BANDWIDTH-AWARE ROUTING TREE (BART) FOR UNDERWATER 3-D GEOGRAPHIC ROUTING

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Tae Hyun Kim

Certificate of Approval:

David Umphress
Assistant Professor
Computer Science and
Software Engineering

Min-Te Sun, Chair
Assistant Professor
Computer Science and
Software Engineering

Yu Wang
Assistant Professor
Computer Science and
Software Engineering

George T. Flowers
Interim Dean
Graduate School

BANDWIDTH-AWARE ROUTING TREE (BART) FOR UNDERWATER 3-D GEOGRAPHIC ROUTING

Tae Hyun Kim

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirement for the

Degree of

Master of Science

Auburn, Alabama
August 9, 2008

BANDWIDTH-AWARE ROUTING TREE (BART) FOR UNDERWATER

3-D GEOGRAPHIC ROUTING

Tae Hyun Kim

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon request of individuals or institutions and their expense. The author reserves all publication rights.

Signature of Author

Date of Graduation

VITA

Tae Hyun Kim, the son of Bong-Chul Kim and You-Soon Choi, was born on October 11th, 1979 in Seoul, Republic of Korea. He attended the Korean Army Academy in Young-Cheon, Republic of Korea and obtained a Bachelor of Science degree in Computer Science in March, 2003. After working in the Army Computer Department in the Republic of Korea military, he enrolled in Auburn University and joined Dr. Sun's lab in fall 2006.

THESIS ABSTRACT

BANDWIDTH-AWARE ROUTING TREE (BART) FOR UNDERWATER 3-D GEOGRAPHICAL ROUTING

Tae Hyun Kim

Master of Science, August 9, 2008
(B.S. Computer Science, Korea Army Academy, South Korea, 2003)

55 Typed Pages

Directed by Min-Te Sun

The limited bandwidth and power resources in underwater sensor networks have made geographic routing a favorite choice. While many detouring strategies in geographic routing do not work for 3-D underwater network topology, spanning tree routing can efficiently find a detour when used for such networks. However, the effectiveness of tree routing depends largely on the quality of the pre-constructed spanning tree. Most existing spanning tree algorithms build trees in a top-down and centralized fashion and do not consider the available bandwidth in the network, and may create trees with poor routing performance. In this research, we propose a novel spanning tree, named Bandwidth-Aware Routing Tree (BART), that is constructed completely in a bottom-up fashion and with available bandwidth in mind. Simulation results show that compared with other spanning trees, BART has much fewer conflicting hulls and higher path throughput, thus leads to better routing performance in 3-D underwater sensor networks.

ACKNOWLEDGMENTS

I would like to express my respect and thanks to my advisor, Dr. Min-Te Sun, for his direction, patience, and encouragement which have been invaluable during the entire time of this research. I am very grateful for the active support that Dr. David Umphress and Dr. Yu Wang provided by being on my thesis committee. I appreciate my research group members, Kazuya Sakai, Kyoung-Min Kim, and Lei Zhang, for their support and friendship. Especially, I give my sincere appreciation to my wife and sons for their love and understanding. Finally, I am thankful to my GOD for providing help, unlimited love, and powerful protection in numerous ways.

Style manual or journal used Guide to Preparation and Submission of Theses and Dissertations

Computer software used Microsoft Word 2007, Excel 2007

TABLE OF CONTENTS

LIST OF TABLES……………………………………………………………..... xi

LIST OF FIGURES…………………………………………………………... xii

CHAPTER 1. INTRODUCTION…………………………………...………... 1

CHAPTER 2. LITERATURE REVIEW……………………………………... 6

2.1 Bread-first and Depth-first Spanning Trees……………………….….. 6

2.2 Minimum Spanning Trees……………………………………….…. 8

2.3 Minimum Path Trees…………………………….…………….... 9

2.4 IEEE 802.1D……………………………………………………….. 10

2.5 Common Problems of Existing Tree Algorithms…………………… 11

CHAPTER 3. BANDWIDTH-AWARE ROUTING TREE…………………..... 12

3.1 Overview of BART………………………………………………… 12

3.2 BART Algorithm…………………………………………………… 14

3.3 Data Structure for Merging The Cluster…………………………… 17

3.4 The Gravity Function……………………………………………… 18

3.5 Concurrent Merge Requests………………………………………… 19

CHAPTER 4. ILLUSTRATION OF TREE CONSTRUCTION…………….....   20

CHAPTER 5. PERFORMANCE STUDY……………………………………...   26

5.1 Simulation Setting…………………………………………………...   26

5.2 Simulation Result and Analysis……………………………………...   29

CHAPTER 6. CONCLUSION AND FUTURE WORK………………………….   37

BIBLIOGRAPHY…………………………………………………………...   39

LIST OF TABLES

Table I. Bandwidth-Aware Routing Tree Algorithm 15

LIST OF FIGURES

Figure 2.1 Sequence of searching the node from root in DFS... 7

Figure 2.2 Sequence of searching the node from root in BFS... 7

Figure 2.3 Minimum Spanning Tree.. 8

Figure 2.4 Minimum Path Tree.. 9

Figure 3.1 Example of Clusters and The Bandwidth Between Them.............................. 14

Figure 4.1 Original Network Topology 20

Figure 4.2 Minimum Path Tree.. 21

Figure 4.3 Construction of BART - Initialization.. 22

Figure 4.4 Construction of BART - Parallel Tree Construction....................................... 23

Figure 4.5 Construction of BART – Completion... 24

Figure 5.1 Number of Conflicting Convex Hulls 30

Figure 5.2 Average Path Hops.. 31

Figure 5.3 Average Path Distance 32

Figure 5.4 Average Path Throughput.. 33

Figure 5.5 Delivery Rate.. 34

Figure 5.6 Energy Consumption.. 35

CHAPTER 1

INTRODUCTION

The research of underwater acoustic sensor networks [2, 24] has recently gained increasing attention due to their potential applications, such as ocean environment monitoring, tactical military surveillance, target tracking, and undersea navigation [9]. Underwater acoustic networks are a special type of wireless sensor network. Due to the limitation of each node’s transmission range and the possibility of nodal failure caused by the erosive underwater environment, data collected by individual sensor nodes are usually sent to a control center through a network of nodes [1]. Thus, appropriate routing algorithms are needed to direct data packets from their source to the destination.

