

QoSRT: a 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
Chicago State University
Chicago, IL 60628
mayyash@csu.edu

June 9, 2008

Abstract

Given the weighted graph $G(V, E)$, where weights represent links bandwidths. We define the Quality of Service Routing Tree (QoSRT) for G as a subgraph $G^t(V, E^t)$ such that G^t is a tree, and for any pair of nodes u and v in G^t there exist a path between u and v , where all intermediate edges form a subset of E^t , and the bottleneck of the path is maximized. In this paper, we propose a distributed algorithm to construct a QoSRT that adapts quickly to changes in links bandwidths. The QoSRT is uniquely constructed for maintenance purposes, so a change to a link s bandwidth is handled within the local subtree. Then we propose both reactive and proactive routing schemes based on the QoSRT. A virtual backbone (VBB) which consists of a subset of the nodes is responsible for routing tables in the case of proactive routing. The QoSRT and its VBB are constructed using $O(n \log n)$ messages in $O(n \log n)$ time.

1 Introduction

Wireless ad hoc networks have become a rapidly growing field. Applications of these networks occur in situations such as emergency search-and-rescue operations, meetings or conventions in which users wish to quickly share information, and data acquisition operations in hostile terrain. In situations like battlefields or major disaster areas, ad hoc networks 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 [16].

The evolution of wireless ad hoc networks and real time applications introduces new challenges in supporting predictable and reliable communication performance. These challenges are a consequence of the vastly increasing number of current and future multimedia products that find applications in wireless ad hoc networks.

A common service model in conventional networks is the best effort service model. The best effort model is not suitable for real-time multimedia applications because it drops packets regardless of their importance and adds extra delay due to retransmissions. As a result of the rising popularity of real-time multimedia applications such as video on demand (VoD) and video conferencing and because of the potential commercial usage of ad hoc networks, many researchers focused their interest on supporting a new service model. This is the Quality of Service (QoS) architecture model [12][17]. Such a model concentrates on providing high-quality media services by meeting specific QoS requirements of applications [10].

QoS can be estimated and specified in terms of several metrics that are of prime importance to the application under consideration. These parameters are used to express the applications requirements that must be guaranteed by the underlying network. Typical QoS metrics are available bandwidth, delay, jitter, tolerable packet loss rate and/or number of hops, and path reliability [11]. While gaining more interest, QoS support is yet to become a common reality and is still relatively uncharted territory especially for multimedia applications.

One of the practices that leads to performance degradation in networking, in general, is the practice of broadcasting information globally throughout the network. A simple method of achieving global broadcasting in wireless ad hoc networks is by flooding. Unfortunately, blind flooding leads to serious problems such as heavy contention, intense collisions, and redundant rebroadcasts. Such practices cause the so-called broadcast storm problem bringing disastrous consequences and must be avoided in wireless ad hoc networks [19].

Generally, the QoS problem is intricate [15]. When it comes to wireless ad hoc networks, the problem becomes more complicated [12][13][14]. The added complexity follows logically because of the special characteristics of the wireless ad hoc networking platform and the continuous monitoring of the QoS metrics. This monitoring must be continuously sustained causing high overhead [8].

Even though the QoS problem suffers from these obstacles, some promising research on QoS routing in ad hoc networks has been done. Examples of these algorithms are: Core-Extraction Distribution Ad Hoc Routing (CEDAR) [18] and Quality of Service for Ad Hoc Optimized Routing Protocol (QOLSR) [7]. Y. Ge et al. in [?] provided a proactive QoS aware routing protocol. This protocol is based on the OLSR protocol [9], and hence it inherits the complexity of the OLSR.

This paper devises a new distributed algorithm for constructing a QoS Routing Tree (QoSRT), which is designed to serve as a QoS virtual backbone (QoS-VBB) for packets routing. Advantages of QoS-VBB approach were discussed in [6]. Several approaches have been proposed in the literature for constructing VBBs for wireless ad hoc networks [4], [2]. Connected dominating sets [5], and weakly connected dominating sets [3] were proposed for VBBs. VBBs were presented for multipath routing in wireless ad hoc networks [1], but non of these approaches addressed the QoS issue.

The unique construction approach of our QoSRT has valuable properties that added the following features to the tree: (1) Guarantees the optimal max-

imum bandwidth path between any pair of nodes (2) Our QoSRT easily adapts to bandwidth changes (3) The maintenance process is locally completed in the corresponding subtree (4) Whenever an edge is promoted to the QoSRT, exactly one edge is demoted from the corresponding subtree only (5) A connected VBB consists of a subset of the QoSRT nodes is responsible for routing information maintenance (6) This QoSRT is practical for both reactive and proactive routing (7) The time and message complexity for its construction is $O(n \log n)$. In addition to the QoSRT with the above mentioned features, we propose a maintenance approach to maintain the tree in response to bandwidth changes. We also propose both a proactive and reactive routing techniques based on the QoSRT.

