctness discussion. In Section IV, we present potential routing approaches over the QoS tree. Finally, we conclude and highlight future directions in Section V.

II. NETWORK MODEL

This section introduces the related network model, assumptions, and definitions. The network under study is modeled by the undirected and weighted graph $G(V, E)$, where V represents the set of vertices in the graph and corresponds to the set of hosts (nodes) in the network, and E represents the set of edges in the graph and corresponds to the set of links in the network. All nodes in the network have the same maximum transmission range, and all nodes remain connected at all times. An edge exists between two vertices *iff* the distance between the two corresponding nodes is less than or equal to the maximum transmission range. Two nodes can communicate directly if they share the same edge. The available bandwidth of each link is represented by the weight of the edge. All nodes share the same scarce bandwidth

channel. We make the typical assumption of using the well known IEEE 802.11 standard. We also make the assumption that each node is aware of the available bandwidth for all its links. We further provide the following definitions as a lead to the understanding of the algorithm.

Definition 1: The *id* of a node u denoted by u itself is a number that uniquely identifies a node in the graph. Each tree T is identified by the *id* of its root, if the root of the tree has *id* i , then the *id* of the tree is T_i .

Definition 2: Given the weighted graph $G(V, E)$, the bottleneck edge of a path between any two nodes in the graph G is the edge with the minimum weight among all edges on the path.

Definition 3: We define the QoSRT for the weighted graph $G(V, E)$ as a subgraph $G'(V, E')$ such that G' is a tree, and for any pair of nodes u and v in G' there exist a path between u and v , where all intermediate edges are subset of E' , and the bottleneck for the path is maximized. We also define the QoS path between two nodes as a path such that the bottleneck is maximized.

Definition 4: Two nodes u and v are neighbors *iff* there exist an edge that is incident at both nodes. An edge between u and v is denoted by $u \leftrightarrow v$. The best edge (*BE*) for the node u (denoted by BE_u) among all edges incident at u (denoted by $u \leftrightarrow v_i$, where v_i is the set of all nodes incident at u) is the edge with the maximum bandwidth. If there exist more than one edge incident at u , and have the same maximum bandwidth, then the BE_u is the one incident at u and at $\max(v_i)$. The best node (*BN*) of u is the node v incident at the edge BE_u .

Definition 5: An edge between two nodes u and v is referred to as an external edge *iff* both of u and v belong to two different adjacent trees, and the edge $u \leftrightarrow v$ is in neither tree. Two trees T_i and T_j are neighbors *iff* there exist at least one external edge between the two trees. T_j is said to be the best neighboring tree (*BT*) for the tree T_i *iff* the the maximum-bandwidth external edge (*MBEE*) for the tree T_i is incident at a node from T_i and a node from T_j , or if there are other trees besides T_j with the same maximum-bandwidth external edge, and T_j has the largest *id* among those trees. If more than one external edge ($u_i \leftrightarrow v_j$) have the same maximum bandwidth between T_i and T_j , then the *MBEE* for T_i is selected as follow: 1) If $T_i > T_j$, then the *MBEE* is the edge incident at $\max(u_i)$ from T_i and $\max(v_k)$ from T_j , where $v_k \subseteq v_j$, and v_k is a set of neighbors to $\max(u_i)$. 2) If $T_j > T_i$, then the *MBEE* is the edge incident at $\max(v_j)$ from T_j and $\max(u_k)$ from T_i , where $u_k \subseteq u_i$, and u_k is a set of neighbors to $\max(v_j)$. The best external-edge weight (*BEEW*) of the tree T_i is the weight of the *MBEE* that connects T_i to T_j . A tree T_i denotes the node u by *BN1*, and the node v by *BN2*.

Lemma 6: Definition 5 defines *MBEE* for a given tree, and insures that if the *MBEE* of two adjacent trees are between the same two trees, then the same edge is selected by both trees as *MBEE*.