Traditional routing algorithms in wireless sensor networks can be classified into two categories: proactive protocols and reactive protocols. In proactive protocols [21, 34], each node maintains a routing table by exchanging link state or distance vector control packets with neighboring nodes. The advantage of proactive routing is that the routing table allows for each node to quickly identify the next hop of a packet. In addition, different metrics (including hop count, available bandwidth, and congestion status) can be taken into consideration in the construction of the routing table [34]. The major disadvantage of proactive routing, however, is that the communication with nodes and storage overhead of routing table maintenance grows as quickly as the size of the network increases. In reactive protocols [11, 20], when a node has a packet to send, it floods the network with queries to discover a route to the destination. The primary difference among various reactive protocols is how the responding path information is cached at intermediate nodes and reused. While reactive routing avoids the overhead incurred in routing table maintenance, the network-wide query flooding of path discovery is an extremely expensive operation and may create additional problems, such as the infamous broadcast storm problem [29]. Therefore, while proactive and reactive routing protocols do not rely on assumptions of network topology and link characteristics, they incur heavy communication and storage overhead, which becomes a serious issue in resource-constrained underwater sensor networks.

In recent years, a new class of routing protocols - geographic routing [4, 12, 14, 15] have been proposed. Under the assumptions that each node knows the geographic locations of itself and its neighbors, as well as the source nodes, and also encodes the geographic location of the destination in the packet, geographic routing can determine where to forward the packet without maintaining a routing table or flooding the network. It makes use of localized geographical location information to route traffic, avoiding a large portion of the communication and storage overhead of proactive and reactive routing protocols [8, 9]. While GPS signal is not available under the surface of the ocean, the relative location of each underwater sensor node can be pinpointed via triangulation [18]. Compared with traditional proactive and reactive protocols, geographic routing tends to create lower overhead and has been suggested for underwater acoustic sensor network in [1, 24]. In general, geographic routing is made up of two parts - greedy forwarding and detouring strategy. If a node holding a packet finds some neighbors that are closer to the destination of the packet, the node forwards the packet to the one that is the closest to the destination. This is called *greedy forwarding*. Note that greedy forwarding alone has low delivery rate even in a connected network. When a local minimum is reached (i.e., no closer neighbor can be found), the geographic routing protocol falls back to its *detouring strategy* to find a detour to leave the local minimum and then move toward the destination. Without the help of routing tables, most existing non-flooding detouring strategies first reduce the original network topology to a planar graph by dropping some edges and then apply one of the heuristics (e.g., perimeter routing [12], face routing [14]) to explore the network via links in the planar graph. Although the criteria and performance analysis of greedy forwarding for underwater sensor networks have been studied in [22], non-flooding detouring strategies have a serious flaw that will keep them from working properly in an underwater environment. The two strong assumptions used by almost all distributed graph planarization algorithms [8, 17, 28] and their routing heuristics are the unit disk link model and the *flat* (i.e., on a 2-D plane) network topology. The unit disk link model does not apply to underwater acoustic links, which have highly dynamic physical characteristics due to time-varying multipath intersymbol interferences and Doppler shifts and spreads [25]. In addition, nodes in underwater sensor networks are mostly deployed over 3-D space at different depths and on the bottom of the ocean, which is normally not flat. These strong assumptions render most existing geographic routing protocols unusable for underwater sensor networks. In [13], a planarization algorithm is proposed that relaxes the assumption of the unit disk link model, but it introduces a large amount of signaling [15] among neighboring nodes, and its heuristic still relies on the 2-D assumption.

In [15], a novel geographic routing protocol is proposed with a detouring strategy based on a pre-constructed spanning tree. In this protocol, a spanning tree is first generated from a root node close to the boundary of the network. Each node keeps a convex hull that encloses all nodes in its subtrees. When a packet comes to a local minimum, a node checks whether the convex hull of any child node encloses the destination. If no such child exists, the packet is forwarded to the parent node. Otherwise, the packet is forwarded to the child node whose convex hull encloses the location of the destination. If multiple children meet this criterion, all these children will be explored in turn. The tree routing continues until a node closer to the destination than the last local minimum is found. Then the greedy forwarding takes over again. The spanning tree detouring strategy does not rely on any of the aforementioned assumptions and has been shown in [15] to produce comparable performance to other detouring strategies in terms of the number of hops. Notice that the effectiveness of the spanning tree-based detouring depends largely on the quality of the pre-constructed spanning tree. If the convex hulls of siblings in a spanning tree intersect each other, they are referred to as the conflicting hulls. If a tree has many conflicting hulls, a packet may need to traverse more branches before it finds the right path to the destination. Consequently, the routing performance will likely be poor. In [16], several different types of spanning trees are studied and the minimum path tree is found to perform best. However, in their study, the root of a tree is always fixed. This places a serious constraint on the construction of the spanning tree. Without knowing the locations of nodes in the tree, a bad selection of the root can easily lead to a large number of conflicting hulls. In addition, the available bandwidth of links is not taken into account in their study. If a link included in a routing tree has low available bandwidth, it will likely become a bottleneck in the network.

In this thesis, we propose the concept of the Bandwidth-Aware Routing Tree (BART), which is constructed in a bottom-up fashion based on both location and available bandwidth. Different from existing spanning trees, BART reduces the number of conflicting hulls and avoids bottlenecks in the routing tree. Simulation results show that BART results in fewer conflicting hulls and higher path throughput than other types of spanning trees. These properties are especially important for underwater sensor networks.

The rest of the thesis is organized as follows. In Chapter 2, several types of spanning trees are reviewed. In Chapter 3, the proposed BART and the algorithm to construct it are introduced. In Chapter 5, the simulation results of BART, as well as two other routing trees are presented. The conclusion and the future research direction are given in Chapter 6.

CHAPTER 2

LITERATURE REVIEW

In the spanning tree detouring strategy, data packets are delivered to their destinations along a path in the pre-constructed spanning tree topology. A spanning tree of a connected network is a tree that contains all of the nodes in the network. There are many different spanning trees for a given network. In this chapter, we review several spanning tree algorithms and discuss their applications in the routing of wireless networks.

2.1. Breadth-First and Depth-First Spanning Trees

The depth-first search (DFS) and the breadth-first search (BFS) are two principal algorithms to traverse a connected network and to create spanning trees [5].

In the DFS algorithm, the starting point of traversal becomes the root of the tree. At each step of the traversal, DFS visits neighboring unvisited nodes as deep as possible until no such node is available, as described in Figure 2.1. The arrow dotted line means the traversal from root to build the tree. Whenever a new, unvisited node is reached for the first time, it is attached as a child to the node from which it is being reached. If there are multiple such unvisited nodes, a tie can be resolved arbitrarily. This process continues until a dead end (i.e., a node without an adjacent unvisited node) is encountered.

At a dead end, the tree construction method backs up one link to the node where it came from and tries to continue visiting unvisited nodes from there. Eventually, it halts after backing up to the starting node, with later will become a dead end. By then, all the nodes within the same connected components as the starting node have been visited.