The rest of the paper is organized as follows. Section 2 introduces the network model and presents the definitions and terminologies related to QoSRT. Section 3 describes the QoSRT algorithm, its analysis and correctness discussion. In Section 4, we present potential routing approaches over the QoSRT. Section 5 presents the maintenance of the QoSRT in response to bandwidth changes. Finally, we conclude and highlight future directions in Section 6.

2 Preliminaries

This section introduces the related network model, assumptions, and definitions. The network model 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 if and only if the distance between the two corresponding nodes is less than or equal to the maximum transmission range. Such two nodes can communicate with each other directly. The weight of the edge represents the available bandwidth of the corresponding link. 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 sub-graph $G(V, E')$ 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

Let 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 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 $v_k < v_j$ 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 $u_k < u_i$ 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 BN_1 and the node v by BN_2 .

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 $v_k < v_j$, 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 $v_k < v_j$, 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 $u_k < u_i$, 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 $u_k < u_i$, and u_k is a neighboring set of $\max(v_j)$. This shows that the selected edge is the same by both trees. ■

3 QoSRT Algorithm

This algorithm consists of two phases. 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 will be 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 the highest level leader, and the closest leader as the lowest level leader. The bottleneck of the path between any two nodes is maximized, thus the tree is an optimal QoSRT.

PHASE I

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:

- 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.
- A set of connected edges form a single component.
- 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), in the first component, since the edge e_{12} 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 e_{23} 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 Given a weighted graph $G(V, E)$ each edge $u-v$ of any component (V_i, E_i) 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. ■

Lemma Given a weighted graph $G(V, E)$ any component (V_i, E_i) 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

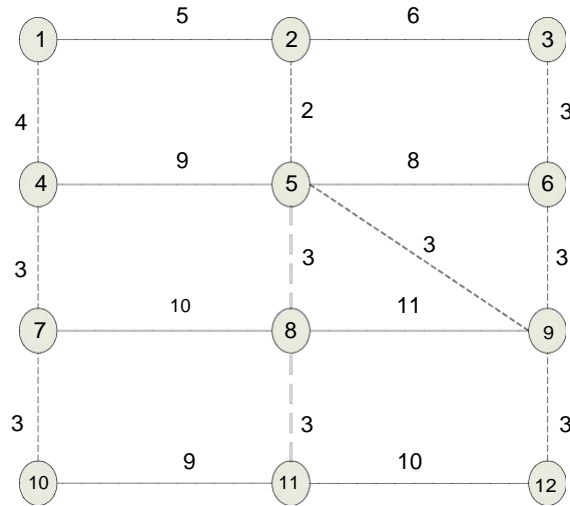


Figure 1: Phase I Example

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 $j < i$, this also implies the edges $u_1 \rightarrow u_2, u_2 \rightarrow u_3, \dots, u_{i-1} \rightarrow u_i$ 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 , $\text{weight}(u_i \rightarrow u_1)$ must be greater than $\text{weight}(u_{i-1} \rightarrow u_i)$. Since u_1 selected the edge $u_1 \rightarrow u_2$ as its best edge, then $\text{weight}(u_1 \rightarrow u_2) > \text{weight}(u_1 \rightarrow u_i)$, but from the ordering of the edges we know that $\text{weight}(u_{i-1} \rightarrow u_i) > \text{weight}(u_1 \rightarrow u_2)$ which implies $\text{weight}(u_{i-1} \rightarrow u_i) > \text{weight}(u_i \rightarrow u_1)$, 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 (V', E') formed by phase- 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 1 Given a weighted graph $G(V, E)$ and the connected component (V', E') formed by phase-1. The path between any two nodes u and v from the component must include the edge that corresponds to the $\min(\text{weight}(BE_u), \text{weight}(BE_v))$

Proof. Since (V', E') 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 $\text{weight}(BE_u) > \text{weight}(BE_v)$. Assume $BE_v \neq n_{j-1} \rightarrow n_j$, this implies the edge $n_{j-1} \rightarrow n_j$ was selected by the node n_{j-1} . To guarantee the connectivity of the path then for each $i < j$, $BE_i = n_i \rightarrow n_{i+1}$, which implies $\text{weight}(n_1 \rightarrow n_2) < \dots < \text{weight}(n_{i-1} \rightarrow n_i) < \text{weight}(n_i \rightarrow n_{i+1}) < \dots < \text{weight}(n_{j-1} \rightarrow n_j)$. Therefore, $\text{weight}(BE_u) < \text{weight}(BE_v)$, which contradicts our claim $\text{weight}(BE_u) > \text{weight}(BE_v)$. 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 (V', E') formed by phase-1. The weight of the bottleneck edge on the path between any two nodes u and v from the component $= \min(\text{weight}(BE_u), \text{weight}(BE_v))$