Proof: Let $u_i \leftrightarrow v_j$ represent the set of all maximum weight edges between T_i and T_j , there are two cases to

consider: *first case*, If $T_i > T_j$, then by Definition 5 the *MBEE* for T_i is the edge incident at $\max(u_i)$ from T_i and $\max(v_k)$ from T_j , where $v_k \subseteq v_j$, and v_k is a neighboring set of $\max(u_i)$, and the *MBEE* for T_j is the edge incident at $\max(u_i)$ from T_i and $\max(v_k)$ from T_j , where $v_k \subseteq v_j$, and v_k is a neighboring set of $\max(u_i)$. *Second case*, If $T_j > T_i$, then by Definition 5 also, the *MBEE* for T_j is the edge incident at $\max(v_j)$ from T_j and $\max(u_k)$ from T_i , where $u_k \subseteq u_i$, and u_k is a neighboring set of $\max(v_j)$, and the *MBEE* for T_i is the edge incident at $\max(v_j)$ from T_j and $\max(u_k)$ from T_i , where $u_k \subseteq u_i$, and u_k is a neighboring set of $\max(v_j)$. This shows that the selected edge is the same by both trees. ■

III. QOS TREE

This algorithm consists of two phases [10]. During the operation of the first phase, nodes are partitioned into fragments. Each fragment consists of a set of connected nodes forming a tree. In the second phase which consists of several rounds, all trees are combined into one large tree. The root of the tree is the highest level leader. Other leaders of subtrees also exist. Thus, a node may have multi level leaders, with the root as its highest level leader, and the closest leader as its lowest level leader. The bottleneck of the path between any two nodes is maximized, thus the tree is an optimal QoS tree.

First Phase: We assume each node knows the *ids* of all its neighboring nodes, and knows the bandwidth of all its edges. The goal of each node is to be part of a tree with at least two nodes. Each node selects the node incident at the highest bandwidth edge to be its neighbor in the tree, and refers to it as its best node (*BN*). Phase I of the algorithm follows the following steps: 1) Each node selects the edge with the highest bandwidth to be in the tree, if more than one edge have the same maximum bandwidth, the one incident at the largest *id* node is selected. Two neighboring nodes may select the same edge. 2) A set of connected edges form a single component. 3) The edge that has been selected by both nodes in each component selects the node with the largest *id* as the root of the tree.

In Figure 1, phase-I generated four components (trees), since the edge $1 \leftrightarrow 2$ has the highest bandwidth among all edges incident at node 1, node 1 selects node 2 for its best edge. Similarly node 2 selects node 3, and node 3 selects node 2. Since the edge $2 \leftrightarrow 3$ is selected by both nodes 2 and 3, and since node 3 has a higher *id* than node 2, node 3 becomes the root of the tree that contains the nodes 1, 2, and 3. Similarly, node 5 becomes the root of the tree that contains the nodes 4, 5, and 6; node 9 becomes the root of the tree that contains the nodes 9, 8, and 7; node 12 becomes the root of the tree that contains the nodes 12, 11, and 10.

Lemma 7: Given a weighted graph $G(V, E)$, each edge $u \leftrightarrow v$ of any component $S(V', E')$ formed by phase-I is selected by either u or v or both.

Proof: The prove is followed from the construction process in Phase-I. ■

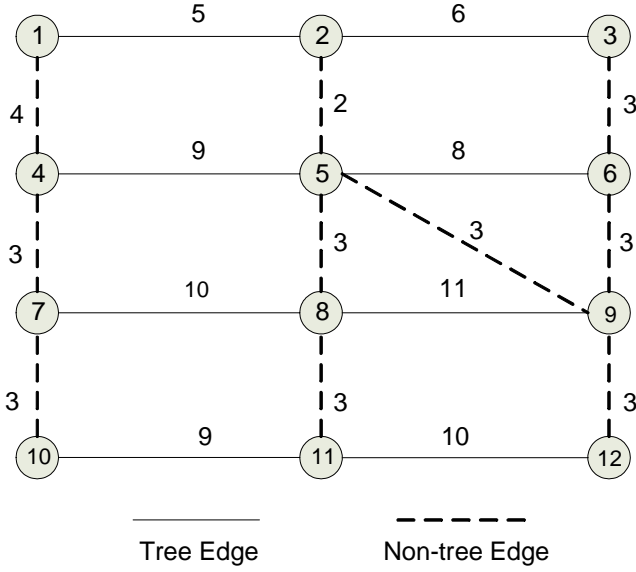


Fig. 1. Phase I Example

Lemma 8: Given a weighted graph $G(V, E)$, any component $S(V', E')$ formed by phase-I is cycle-free, and hence it is a tree.