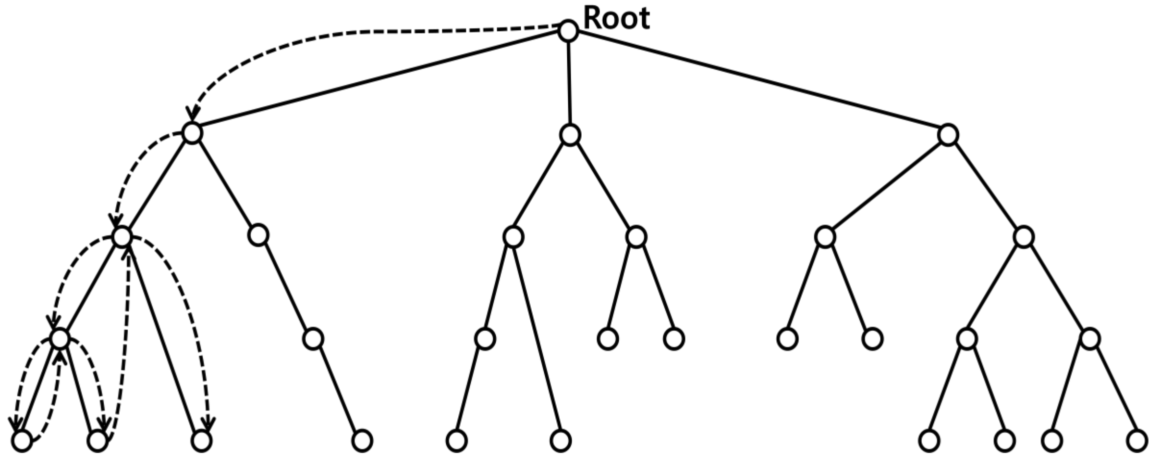

Figure 2.1 Sequence of searching the node from root in DFS

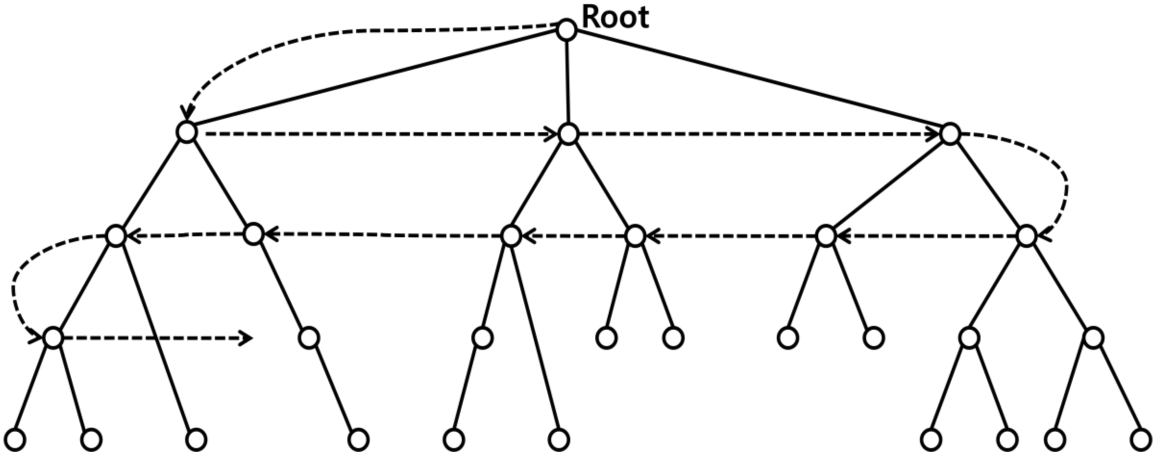

Figure 2.2 Sequence of searching the node from root in BFS

The BFS algorithm, on the other hand, visits the neighboring unvisited nodes without limitation until no such node is available. In the BFS algorithm, the starting point of traversal becomes the pre-constructed root of the tree. It proceeds in a concentric manner by first visiting all the nodes that are adjacent to the starting node, then all unvisited nodes two links apart from it, and so on, until all the nodes in the same connected component as the starting node are visited, as shown in Figure 2.2. Compared with Figure 2.1, we can clearly see the different sequence of search.

While DFS seeks to go as far as it can, BFS tries to exhaust the neighborhood first. The results of these traversals are the DFS trees and the BFS trees. Both DFS and BFS trees provide a route to reach every node in the network.

2.2. Minimum Spanning Trees

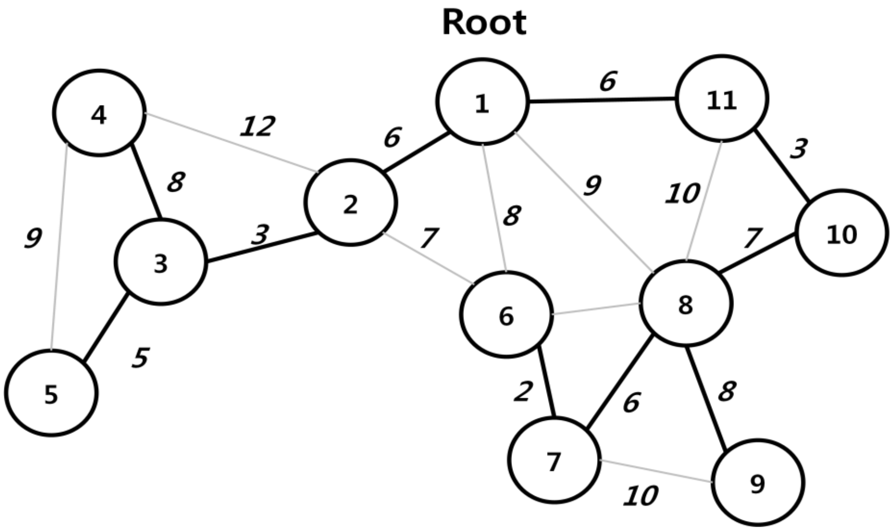

Figure 2.3 Minimum Spanning Tree

Figure 2.3 shows that a minimum spanning tree has the smallest total weight of links among all spanning trees of the network. The starting point of the traversal becomes the root of the tree to connect the nodes. It is similar to DFS and BFS due to use the top-down fashion. A link between two nodes means that they are neighbors to each other. The number on the link means the weight between each neighbor nodes. The bold lines indicate the minimum spanning tree. The minimum spanning tree for a network can be constructed by using either the Prim's or the Kruskal's algorithm [5]. If additional nodes and links can be added in the process of constructing the minimum spanning tree, the total weight can be further reduced. The result is called a Steiner tree [5].