Proof. Let the sequence $u = n_1, n_2, \dots, n_i, n_{i+1}, \dots, n_j = v$ represents the path from u to v . Assume the bottleneck edge is the edge $n_i \rightarrow n_{i+1}$, where $i < j - 1$. By Lemma 7, the edge $n_i \rightarrow n_{i+1}$ is selected by either n_i or n_{i+1} or by both. If the edge is selected by n_i , then $\text{weight}(n_{i-1} \rightarrow n_i) < \text{weight}(n_i \rightarrow n_{i+1})$. Since the edge $n_{i-1} \rightarrow n_i$ 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 $\text{weight}(n_{i+1} \rightarrow n_{i+2}) < \text{weight}(n_i \rightarrow n_{i+1})$, which contradicts our assumption. If the edge is selected by both n_i and n_{i+1} , then $\text{weight}(n_{i-1} \rightarrow n_i) < \text{weight}(n_i \rightarrow n_{i+1}) > \text{weight}(n_{i+1} \rightarrow n_{i+2})$, 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 $\min(\text{weight}(n_1 \rightarrow n_2), \text{weight}(n_{j-1} \rightarrow n_j))$.

Theorem 12 Given a weighted graph $G(V, E)$ and a connected component (V', E') formed by phase-1. 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 \mathcal{P} . By Lemma 11 the edge BE_v is the bottleneck edge on the path from u to v over \mathcal{P} . 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 \mathcal{P} , it follows that this path is the QoS path. ■

PHASE II

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 1 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 maintains the variables: $id(BT)$, BN , $BN2$, $NEXT$, and they are initialized to NIL , and the variable $BEEW$, which is initialized 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 0. Round number (r) is used to synchronize the $SEARCH$ messages among neighboring trees.
- 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 $REPLY(u, id(BT), BEEW, BN, BN2)$ message, where $BN = u$ 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 $id(BT)$, $BEEW$, BN , 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 $BEEW$, $id(BT)$, BN

and BN_2 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 $id(BT)$, BN_1 , and BN_2 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:

- If $id(T_i) > id(BT)$, it compares the value of BN_1 in the REPLY message to its BN_1 , if the value of BN_1 in the REPLY message is larger than its own, it updates its BN_1 and BN_2 to the values in the REPLY message, it also stores the id of the sender in its NEXT variable.
- If $id(T_i) < id(BT)$, it compares the value of BN_2 in the REPLY message to its BN_2 , if the value of BN_2 in the REPLY message is larger than its own, it updates its BN_1 and BN_2 to the values in the REPLY message, it also stores the id of the sender in its NEXT variable.

In this procedure 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. If two trees have more than one edge with the same maximum weight to join each other, only one edge will be selected by both trees as a bridge-edge.

- 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, $id(BT)$, BN_1 , BN_2) 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:
 - If BT is adjacent to the root, and BN_1 is the root itself, then, the root sends a BRIDGE(BN_1 , BN_2) message to BN_2 in BT . If the edge between BN_1 and BN_2 is not a unique edge (selected by both trees) then the tree T_i rooted at BN_1 becomes a subtree with the parent BN_2 . Otherwise if the edge between BN_1 and BN_2 is a unique edge, then the node with the largest id (BN_1 or BN_2) becomes the root of the new tree.
 - If BT is adjacent to the root, and BN_1 is different from the root, or if BT is not adjacent to the root, then the root sends a REQUEST(NEXT, BN_1 , $id(BT)$) 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:
 - If the value of BN is the same as its own id, it sends a $BRIDGE(BN1tBN)$ message to $BN 2$ in BT .
 - If the value of BN is the 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 BN , then BN sends a $BRIDGE(BN1tBN)$ message to $BN 2$ in BT .
- Whenever a node v in BT receives a $BRIDGE(BN, BN 2)$ 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 the root of T_i , 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 $REPLY(utN tOtN tN)$ messages from all its children, and all the nodes in the neighborhood of the root i are children of 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 $1 \rightarrow 4$, 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 $9 \rightarrow 12$, 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 $9 \rightarrow 6$, 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 QoSRT 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.