Proof: Assume a given component generated by phase-I has at least one cycle $u_1, u_2, \dots, u_i, u_1$. By Lemma 7 each edge is selected by at least one of the nodes incident at that edge, since the number of edges in any cycle equals to the number of nodes, then each node selects one unique edge. Let each consecutive pair of nodes in the sequence u_1, u_2, \dots, u_i , represent the edge selected by the first node in the pair, i.e. u_j selects u_{j+1} for its BE, where $1 \leq j < i$, this also implies the edges $u_1 \leftrightarrow u_2, u_2 \leftrightarrow u_3, \dots, u_{i-1} \leftrightarrow u_i$ are sorted in ascending order of their weight. To complete the cycle, node u_i selects u_1 , but in order for u_i to select u_1 , $weight(u_i \leftrightarrow u_1)$ must be greater than $weight(u_{i-1} \leftrightarrow u_i)$. Since u_1 selected the edge $u_1 \leftrightarrow u_2$ as its best edge, then $weight(u_1 \leftrightarrow u_2) > weight(u_1 \leftrightarrow u_i)$, but from the ordering of the edges we know that $weight(u_{i-1} \leftrightarrow u_i) > weight(u_1 \leftrightarrow u_2)$ which implies $weight(u_{i-1} \leftrightarrow u_i) > weight(u_i \leftrightarrow u_1)$, thus u_i must select u_{i-1} for its best edge, and the cycle can't be completed. Hence the component may not have a cycle, therefore, the component is a tree. ■

Theorem 9: Given a weighted graph $G(V, E)$, any connected component $S(V', E')$ formed by phase-I has exactly one edge selected by both of the nodes that are incident at the edge itself. This edge is referred to as a *unique edge*.

Proof: Each node selects exactly one edge to be its best edge, this implies each node selects exactly one other node to be in its component. We prove our theorem by contradiction. First, assume that each node in the component selects a different edge from the rest of the nodes; i.e. no component has an edge that is selected by two nodes. This implies, $|E'| = |V'|$, which means the component must have a cycle, and therefore, our assumption violates and contradicts

Lemma 8. Second, assume that more than one edge in the component have been selected by both of their nodes. Since the edge between any pair of nodes is selected by at least one of the nodes, then our second assumption implies that $|E'| < |V'| - 1$. This contradicts our claim that the component is connected and it is a tree. Since the generated component is a tree (by Lemma 8), then $|E'| = |V'| - 1$. Therefore, there is exactly one edge selected by both of its nodes. ■

Lemma 10: Given a weighted graph $G(V, E)$, and the connected component $S(V', E')$ formed by phase-I. The path between any two nodes u and v from the component S must include the edge that corresponds to the $\min(weight(BE_u), weight(BE_v))$.

Proof: Since S is a tree, only one unique path exists between u and v . Let the node sequence $u = n_1, n_2, \dots, n_i, n_{i+1}, \dots, n_j = v$ represent the path from u to v . Let $weight(BE_u) > weight(BE_v)$. Assume $BE_v \neq n_{j-1} \leftrightarrow n_j$, this implies the edge $n_{j-1} \leftrightarrow n_j$ was selected by the node n_{j-1} . To guarantee the connectivity of the path then for each $1 \leq i < j$, $BE_i = n_i \leftrightarrow n_{i+1}$, which implies $weight(n_1 \leftrightarrow n_2) < \dots < weight(n_{i-1} \leftrightarrow n_i) < weight(n_i \leftrightarrow n_{i+1}) < \dots < weight(n_{j-1} \leftrightarrow n_j)$. Therefore, $weight(BE_u) < weight(BE_v)$, which contradicts our claim $weight(BE_u) > weight(BE_v)$. It further shows that the end edge with the minimum weight must be on the path. ■

Lemma 11: Given a weighted graph $G(V, E)$, and the connected component $S(V', E')$ formed by phase-I. The weight of the bottleneck edge on the path between any two nodes u and v in the component $S = \min(weight(BE_u), weight(BE_v))$.