2.3. Minimum Path Trees

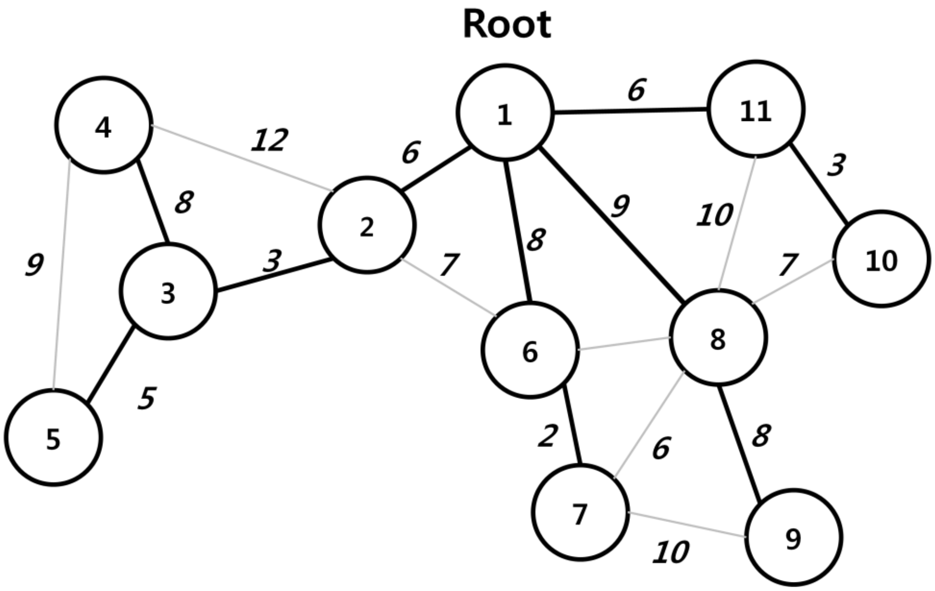

Figure 2.4 Minimum Path Tree

A minimum path tree optimizes the spanning trees in another way. It first selects a node at the extreme other end of the network from the root. When building the tree, each node chooses the neighbor with the smallest number of hops to the root as its parent. If multiple neighboring nodes have the same number of hops to the root, the node that is geographically closest is chosen. Figure 2.4 illustrates the result of the minimum path tree. Compared with Figure 2.3, we find out that a minimum path tree looks heavier than the minimum spanning tree in terms of total weight. A minimum path, however, has less weight. If we traverse to *node 6* from the root in Figure 2.3, the total weight is 24. Otherwise, in Figure 2.4, we figure out that the total weight to go to *node 6* is only 8. Shorter links in the tree construction process reduce the occurrences of crossing links, and result in a tree with subtrees that are better clustered together, thereby reducing the probability of intersections between convex hulls. However, the disadvantage of the minimum path tree is that the root is pre-selected without knowing the locations of the nodes in the network. This puts a serious constraint in the tree construction. In the case of a root that is poorly chosen, it may lead to a large number of conflicting hulls.

2.4. IEEE 802.1D

Although not designed for wireless networks, the IEEE 802.1D standard [32] specifies how a spanning tree can be generated autonomously among bridges that connect multiple local area networks distributively. It is in essence a two-step process. First, each bridge broadcasts its unique serial number so that the bridge with the lowest serial number can be selected as the root. Second, the shortest path from the root to each bridge and local area network is included to form a spanning tree topology. The ports of some bridges are intentionally disabled so that no loop is created. The protocol is also capable of maintaining the spanning tree in the presence of topology changes. However, the IEEE 802.1D root election process requires each node to broadcast its unique identification, which is an extremely expensive operation in the underwater environment.

2.5 Common Problems of Existing Tree Algorithms

Except for the IEEE 802.1D, the above tree construction methods all follow the top-down approach. They usually require centralized knowledge about the entire network. In practice, the centralized algorithms are implemented by sending information from all nodes to a centralized node, and then disseminating the decision to the entire network. This normally involves intensive message exchanges and may cause bottlenecks and low reliability. Clearly, distributed algorithms are more preferred in practice, but without centralized knowledge, they are difficult to develop.

The second problem of existing tree algorithms is that they construct the trees according to the node locations, but do not consider the bandwidth of each link. Consequently, some low bandwidth links in the tree may become bottlenecks and the resulting tree routing could suffer from congestion.

In the following chapter, we will develop a greedy spanning tree construction algorithm that considers both the location and the available bandwidth to minimize the number of the conflicting hulls and avoid the bottlenecks in the resulting spanning tree.

CHAPTER 3

BANDWIDTH-AWARE ROUTING TREE

In Chapter 2, we have discussed the fundamental issues and the tree routing commonly shared by the existing spanning tree algorithm. Most tree routings show a tendency to build the tree in similar manners, like a top-down and centralized fashion. In this chapter, we will introduce our routing tree algorithm namely Bandwidth-Aware Routing Tree (BART) to build the tree distributively in a bottom-up fashion. Our manner is focused on how we can define and collect the cluster using locations and bandwidth. First, we will present an overview of the BART.

3.1 Overview of BART

As explained in Chapter 2, existing spanning trees suffer from a number of problems that make tree construction and tree routing inefficient. To address them, we propose building the tree distributively in a bottom-up fashion based on locations of nodes and available bandwidth of links in the network. The notations and terms used for the rest of the thesis are defined as follows. A cluster is defined to be a set of nodes. Each cluster has a cluster head (or simply head when there is no confusion), and each node belongs to exactly one cluster. The center of a cluster is defined to be the average of the locations of the nodes in the cluster. The distance between two clusters $C1$ and $C2$, denoted as `dist`($C1$,$C2$), is defined as the Euclidean distance between the centers of them. Two clusters are said to be neighboring clusters if at least one node in the first cluster has a direct link to a node in the second cluster. Notice that in our definition that the distance between neighboring clusters can be greater than the transmission range of a node. To take the bandwidth factor into account, we assume each node is able to estimate the available bandwidth to each of its neighbors via beacon exchanges. The virtual bandwidth between two neighboring clusters $C1$ and $C2$, denoted as `bandwidth`($C1$,$C2$), is defined as the sum of all available bandwidth between the nodes of two clusters. Notice that the virtual bandwidth between two neighboring clusters is an approximation to the actual available bandwidth between them. The reason to use the virtual bandwidth instead of the actual one is the virtual bandwidth is easier to obtain between two neighboring clusters and should be a good indication to the amount of actual available bandwidth. For instance, in Figure 3.1, the cluster with head P has two neighboring clusters. The virtual bandwidth between the cluster with head P and the cluster with the head Q is the sum of the bandwidth between neighboring pairs (A, X), (B, Y), and (C, Y). The size of a cluster C is the number of nodes in C, and is denoted as $|C|$. The head of a cluster maintains the cluster information, including the center of the cluster, the size of the cluster, and the virtual bandwidth as well as the link with the largest available bandwidth to each of its neighboring clusters.