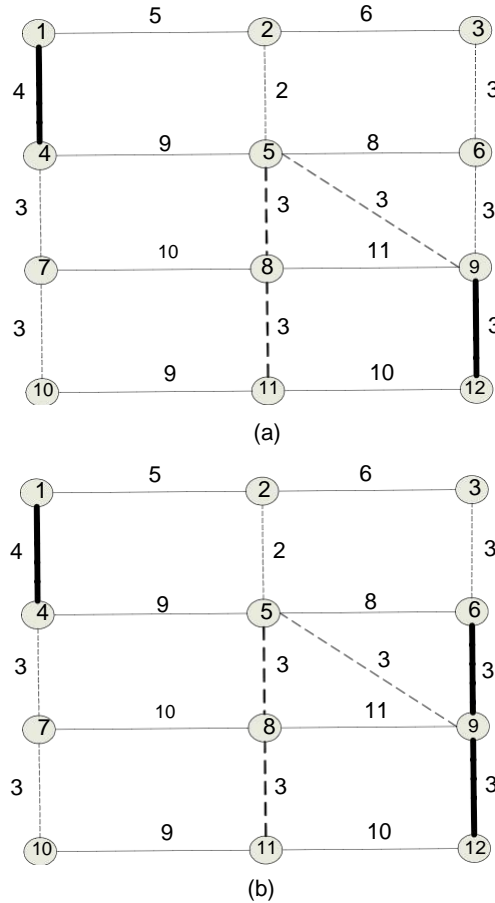


Figure 2: Phase II Example

3.1 Correctness and Complexity

In this section, we prove that the above algorithm creates a QoSRT, 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 $G = (V, E, w)$ the graph $G' = (V, E')$ formed by the QoSRT Algorithm is a QoS Routing Tree for the graph $G = (V, E, w)$

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. ■

4 Routing

After the construction of the QoSRT, the routing process becomes very simple. The QoSRT supports a unique path between any pair of nodes in the network, and this path guarantees the maximum possible bandwidth for the bottleneck link. Either a proactive or reactive routing protocol may be applied.

4.1 Proactive Routing

In a proactive routing protocol only the VBB (internal nodes and the root) of the QoSRT 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 1 shows the routing table for node 2 in Figure 3. The solid lines in Figure 3 represent the edges of the QoSRT, 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, 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

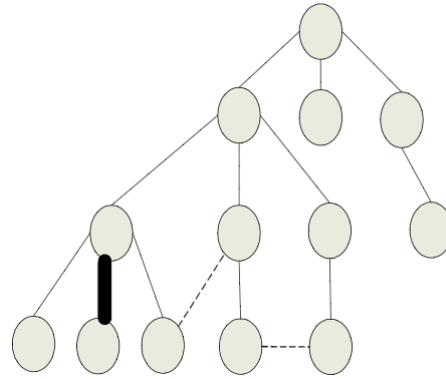


Figure 3: Routing example

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

Table 1: Routing table for node 2

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 5 needs to send a message to its neighbor node 6 and the link $5 \rightarrow 6$ satisfies the bandwidth requirement, even though it is not a link in the tree, the message is sent directly over the link $5 \rightarrow 6$. If node 5 needs to send a message to node 3, since node 3 is not a neighbor to node 5, and node 5 is a leaf and has no table, node 5 upwards the message to its gateway node 2. Similarly, node 3 is not a neighbor to node 2, and it is not a descendant of node 2, node 2 inspects its table and finds node 3 as a descendant of its child node 7, thus it forwards the message to node 7. Similarly, node 7 forwards the message to node 3. 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 5.

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 $O(n)$.

4.2 Reactive Routing

In a reactive routing protocol no routing tables are needed. Let u be the id of the source node, and let v be the id of the destination node. Four types of messages are used to find the QoS path: 1) ROUTE-DISCOVERY broadcast message, which consists of four parameters: message-id, sender-id, destination-id, and the bandwidth-requirement. 2) FAIL unicast message, which consists of three parameters: message-id, sender-id, and destination-id. 3) FOUND unicast message, which consists of three parameters: message-id, sender-id, and destination-id. 4) FAILURE unicast message, which consists of three parameters: message-id, sender-id, and destination-id. All four messages are identified by the message id. The destination-id in FAIL, FAILURE and FOUND messages is the id of the next recipient in the unicast message. While, the destination-id in ROUTE-DISCOVERY message is the id of the final destination, which is node v . Even though, the ROUTE-DISCOVERY message is a broadcast message, but it gets processed only after the first time it is received from either a parent or a child node only. Otherwise the message is discarded. Each node maintains a variable called message-id, and the associated variables sender and neet forming the record [message-id, sender, neet]. The protocol works as follow:

- Whenever a source node u needs to send a message to a destination node v , if v is u 's neighbor and the bandwidth-requirement of the link $u \rightarrow v$ is met, the message is sent directly to the destination v even if the link $u \rightarrow v$ is not part of the QoSRT.
- If the destination v is not u 's neighbor, or if it is u 's neighbor, but the link $u \rightarrow v \notin \text{QoSRT}$, and $u \rightarrow v$ does not satisfy the bandwidth-requirement, u sends a ROUTE-DISCOVERY message to its children and to its parent if at least one of them exist and satisfies the bandwidth-requirement.
- Whenever a node w receives a ROUTE-DISCOVERY message from one of its neighbors in the tree over a link that does not satisfy the bandwidth-requirement, w sends a FAIL message to its sender. If the link satisfies the bandwidth-requirement, w stores the message id in message-id, and the sender id in sender, then it acts as follow: 1) If v is w 's neighbor and the bandwidth-requirement of the link $w \rightarrow v$ is met, w stores the id of v in neet, then it sends a FOUND message to sender even if the link $w \rightarrow v$ is not part of the QoSRT. 2) If v is not w 's neighbor or if v is w 's neighbor, but the link $w \rightarrow v$ is not part of the QoSRT, and the bandwidth-requirement of the link $w \rightarrow v$ is not met, w forwards the ROUTE-DISCOVERY message to the rest of its neighbors in the tree if at least one of them ex-

ist and meets the bandwidth-requirement. 3) If v is w 's neighbor in the QoSRT, and the bandwidth-requirement of the link $w \rightarrow v$ is not met, then w sends a FAILURE message to its sender (this means the destination is unreachable with the bandwidth-requirement). 4) If all links with w in the QoSRT except the link with the sender have a lower bandwidth than the bandwidth-requirement, then w sends a FAIL message to its sender.