Proof: Let the sequence $u = n_1, \dots, n_i, \dots, n_j = v$ represents the path from u to v . Assume the bottleneck edge is the edge $n_i \leftrightarrow n_{i+1}$, where $1 < i < j - 1$. By Lemma 7, the edge $n_i \leftrightarrow n_{i+1}$ is selected by either n_i or n_{i+1} or by both. If the edge is selected by n_i , then $weight(n_{i-1} \leftrightarrow n_i) < weight(n_i \leftrightarrow n_{i+1})$. Since the edge $n_{i-1} \leftrightarrow n_i$ is on the path from u to v , this contradicts our assumption that the bottleneck edge is the edge (n_i, n_{i+1}) . If the edge is selected by n_{i+1} , then $weight(n_{i+1} \leftrightarrow n_{i+2}) < weight(n_i \leftrightarrow n_{i+1})$, which contradicts our assumption. If the edge is selected by both n_i and n_{i+1} , then $weight(n_{i-1} \leftrightarrow n_i) < weight(n_i \leftrightarrow n_{i+1}) > weight(n_{i+1} \leftrightarrow n_{i+2})$, which contradicts our assumption also. This implies the nodes of the bottleneck edge can't have any edge on the path from u to v with smaller weight than their edge. Since all the intermediate nodes on the path have exactly two edges from the path, except for only the two end nodes, where each has exactly one edge from the path edges. This implies the only two possible edges left for the bottleneck edge are the two edges incident at the source or at the destination. Since the bottleneck edge is the one with the minimum weight, then the bottleneck edge must be the one with the $\min(weight(n_1 \leftrightarrow n_2), weight(n_{j-1} \leftrightarrow n_j))$. ■

Theorem 12: Given a weighted graph $G(V, E)$, and a connected component $S(V', E')$ formed by phase-I. Let the nodes $\{u, v\}$ be any pair of nodes from V' , then there exist one QoS-path from u to v such that all edges on the path are subset of E' and the bottleneck is maximized.

Proof: Let $BE_u > BE_v$, by Lemma 10 the edge BE_v must be on the path from u to v over S . By Lemma 11 the edge BE_v is the bottleneck edge on the path from u to v over S . Since the edge BE_v has the maximum weight among all edges adjacent to v , and the edge BE_v is the bottleneck edge on the path from u to v over S , it follows that this path is the QoS path. ■

Second Phase: The second phase consists of several rounds. In each round we implement phase I of the algorithm by treating the trees as nodes and the edges between the trees are treated as edges between nodes. Each tree is initially active, then in each round of this phase an active tree is combined with at least one other active tree forming a new larger tree. The same rules in phase I apply to phase II. Thus, each new component will have exactly one unique edge, where this edge is selected by the two trees connected by the edge. This edge will be referred to as *unique Bridge-edge*, while all other edges in the component that connect two trees are referred to as *Bridge-edges*. The node with the largest id in the unique Bridge-edge becomes the root of the new tree, which consists of all the nodes of the new formed component. This process is repeated for each new set of trees until we finally have one tree that includes all the trees from phase I. The following steps summarize the details of Phase II:

- Each node initializes the variables: $id(BT)$, $BN1$, $BN2$, $NEXT$ to NIL , and the variable $BEEW$ to 0.

- The root i of each tree sends a $SEARCH(T_i, r)$ message to all nodes in the tree to look for the best neighboring tree, where r represents the round number and is initialized to 1. Round number (r) is used to synchronize the $SEARCH$ messages among neighboring trees. The $SEARCH$ message is retransmitted by each node in the tree.