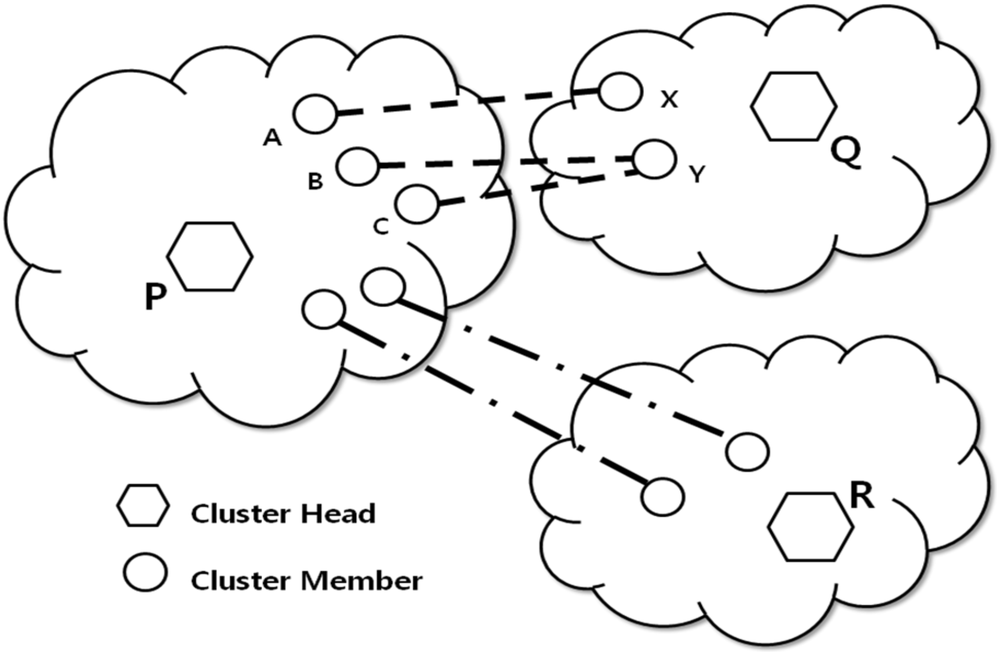

Figure 3.1 Example of Clusters and The Bandwidth Between Them

3.2 BART Algorithm

The basic idea of our Bandwidth-Aware Routing Tree algorithm is that each pair of neighboring clusters has a gravity, which is computed according to a function of the distance and the virtual bandwidth between them. The strength of the gravity is represented by a number in the range of $[1, \infty)$. The stronger the gravity two clusters have between them, the sooner they are merged into one cluster. This process is repeated until there is only one cluster left or there is no gravity between any pair of neighboring clusters. The cluster head can be in one of the following four states: comparing, requested, responded, and committed. To form the Bandwidth-Aware Routing Tree, each cluster head runs the Bandwidth-Aware Routing Tree algorithm stated in Table I.

Table I. Bandwidth-Aware Routing Tree Algorithm

```
/* The cluster head of a cluster $C$ runs the following */
if $|C| = 1$ { // Initialization
  compute the gravity $G_i$ to its neighboring clusters $C_i$
  find the largest gravity $G_T = max_{\forall i} G_i$
  set target cluster as $C_T$ and timer $T$ as $T_{max} / G_T$
  set state to be comparing }
while ( true ) {
  if ( state = comparing ) {
    repeat {
      reduce timer $T$ value
      if ( receive an update message from cluster $C_A$ )
        update information about $C_A$
    } until ( $T$ expires ) or ( receive a m-request )
    if ( $T$ expires )
      send a m-request to  and set state to be requested
    else {
      set target cluster $C_T$ to be the one sending m-request
      send a m-response to $C_T$ and set state to be responded }
  } else if ( state = requested ) {
    wait for $T_{merge}$ for m-response
    if ( $T_{merge}$ expires before receiving a response )
      rollback $C_T$ and $T$, and set state to be comparing
    else
      send a commit message and set state to be committed
  } else if ( state = responded ) {
    wait for $T_{merge}$ for commit message
    if ( $T_{merge}$ expires before receiving the commit message )
      rollback $C_T$ and $T$, and set state to be comparing
    else
      set state to be committed
  } else if ( state = committed ) {
    if ( $|C| < |C_T|$ ) {
      send cluster info to the cluster head of $C_T$
      turn into a regular cluster member }
    else {
      wait for $T_{merge}$ for the cluster info from $C_T$
      if ( $T_{merge}$ expires before receiving the cluster info )
        rollback $C_T$ and $T$, and set state to be comparing
      else {
        add a link to connect $C$ and $C_T$
        compute the gravity $G_i$ to its neighboring clusters $C_i$
        find the largest gravity $G_T = max_{\forall i} G_i$
        set timer $T$ as $T_{max} / G_T$
      set state to be comparing } } } }
```

In the algorithm, *Tmax* is the longest timer a node is allowed to set in number of the beacon period. Initially, each node is treated as a one-node cluster; the target cluster is set to be the neighbor with the strongest gravity to the node itself; and the timer is set to be *Tmax* divided by the gravity to the target cluster. When the timer of a cluster head expires, it sends a merge request containing the size of the cluster to the head of its target cluster to solicit a merge response. If the target cluster decides to accept the merge, it returns a merge response. After exchanging merge requests and responses, the two cluster heads execute the merge process. The head of the larger cluster becomes the head of the merged cluster, and the head of the smaller cluster submits its cluster information to the new cluster head and becomes a regular cluster member. In each merge, the head of the merged cluster includes the link with the highest available bandwidth between the two original clusters to connect them. It is not difficult to see that this process will introduce *n* – 1 links to connect n number of nodes. In other words, the result of the algorithm naturally forms a tree. We call this tree our Bandwidth-Aware Routing Tree (BART). Notice that according to our algorithm the cluster head is not necessarily the root of the tree.

When two clusters are merged, the center of the cluster may shift and the virtual bandwidth to their neighboring clusters may change due to the inclusion of new cluster members, which will affect the gravity to the neighboring clusters. The center of the cluster is the average of the locations of nodes in the cluster, and can be calculated directly from the centers and the sizes of the original clusters. Similarly, the virtual bandwidth of the merged cluster to the neighboring clusters can be calculated directly from the virtual bandwidth of the original clusters to their neighboring clusters. The link with the highest available bandwidth of the merged cluster to a neighboring cluster is the one with the higher bandwidth of the original two clusters to the neighboring cluster. As long as the head of the merged cluster obtains the cluster information from the head of the original smaller cluster during the merge process, it can derive these pieces of information and then calculate the new gravity to each of its neighboring clusters without additional message exchanges. After two clusters are merged into one, the head of the merged cluster will have to send an update message to inform heads of neighboring clusters of the new size of the merged cluster and the new virtual bandwidth between them. A cluster head who receives an update message from a neighboring cluster updates its information about that neighboring cluster, but will not reset its timer if it is in the comparing state. This is to ensure that the timer of a cluster head will eventually expire.

It is worth noting that before all nodes in the entire network are merged into one cluster, each cluster head maintains its own bandwidth-aware routing tree inside its cluster, which is used to route the merge requests/responses and cluster update messages to neighboring clusters.