- Whenever a leaf node (cannot be the destination) receives a ROUTE-DISCOVERY message from its parent, it sends a FAIL message to its sender.
- Whenever a node receives a FAIL message from each of its neighbors in the QoSRT except the sender of the ROUTE-DISCOVERY, it sends a FAIL message to its sender.
- If the source u receives FAIL messages from its parent and from all its children, it determines the unreachability of the destination.
- Whenever a node receives a FAILURE message from a neighbor in the tree, it sends a FAILURE message to its sender. Whenever the root receives a FAILURE message from one of its children, it determines the unreachability of the destination.
- Whenever a node receives a FOUND message regarding a stored message-id, it stores the sender id in the variable $next$ that corresponds to the message-id, then it forwards this message to the node with the id stored in $sender$ (the sender of the ROUTE-DISCOVERY message).
- Whenever the source node u receives a FOUND message, it stores the id of the sender in $next$, then it sends the packet (message) to $next$.
- Whenever a node receives a packet addressed to itself, it sends it to the node with the id stored in $next$. This process continues until the packet is received by the destination.

Each internal node sends at most one ROUTE-DISCOVERY message, and at most one of the three messages FAIL, FAILURE, or FOUND. Thus, we have at most $O(n)$ messages for the route discovery process, and this is done in at most $O(n)$ time. Notice the size of the message is small, which keeps the bandwidth consumption very low, and the delay very reasonable. On the other hand, proactive approach requires more memory to store the tables, incurs more bandwidth consumption, and extra time to search the tables.

Now, we provide an example to show how the algorithm works. In Figure 3, if node 11 needs to send a message with a bandwidth-requirement of 4 to node 8, node 11 sends a ROUTE-DISCOVERY message, the message will be received by nodes 5 and 6. However, node 6 ignores the message, since the edge $6 \rightarrow 5$ is a non-tree edge. Node 5 processes the message, by storing the message-id, and the id of the sender in the corresponding variables. Since node 8 is not a neighbor to node 5, and since there is at least one link between 5 and its parent

or between 5 and one of its children other than node 11, and this link satisfies the bandwidth-requirement, node 5 broadcasts the ROUTE-DISCOVERY message. The ROUTE-DISCOVERY message will be received by nodes 2, 6, 9, 10, and 11. Since the edge 5-6 is a non-tree edge, node 6 ignores the message. Since the message-id in the ROUTE-DISCOVERY message is same as the message-id stored by node 11, this means the same message was sent previously by node 11. Hence, node 11 ignores the message also. Each of nodes 9 and 10 will process the message, and since it is leaf node, and it is not the destination, it sends a FAIL message to node 5. Node 5 maintains the FAIL messages and wait for answers from node 2. Node 2 broadcasts the ROUTE-DISCOVERY message. The ROUTE-DISCOVERY message will be received by nodes 5, 6, 7, and 1. Node 5 ignores the message. Node 2 receives the message and broadcast it. Both of nodes 6 and 7 receive the message and broadcast it. Each of nodes 12 and 13 send a FAIL message to its sender. Then each of nodes 6 and 7 send a FAIL message to its sender. After node 1 receives the ROUTE-DISCOVERY message it broadcasts it. Node 2 ignores the message, and node 3 sends a FAIL message to node 1. Node 4 detects that the destination 8 is one of its neighbors in the QoSRT, but since the bandwidth is less than the required bandwidth, it sends a FAILURE message to node 1. Node 1 sends a FAILURE message to node 2. Node 2 sends a FAILURE message to node 5. Node 5 sends a FAILURE message to node 11. Now, node 11 decides the unavailability of the QoS path.

In the same example if the bandwidth-requirement is 3 instead of 4. The same action is taken by all the nodes except for node 4. After node 4 receives the ROUTE-DISCOVERY message it determines that node 8 is one of its neighbors, and the bandwidth-requirement is met. Node 4 stores the id of node 8 in its neet variable, then it sends a FOUND message to sender (node 1). Node 1 stores the id of node 4 in its neet variable, then it sends a FOUND message to sender (node 2). Node 2 stores the id of node 1 in its neet variable, then it sends a FOUND message to sender (node 5). Node 5 stores the id of node 2 in its neet variable, then it sends a FOUND message to sender (node 11), which is the source of the ROUTE-DISCOVERY message. Now the message can be sent over the QoS path by sending the message to the node that has the id stored in neet. Whenever a node receives the message it forwards it to neet until the message gets to the destination.