- Whenever a leaf node u in the tree T_i that is adjacent to other trees receives a $SEARCH$ message from its parent, and $SEARCH$ messages from all its neighboring trees for the current round (r), it finds the best tree, then it sends to its parent the $REPLY(u, id(BT), BEEW, BN1, BN2)$ message, where $BN1 = u$ in this case.

- Whenever a leaf node u in the tree T_i that is not adjacent to any other tree receives the $SEARCH$ message, it sends to its parent a $REPLY(u, NIL, 0, NIL, NIL)$ message.

- Whenever an internal node u in the tree T_i receives a $SEARCH$ message from its parent, and $SEARCH$ messages from all its neighboring trees for the current round (r), it looks for the best neighboring tree if any and the best edge that connects u to the BT . Then, it assigns the corresponding values for the variables $id(BT)$, $BEEW$, $BN1$, and $BN2$ accordingly.

- Whenever a node u receives a $REPLY$ message from a child node, it first compares the value of $BEEW$ in the $REPLY$ message with its own $BEEW$, if it is greater than its own, it updates its $BEEW$, $id(BT)$, $BN1$ and $BN2$ to the values in the $REPLY$ message, it also stores the id of the sender in its $NEXT$ (initially assigned to NIL) variable to maintain the path to the best edge. If the value of $BEEW$ in the $REPLY$ message is same as its own, then it compares

the values of $id(BT)$, if the value of $id(BT)$ in the $REPLY$ message is larger than its own $id(BT)$, it updates its $id(BT)$, $BN1$, and $BN2$ to the values in the $REPLY$ message, it also stores the id of the sender in its $NEXT$ variable. If the value of $id(BT)$ in the $REPLY$ message is same as its own $id(BT)$, then it acts as follow: 1) If $id(T_i) > id(BT)$, it compares the value of $BN1$ in the $REPLY$ message to its $BN1$, if the value of $BN1$ in the $REPLY$ message is larger than its own, it updates its $BN1$ and $BN2$ to the values in the $REPLY$ message, it also stores the id of the sender in its $NEXT$ variable. 2) If $id(T_i) < id(BT)$, it compares the value of $BN2$ in the $REPLY$ message to its $BN2$, if the value of $BN2$ in the $REPLY$ message is larger than its own, it updates its $BN1$ and $BN2$ to the values in the $REPLY$ message, it also stores the id of the sender in its $NEXT$ variable. In the above two steps we prevent any cycle by preventing two neighboring trees from selecting two different edges of the same weight as bridge-edges between the two trees.

- After an internal node u has received $REPLY$ messages from all its children and considered all its adjacent trees, it sends a $REPLY(u, BEEW, id(BT), BN1, BN2)$ message to its parent. This process is repeated until the root receives $REPLY$ messages from all its children.

- After the root receives $REPLY$ messages from all its children, and considers all its adjacent trees to determine the $id(BT)$ and $BEEW$, it acts as follow: 1) If BT is adjacent to the root, and $BN1$ is the root itself, then, the root sends a $BRIDGE(BN1, BN2)$ message to $BN2$ in BT . If the edge between $BN1$ and $BN2$ is not a unique edge (selected by both trees) then the tree T_i rooted at $BN1$ becomes a subtree with the parent $BN2$. Otherwise if the edge between $BN1$ and $BN2$ is a unique edge, then the node with the largest id ($BN1$ or $BN2$) becomes the root of the new tree. 2) If BT is adjacent to the root, and $BN1$ is *different* from the root, or if BT is not adjacent to the root, then the root sends a $REQUEST(NEXT, BN1, id(BT))$ message addressed to the child that is stored in its $NEXT$.