3.3 Data Structure for Merging The Cluster

To optimize the performance of the algorithm efficiently, two data structures are used by each cluster head. The first is the disjoint set of member nodes in the cluster. Initially each node is treated as a disjoint set. As two clusters are merged, their sets are unioned together. The disjoint set is implemented with union-by-size and find-with-path-compression. The second data structure is the priority queues to store the virtual bandwidth and distances to neighboring clusters. These priority queues can reduce the time required to find the merge target from neighboring clusters. Whenever an update is received from a neighboring cluster, the corresponding entries in the priority queues are updated. In the process of a merge, the head of the smaller cluster submits both data structures as the cluster information to the new cluster head.

In the following subsections, we address two design issues in more detail: the gravity function and concurrent merge requests.

3.4 The Gravity Function

As explained in Chapter 1, more occurrences of conflicting hull in the network is likely to create a lengthy path to the destination. To minimize the possibility of having conflicting hulls between siblings, closer clusters should be merged early by assigning a larger gravity between them. To avoid creating bottlenecks in the tree, links with higher available bandwidth should be given higher priority so that they can be added to the tree. These observations lead us to define the gravity function between two neighboring clusters $C1$ and $C2$ as:

$$\mathtt{gravity}\,(C_1, C_2) = \frac{\mathtt{bandwidth}\,(C_1, C_2)}{\mathtt{dist}\,(C_1, C_2)^2} \tag{1}$$

As in IEEE 802.11b specification [31], neighboring nodes exchange beacons periodically. By examining the beacon signals from a neighbor, a node can estimate the available bandwidth to the neighbor. So, as long as the network is connected, after a series of merges all nodes should eventually belong to one cluster.

3.5 Concurrent Merge Requests

It is possible for multiple clusters have their timer expire before their merge process is complete. In this case, there will be concurrent merge requests sent out in the network. If the source and destination of these requests are different, the merge can be performed in parallel. Otherwise, the merge must be performed sequentially.

In Table I, when a cluster head intends to merge with its target cluster, it first sends a merge request to the target cluster and waits for a merge response from the target cluster. The cluster heads will go through a process similar to the two-phase commit protocol [19]. The head initiating the merge will go through the requested and committed states, while the target cluster head will go through the responded and committed states. In addition, the statements associated with the committed state must be treated as one atomic action, i.e., the cluster head will not be preempted before the merge process is complete. By incorporating these protection mechanisms, a cluster head will be involved in at most one merge request. For instance, in Figure 3.1, after cluster heads *P* and *Q* go through the two-phase commit and execute the merge process under the committed state, if *P* receives a merge request from *R* before its merge process is complete, it will ignore the request. For cluster head *R*, it will simply switch back to a comparing state if it does not get a merge response for its merge request for a short period of time *Tmerge*.

CHAPTER 4

ILLUSTRATION OF TREE CONSTRUCTION

In this chapter, we illustrate how a BART is construed distributively and explain the differences between BART and the minimum path tree. We use a 2-D example to assist understanding of the tree construction.

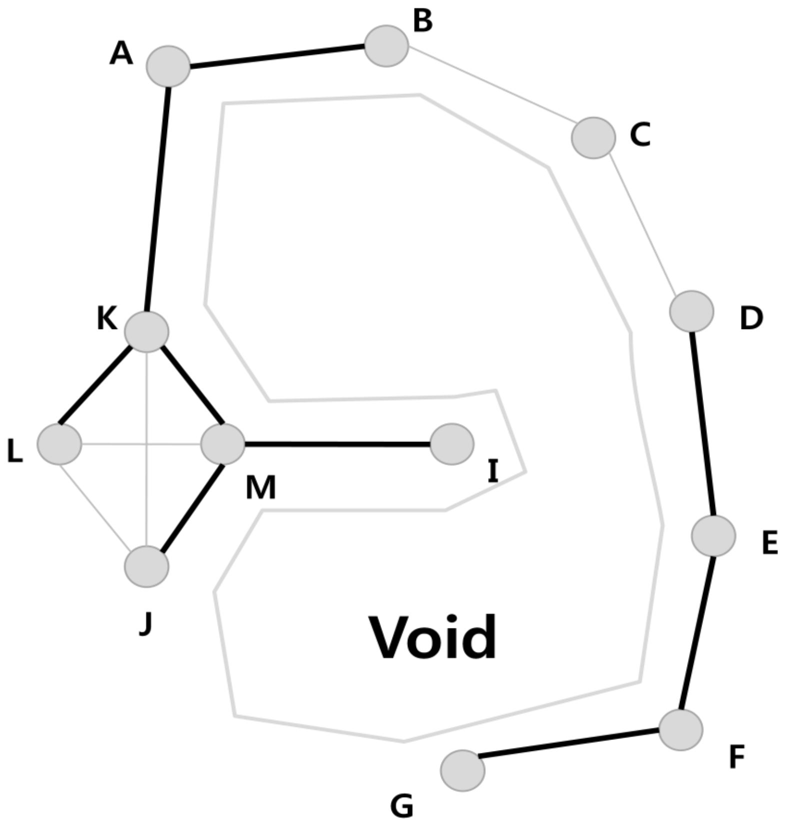

Figure 4.1 Original Network Topology

Initially, we set up a network of 12 nodes, as shown in Figure 4.1. A link between two nodes means that they are neighbors to each other. The wider a link is the more available bandwidth between two neighbors. An irregular shape void region is located in the center of the network.