If node 9 needs to send a message to node 2 with a bandwidth-requirement of 6, a ROUTE-DISCOVERY message is sent by 9. When 5 receives the ROUTE-DISCOVERY message, it determines the unreachability to its parent, and it broadcasts the ROUTE-DISCOVERY message. After it has received a FAIL message from each of 10 and 11, it sends a FAIL message to node 9.

Notice also the ROUTE-DISCOVERY message is terminated as soon as the destination is found. If node 11 needs to send a message to node 2. In this case node 5 responds to node 11 with FOUND message, instead of forwarding the ROUTE-DISCOVERY message.

5 QoSRT Maintenance

In this section we focus on the maintenance process of the QoSRT due to changes in bandwidth values. These changes may affect two types of edges; phase-I edges, which are selected by nodes in the first phase, and phase-II edges, which are selected by trees in the second phase. Phase-I edges are selected by either one node or two nodes, and Phase-II edges are selected by either one tree or two trees. It is very important to distinguish these different edges from each other. It is very important also to maintain the properties of the QoSRT. These properties are a result of how the QoSRT was constructed. To maintain the properties of the QoS Tree, each node must be connected to the tree by its best edge. Two trees that are connected by a bridge must maintain connectivity with the maximum bandwidth bridge. These properties of the QoSRT contribute to the ease and low overhead complexity of the maintenance process. Whenever a change to the bandwidth for some links affects the status of Phase-I or Phase-II edges, the update process occurs only in the same subtree where bandwidth changes take place. There are two possible scenarios when a tree needs to be updated:

First Scenario: Whenever a node u discovers a better link than its current BE, there are two cases to be addressed: 1) if the node is a child to its current BN. 2) if the node is a parent to its current BN.

Case-1 When u is a child to its current BN u' , there are two cases:

Case-1-A When BE_u is not a unique edge: if the new BN u'' is not a member in u 's subtree, it simply selects the new BN as its parent. If the new BN is a member in u 's subtree, and u'' is not a child of u , then u'' is selected as a child for u . If u'' is not a child of u , the first concern would be the expectation of a cycle. If u'' is a child of u ($u \leftrightarrow u''$ is either a bridge edge or $BE_{u''}$), then the possibility of a cycle is eliminated. Otherwise, there are two suspicious cases for the possibility of a cycle, the first case; if the former path between u'' and u does not contain any bridge edge. In this case the cycle is removed automatically. Since the former path from u to u'' does not contain any bridge edge, then all the nodes on the path belong to phase 1 of the algorithm. By Lemma 10 the former path between nodes u and u'' must include the edge that corresponds to the $\min(\text{weight}(BE_u), \text{weight}(BE_{u''}))$, and by Lemma 11 this edge is the bottleneck edge. Since BE_u is the parent of u , then BE_u is not on the former path from u to u'' , therefore, By Lemma 10 $BE_{u''}$ must be on the former path from u to u'' , and by Lemma 11 this edge must be the bottleneck edge. This implies $\text{weight}(u \leftrightarrow u'') > \text{weight}(BE_{u''})$. Therefore node u'' selects the edge ($u \leftrightarrow u''$) as its $BE_{u''}$, and the former $BE_{u''}$ edge is removed, since it was selected by u'' . Thus the cycle is removed automatically. Notice that the edge ($u \leftrightarrow u''$) becomes a unique edge. In the next step we connect the subtree rooted at node u to the rest of the tree. This process is completed based on phase II of the QoS Tree algorithm. The node u issues a SEARCH message looking for the best edge that connects its subtree to the rest of the tree. The node of the subtree that has the best edge to connect to the rest of the tree becomes the new root of the subtree. The second case if the former path between u'' and u

contains at least one bridge edge in addition to edges selected by the nodes as best edges (BE). In this case if $BE_{u''}$ is on the former path from u'' to u and $\text{weight}(u \leftrightarrow u'') > \text{weight}(BE_{u''})$ then the former edge $BE_{u''}$ is removed, and therefore the cycle is removed. Otherwise, the bridge edge with the minimum weight on the path must be removed, and nodes on the former path will update their parent and child pointers accordingly.

Case-1-B: when BE_u is a unique edge: if the new BN u'' is not a member in u 's subtree, it selects u'' as its child. The issue of the cycle is handled as in Case-1-A above. If the new BN u'' is a member in u 's subtree, and u'' is a child of u , then we simply change BE_u to the edge $(u \leftrightarrow u'')$ and the edge $(u \leftrightarrow u'')$ becomes a unique edge. If u'' is not a child of u , then u'' is selected as a child for u , and the process continues as in Case-1-A, except the step of connecting the subtree to the rest of the tree, since the subtree is not disconnected from the rest of the tree as in case-1-A.