- If a node receives a $REQUEST$ message from its parent addressed to itself, it acts as follow: 1) If the value of $BN1$ is the same as its own id , it sends a $BRIDGE(BN1, BN2)$ message to $BN2$ in BT . 2) If the value of $BN1$ is different from its own id , it replaces the value of $NEXT$ in the $REQUEST$ message with the value in its own $NEXT$ variable, and forwards the $REQUEST$ message to the child with the same id as the value in its $NEXT$ variable. This process is repeated until the $REQUEST$ message gets to its destination $BN1$, then $BN1$ sends a $BRIDGE(BN1, BN2)$ message to $BN2$ in BT .

- Whenever a node v in BT receives a $BRIDGE(BN1, BN2)$ message from node u , and sends a similar message to node u , then the edge (u, v) is a unique Bridge-edge, and the node with the largest id (u or v) becomes the root of a new tree that consists of all the trees in the new component. If the bridge is not unique, then node u becomes the child of node v , and all nodes in T_i will update their parents and children accordingly. All bridges in the new

component become edges in the new tree.

–The root of each new tree increments r by 1, then it performs the above steps starting at the first step of phase II until the root i of a tree T receives $\text{REPLY}(u, \text{NIL}, 0, \text{NIL}, \text{NIL})$ messages from all its children, and all the nodes in the neighborhood of the root i are part of T_i , then i declares itself as the root of the QoS-tree, and hence the leader of the graph also. All internal nodes of the tree form the virtual backbone (connected dominating set).

Example: We use the example of Phase I in Figure 1 to illustrate the procedures of Phase II. In Figure 1, phase-I generated the four trees T_3 , T_5 , T_9 , and T_{12} rooted at the nodes 3, 5, 9, and 12 respectively. In the first round each of the roots 3, 5, 9, and 12 sends a SEARCH message to all the nodes in its tree. In response to the SEARCH message of the first round T_3 and T_5 join each other through the bridge edge $1 \leftrightarrow 4$, forming the tree T_4 rooted at the node 4 and consists of the nodes 1, 2, 3, 4, 5, and 6. T_9 and T_{12} join each other also through the bridge edge $9 \leftrightarrow 12$, forming the tree T_{12} rooted at the node 12 and consists of the nodes 7, 8, 9, 10, 11, and 12 (see Figure 2 (a)). In the second round each of the roots 4 and 12 sends a SEARCH message to all the nodes in its tree. In response to the SEARCH message of the second round T_4 and T_{12} join each other through the bridge edge $9 \leftrightarrow 6$, forming the tree T_9 rooted at the node 9 and consists of the nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12, rooted at the node 9 (see Figure 2 (b)). The resulting tree is the QoS tree with the root node 9, the leaves 3, 7, and 10, and all internal nodes (1, 2, 4, 5, 6, 8, 9, 11, 12) form the VBB.

A. Correctness and Complexity

In this section, we prove that the above algorithm creates a QoS tree, and each of the time and message complexity is $O(n \log n)$.

Theorem 13: The message complexity of the QoSRT Algorithm is $O(n \log n)$ and the time complexity is $O(n \log n)$.

Proof: In the first phase each node sends a constant number of messages, and in the worst case only one node can transmit at a time. Thus both of the message and the time complexity for Phase I is $O(n)$. In Phase II, in each round each tree is combined with at least one other tree, this implies the number of trees in each round is reduced by at least one half. Thus, the maximum number of rounds is at most $\log n$. In each round each node sends a constant number of messages with a total of $O(n)$ messages. Therefore, the total number of messages in $\log n$ rounds is $O(n \log n)$ messages. Since the time complexity in each round depends on the size of the largest tree, and since there is no limit on how large the tree can grow, the time complexity for each round is $O(n)$, and hence for $\log n$ rounds is $O(n \log n)$. Hence for both phases combined each of the message complexity and time complexity is $O(n \log n)$. ■

Theorem 14: Given a weighted graph $G(V, E)$, the graph $G'(V, E')$ formed by the QoSRT Algorithm is a QoS Routing Tree for the graph $G(V, E)$.