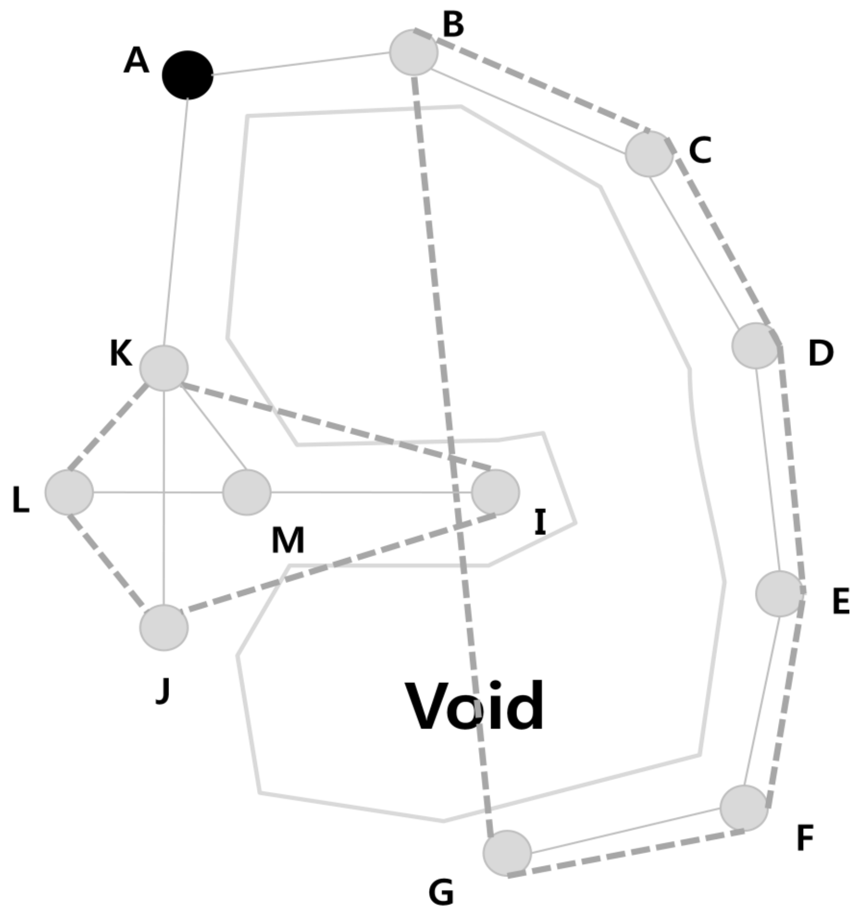

Figure 4.2 Minimum Path Tree

Using the algorithm described in [16], a minimum path tree can be constructed for the given network, as shown in Figure 4.2. Node *A* is first selected as the root of the tree (represented by the solid circle) because it is at the boundary of the network. As can be seen by the dotted lines, this root selection leads to the conflicting hulls at sibling nodes B and K. In addition, the link < *J*, *K* > is included in the tree to connect node *J*, but it has much a lower available bandwidth than the edge < *M*, *J* >, as shown in Figure 4.1.

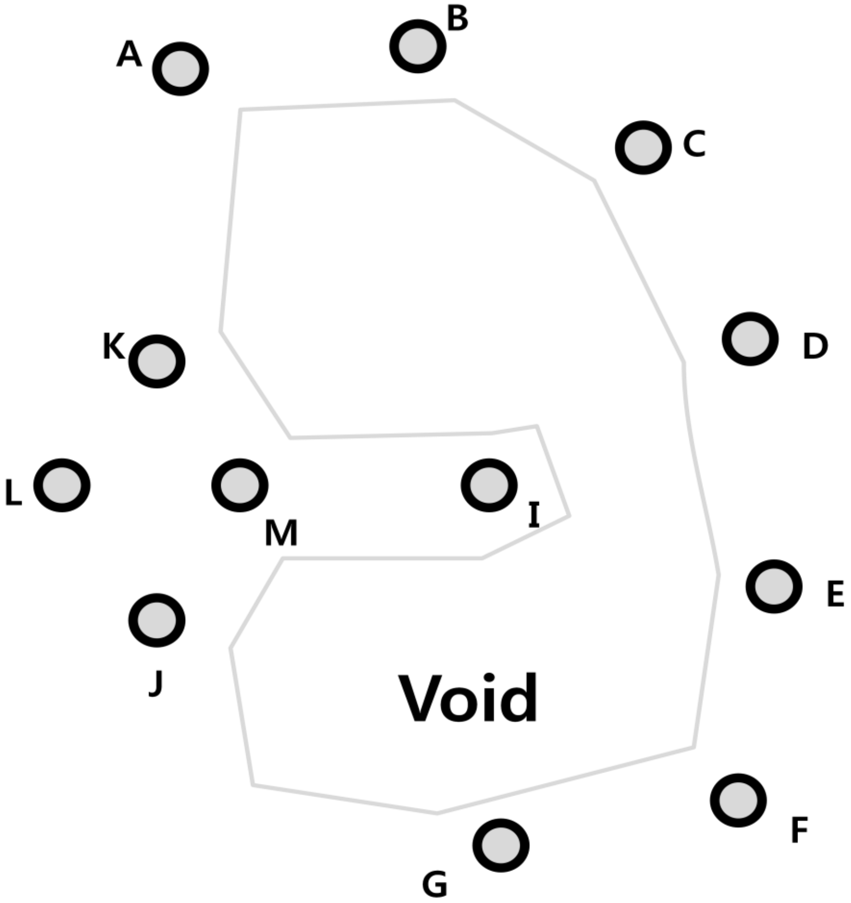

Figure 4.3 Construction of BART - Initialization

In contrast, BART is constructed in a bottom-up fashion and does not first make a selection of the root. As illustrated in Figure 4.3, initially each node is a single-node cluster and its own cluster head (represented by the thick circle). Each node then computes the gravity to each of its neighbors based on available bandwidth and distance and sets a timer inversely proportional to the largest gravity toward its target neighboring cluster.

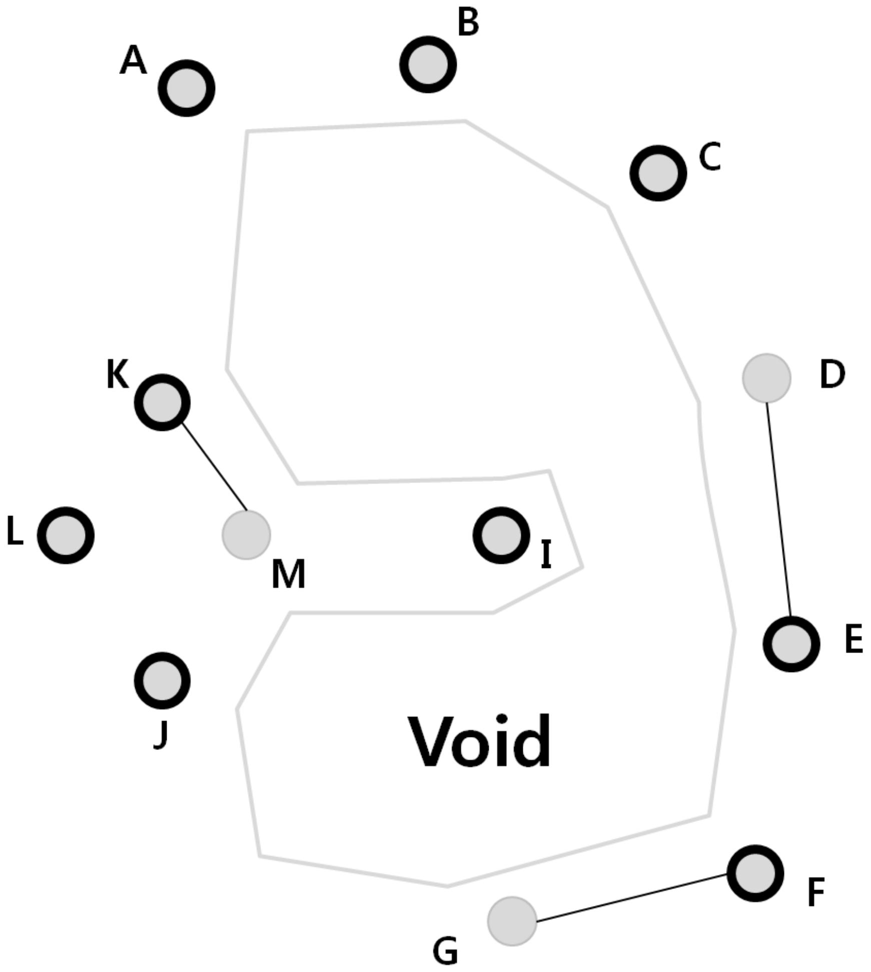

Figure 4.4 Construction of BART - Parallel Tree Construction

As shown in Figure 4.4, node pairs < *K*, *M* > < *D*, *E* >, and < *F*, *G* > have higher gravity due to more available bandwidth and a closer distance. Consequently, they will merge together first. After these merges, the node that has the most members becomes the root in the new cluster. In the cluster < *K*, *M* >, node *K* becomes the head of the new cluster and node *M* becomes a member of the new cluster (represent by thinner circle).