Case- : When u is a parent to its current BN u , there are three cases:

Case- -A: When BE_u is not a unique edge: 1) If the new BN u'' is u 's parent or u 's child, and the edge $(u \leftrightarrow u'')$ is a bridge edge, it simply becomes BE_u , if the edge $(u \leftrightarrow u'')$ is $BE_{u''}$, it also becomes BE_u , and hence it becomes a unique edge. 2) If the new BN u'' is not a member in u 's subtree, then u selects u'' as its child. The issue of the cycle is handled as in Case-1-A above. Since the edge $(u \leftrightarrow u'')$ is removed from the QoSRT, we need to connect the subtree rooted at node u to the rest of the tree. This process is completed based on phase II of the QoSRT algorithm. The node u issues a SEARCH message looking for the best edge that connects its subtree to the rest of the tree. The node of the subtree that has the best edge to connect to the rest of the tree becomes the new root of the subtree. 3) If the new BN u'' is a member in u 's former subtree, then u selects u'' as its child. The rest of the nodes in the subtree update their parent and child pointers accordingly.

Case- -B: when BE_u is a unique edge: 1) If the new BN u'' is u 's parent or u 's child, we follow the same process as in case- -A except the step of connecting the subtree rooted at u to the rest of the tree, since the subtree is not disconnected from the rest of the tree. 2) If the new BN u'' is not a member in u 's subtree, it selects u'' as its child, and the rest of the process continues as in case- -A except the step of connecting the subtree rooted at u to the rest of the tree, since the subtree is not disconnected from the rest of the tree as in case- -A. 3) If the new BN is a member in u 's subtree, then u'' is selected as a child for u , and the cycle is handled as in case-1-A.

Second Scenario: We define a complete phase-1 component or complete phase-1 subtree as a tree rooted at a node connected to its parent by a bridge edge and forms a connected subtree without any bridge edges, such that the leaves of the subtree are either leaves of the QoS routing tree, or incident at at least one bridge edge. We define a phase-1 component or phase-1 subtree as a connected subtree without any bridge edges. We also emphasize the standard definition of a subtree for any given tree rooted at a given node u to be the tree rooted at the node u and includes all descendants of u . Since bridge edges connect two components, and each component has at least one edge from phase 1, then each

node that is incident at a bridge edge is a root of a sub tree. Since each bridge edge consists of two nodes, then one of the nodes is a parent to the other (child) in the QoS routing tree. This child node is referred to as a leader.

In the maintenance process we are interested in a special type of subtrees, in particular those subtrees rooted at the leader nodes. Let each leader be responsible for the connection of its subtree to the rest of the QoS routing tree. We need to maintain this connection through the best possible edge. Thus, the bridge edge must be replaced by a better edge whenever a better edge that connects the subtree to the QoS routing tree is discovered. Nodes in the QoS tree that belong to the complete phase-1 subtree rooted at the root of the QoS tree do not have a leader. Each of the nodes in the remaining QoS routing tree belong to at least one sub tree rooted at a leader node. Thus, each node may belong to multi level subtrees. The lowest level subtree is the one with the closest leader, and the highest level is the one with the furthest leader. Each subtree rooted at a given leader has the id of its leader. Each subtree node must maintain the ids of all subtrees where it belongs. It further maintains the ids of all subtrees of its neighbors. Based on the above information we propose two approaches to maintain the bridge edges. In the first approach we follow the following steps:

- Each leader sends a periodic search message with its own id to all nodes in its subtree
- Whenever a node receives the search message from its parent it looks for the best edge that connects it to another neighboring subtree, and stores the weight of that edge in a variable called bridge-weight.
- Whenever a leaf node receives the search message from its parent it looks for the best edge that connects it to another neighboring subtree, and stores the weight of that edge in a variable called bridge-weight. Then it sends a REPLY message with its id in the candidate field and the value of bridge-weight in the bridge-weight field to its parent. The candidate field has the id of the candidate node to lead the subtree.
- Whenever an internal node receives a REPLY message from its child if its bridge-weight is greater than or equal the bridge-weight in the REPLY message, it updates the REPLY message by replacing the value in the candidate field with its own id, and the value in bridge-weight field with its own bridge-weight value. This process continues until an internal node has received REPLY messages from all its children.
- After an internal node has received REPLY messages from all its children, and has considered all its neighbors from other subtrees, it sends a REPLY message to its parent.
- After the root has received REPLY messages from all its children, it selects the candidate node with the largest bridge-weight value, if it is greater

than its own bridge-weight value, it sends an UPDATE message to the new candidate node.

- Whenever the candidate node receives the UPDATE message, it selects the other node incident at the new proposed bridge edge as its parent, then it sends parent message to all its neighbors in the same subtree. Nodes in the subtree will update their parent and child pointers accordingly. The former root of the subtree updates its parent to one of its children in the former subtree, thus the former bridge edge is removed.