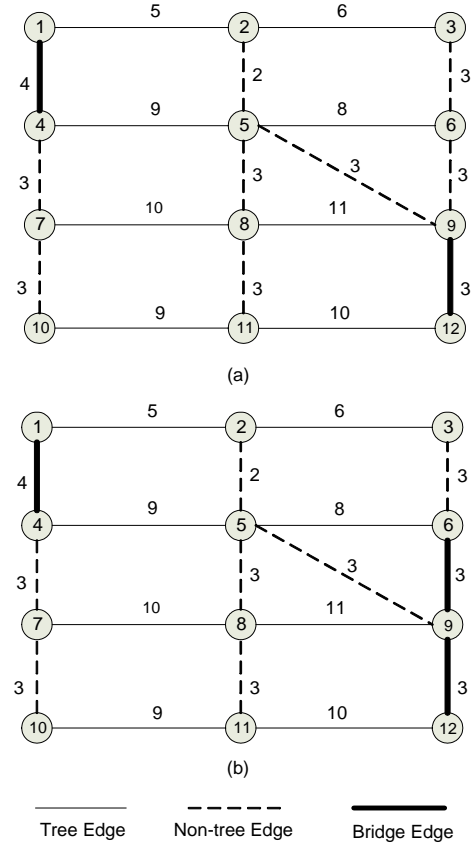


Fig. 2. Phase II Example

Proof: Following the proof of Theorem 12, in each round we treat each tree as a node with the tree id and the edges between trees as edges between nodes. To address the case when more than one edge exist between two neighboring trees, Phase II prevent the two trees from selecting two different edges. Rounds are repeated as long as there is more than one tree, and terminated when only one tree includes all nodes in the graph. ■

IV. ROUTING

After the construction of the QoS tree, the routing process becomes very simple. The QoS tree supports a unique path between any pair of nodes in the network, and this path guarantees the maximum possible bandwidth for the bottleneck link.

In our routing protocol only the VBB of the QoS tree maintain routing table information. None of the leaf nodes is required to maintain any routing table. The routing table of each internal node has one row for each child. Each row has two entries, the first entry has the child *id* and the second entry contains a list of child's descendants. Table I shows the routing table for node 2 in Figure 3. The solid lines in Figure 3 represent the edges of the QoS tree, while the dashed lines indicate the non-tree edges. Routing tables are built as follow: the parent of each leaf node creates a descendant set,

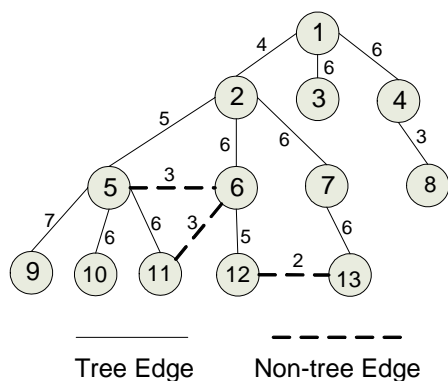


Fig. 3. Routing example

Child	Descendant
5	9, 10, 11
6	12
7	13

TABLE I
ROUTING TABLE FOR NODE 2

which includes a list of all its children. It also sends this set to its parent. Whenever a node receives a descendant set from its child, it saves this set in the table as a parameter associated with its child. Whenever an internal node receives descendant sets from all its children, it sends to its parent its own descendant set, which consists of all its children and their corresponding descendant sets. This process continues till the root receives descendant sets from all its children. After this process is completed, each node in the tree has a list of all its descendants. Since leaf nodes have no children, this information is not maintained.

In a proactive protocol after the routing tables are built, the routing process works as follow: Whenever a node needs to send or forward a message, if the destination is one of its neighbors, and the bandwidth requirement is met, the message is sent directly to the destination. Otherwise, the node checks its table to see if the destination is one of its descendants, if it is, it sends the message to the corresponding child. Otherwise, the message is up warded to the parent of the node. The parent of the node plays the role of the gateway in this case. Notice the message is forwarded in a unicast manner. Whenever a message encounters a link with a below requirement bandwidth, a FAIL message is sent to the original source of the message. Otherwise, this process continues until the message gets to its destination.

In Figure 3, if node 11 needs to send a message to its neighbor node 6 and the link $11 \leftrightarrow 6$ satisfies the bandwidth requirement, even though it is not a link in the tree, the message is sent directly over the link $11 \leftrightarrow 6$. If node 11 needs to send a message to node 13, since node 13 is not a neighbor to node 11, and node 11 is a leaf and has no table, node 11 upwards the message to its gateway node 5. Similarly, node 13 is not a neighbor to node 5, and it is not a descendant of

node 5, node 5 upwards the message to its gateway node 2. Since node 13 is not a neighbor to node 2, node 2 inspects its table and finds node 13 as a descendant of its child node 7, thus it forwards the message to node 7. Similarly, node 7 forwards the message to node 13. In this example we made the assumption that bandwidth requirement is met by all the links. However, if bandwidth requirement is not met by any of the links, a FAIL message will be sent back to node 11.

Notice, only internal nodes are responsible for routing process, and only edges of the QoS Tree are used to carry the data messages, unless in some cases when the recipient is a neighbor to the sender. The cost to build the routing tables is $\sum_{i=1}^{n-1} i = O(n^2)$ messages, and the time it takes to transmit these messages (we ignore the local processing time, and count each transmission as one time unit) is $O(n)$.

V. CONCLUSIONS

In this paper we provided a distributed algorithm to construct a QoS routing tree. Our quality of service routing tree has the following unique properties: (1) Provides an optimal maximum bandwidth path between any pair of nodes (2) Provides a connected virtual backbone, which consists of a subset of the QoS tree nodes (3) Each of the time and message complexity of the QoS tree construction is $O(n \log n)$. In addition to the QoS tree with the above mentioned features, we also proposed a proactive routing technique to run on the QoS tree. Proofs of the properties and correctness of the approaches are provided.

REFERENCES

- [1] E. Royer and T. Chai-Keong, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", *IEEE Personal Communications*, vol. 6, pp. 46–55, 1999.
- [2] Z. Demetrios, "A Glance at Quality of Service in Mobile Ad-Hoc Networks", Tech. Rep., University of California, 2001.
- [3] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: a Core-Extraction Distributed Ad Hoc Routing Algorithm", *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1454–1465, 1999.
- [4] H. Badis and K. Agha, "Quality of Service for Ad Hoc Optimized Protocol (QOLSR)", *Internet-Draft by IETF MANET Working Group*, 2005.
- [5] M. Ayyash, D. Ucci, K. Alzoubi, and R. Tandukar, "RQoS: a Robust Quality of Service Routing Algorithm for Wireless Mobile Ad Hoc Networks", in *Proceedings of the Sixth IEEE International Conference on Electro/Information Technology (EIT'06)*, 2006.
- [6] M. Ayyash, K. Alzoubi, and Y. Alsou, "Preemptive Quality of Service Infrastructure for Wireless Mobile Ad Hoc Networks", in *Proceedings of the IEEE/ACM International Wireless Communications and Mobile Computing Conference (IWCMC'06)*, 2006.
- [7] M. Ayyash, D. Ucci, K. Alzoubi, and R. Tandukar, "Robust Quality of Service Backbone for Mobile Ad Hoc Networks", in *Proceedings of the IEEE Military Communications Conference (MILCOM2005)*, October 2005.
- [8] K. M. Alzoubi, "Connected Dominating Set and its Induced Position-less Sparse Spanner for Mobile Ad Hoc Networks", in *The 8th IEEE Symposium on Computers and Communications (ISCC'2003)*, 2003.
- [9] K. Alzoubi, X. Li, Y. Wang, P. Wan, and O. Frieder, "Geometric Spanners for Wireless Ad Hoc Networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, 2003.
- [10] K. Alzoubi and M. Ayyash, "Maintaining a Quality of Service Routing Tree for Mobile Ad Hoc Networks", in *Proceedings of the IEEE/ACM International Wireless Communications and Mobile Computing Conference (IWCMC'06)*, 2006.