In the second approach, each node maintains the bridge-weight value for the bridges of all subtrees it belongs to. Whenever a node detects an edge that links its subtree to another subtree, and this edge has a larger weight than the weight of its subtree's bridge, it sends an ALERT message to its subtree leader. The ALERT message has the two fields candidate and bridge-weight. Whenever the leader receives one or more ALERT messages, it selects the one with the largest bridge-weight, then it sends to the associated candidate an UPDATE message. Whenever the candidate receives the UPDATE message it updates its child and parent pointers. The remaining nodes in the subtree will update their parent and child pointers accordingly.

6 Conclusion

In this paper we provided a distributed algorithm to construct and maintain 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) Adapts quickly to bandwidth changes, and the maintenance process is completed in the local subtree (3) Provides a connected virtual backbone, which consists of a subset of the QoSRT nodes (4) Each of the time and message complexity of the QoSRT construction is $O(n \log n)$ (5) . In addition to the QoSRT with the above mentioned features, we also proposed a maintenance approach to maintain the tree in response to bandwidth changes. Both reactive and proactive routing techniques are provided for the QoSRT. Proofs of the properties and correctness of the approaches are provided. In future work we will provide the maintenance process in mobility environment. The implementation of these approaches are currently under process.

References

- [1] K. Alzoubi and M. Ayyash, "Multispanners for Robust Routing in Mobile Ad Hoc Networks," in Parallel and Distributed Computing and Networks PDCN for the International Association of Science and Technology for Development AST D , 2004.

- [2] K. Alzoubi, X. Li, Y. Wang, P. Wan, and O. Frieder, "Geometric Spanners for Wireless Ad Hoc Networks," *Transactions on Parallel and Distributed Systems*, vol. 14, 2003.
- [3] K. Alzoubi, P. Wan, and O. Frieder, "Weakly-Connected Dominating Sets and Sparse Spanners in Wireless Ad Hoc Networks," in *CDCSt The 3rd International Conference on Distributed Computing Systems*, 2003.
- [4] K. M. Alzoubi, "Connected Dominating Set and its Induced Position-less Sparse Spanner for Mobile Ad Hoc Networks," in *The 8th Symposium on Computers and Communications SCC' 003*, 2003.
- [5] K. M. Alzoubi, P.-J. Wan, and O. Frieder, "New Distributed Algorithm for Connected Dominating Set in Wireless Ad Hoc Networks," in *1st HCSS35t Hawaii*, 2002.
- [6] M. Ayyash, D. Ucci, K. Alzoubi, and R. Tandukar, "Robust Quality of Service Backbone for Mobile Ad Hoc Networks," in *Military Conference MCOM 005*, October 2005.
- [7] H. Badis and K. Agha, "Quality of Service for Ad Hoc Optimized Protocol (QOLSR)," *Internet-Draft by IETF MANET Working Group*, 2005.
- [8] T. Chen and M. Gerla, "QoS Routing Performance in Multihop, Multimedia, Wireless Networks," in *Proceedings of CUPC'97*, 1997.
- [9] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," *IETF Mobile Ad Hoc Networks (MANET) Working Group Charter*, October 2003, www.ietf.org/rfc/rfc3626.txt.
- [10] E. Crawley, R. Nair, B. Rajagopalan, and H. Samdriick, "A Framework for QoS Based Routing in the Internet: RFC2386," *Tech. Rep.*, 1998.
- [11] M. Curado and E. Monteiro, "An Overview of Quality of Service Routing Issues," in *Proceedings of the 5th World Multiconference on Systemicst Cybernetics and Informatics SC 001*, June 2001.
- [12] Z. Demetrios, "A Glance at Quality of Service in Mobile Ad-Hoc Networks," *University of California, Tech. Rep.*, 2001.
- [13] C. Lin and J. Liu, "QoS Routing in Ad Hoc Wireless Networks," *Journal on Selected Areas in Communications*, vol. 17, pp. 1426-1438, 1999.
- [14] D. Lorenz and A. Orda, "QoS Routing in Networks with Uncertain Parameters," *ACM Transaction on Networking*, vol. 8, pp. 768-778, 1998.
- [15] A. Orda and A. Sprintson, "Precomputation Schemes for QoS Routing," *ACM Transaction on Networking*, vol. 11, pp. 578-591, 2003.

- [16] E. Royer and T. Chai-Keong, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," *Personal Communications*, vol. 6, pp. 46-55, 1999.
- [17] P. Sinha, R. Sivakumar, and V. Bharghavan, "Enhancing Ad Hoc Routing with Dynamic Virtual Infrastructures," in *Proceedings of NFOCOM 001*, April 2001, pp. 1763-1772.
- [18] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: a Core-Extraction Distributed Ad Hoc Routing Algorithm," *Journal on Selected Areas in Communications*, vol. 17, pp. 1454-1465, 1999.
- [19] Y. Tseng, S. Ni, Y. Chen, and J. Sheu, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," in *Proceedings of the 5th annual ACM/ international conference on Mobile computing and networking*, 1999, pp. 151-162.