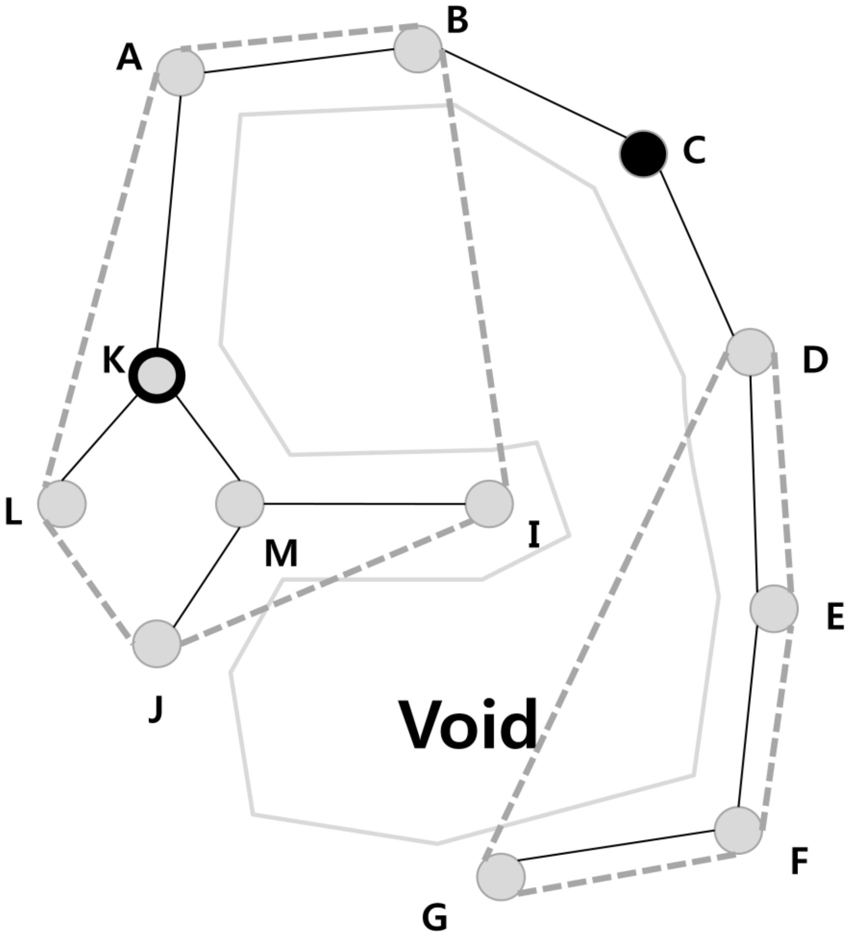

Figure 4.5 Construction of BART – Completion

These merges are independent and can happen in parallel. The merge process is repeated until all nodes join the tree and no other clusters remain, as illustrated in Figure 4.5. In the last merge, the link < *C*, *D* > is included in the tree. Therefore, the node *C* becomes the root of the whole tree. It is not difficult to see that there is no conflicting hull between any pair of sibling nodes.

In addition, the edges with the most available bandwidth are more likely to be included in the tree. For instance, the edge $< M, J >$ is now used in the tree to connect node J instead of edge $< J, K >$ compared with the minimum path tree in Figure 4.2. This effectively reduces the chance of creating traffic bottlenecks in the tree and more traverses to send packets to a destination.

CHAPTER 5

PERFORMANCE STUDY

In this chapter, we evaluate the performance of the proposed Bandwidth-Aware Routing Tree (BART). To show its performance, we use our own simulator developed through Java programming and ns-2 simulation [33] to implement the underwater situation. Also we compare its performance with two other spanning trees: Breadth First Search Tree (BFST) and Minimum Path Tree (MPT). To showcase the capability of the tree routing, the simulations are carried out in 3D network scenario.

5.1. Simulation Settings

The transmission range of each node is set to be 300 *m*. The transmission power is set to be 92 *dB*, which approximately corresponds to the transmission range. We randomly generated the topologies by placing nodes in a 3-D cube having sides with a length of 1000 *m* according to the uniform distribution. The cube is wrapped at both ends of each dimension to eliminate the edge effect. The total number of nodes placed in the cube ranges from 30 to 150, which corresponds to network densities from 3.39 to 16.96 neighbors per node. For each network density, we generate 50 topologies and use them to evaluate the performance of the three spanning trees. For each pair of neighbors, the available bandwidth between them is randomly generated in the range from 1 *kbps* to 100 *kbps* with uniform distribution. For each realization, two pairs of source and destination, which are connected through the network, are randomly selected. Each source node generates constant bit-rate (CBR) traffic flows to its destination simultaneously. Each CBR flow sends 100 consecutive packets of 50 *bytes* at the transmission rate of 10 *Kbps*. The inter arrival time of packets is set to be 30 seconds.

To simulate the characteristics of the underwater environment, the ns-2 underwater module [10] is used. For MAC protocol, CSMA/CA [23] with stop-n-wait ARQ (i.e., ACK) is implemented. The following subsections elaborate on the physical and MAC layer configurations.

1) Physical Layer Configurations: In [10], the physical characteristics of the underwater acoustic sensor networks (UWSN) are realized by the creation of propagation, channel, and physical layer models. The propagation model calculates the signal-to-noise ratio (SNR) at each receiver node after attenuation, ambient noise, and the interference range of a signal. The attenuation is based on the spreading loss [30] and on Thorp's approximation [3] for the absorption loss. In addition, the orientation of the link (i.e., whether it is horizontal or vertical) affects the signal attenuation through acoustic fading channels. The ambient noise in the underwater environment is obtained from turbulence, shipping, wind, and thermal.

The channel model is used for the calculation of neighbor sets, collision, and so on. In addition, it calculates propagation delay, which is formulated by the speed of the sound, the depth of the water, and the temperature of the water [35].

The physical layer model is responsible for packet reception, including transmission error, transmission time, and propagation delay by calling the function in the propagation and channel models. In addition, it calculates the energy consumption for transmission. The hardware related parameters of the physical interface, such as the receiving signal strength threshold and the maximum transmission power, are set according to the WHOI micro modem [26].

Although the underwater model in [10] does not provide the bit error rate (BER), it is able to calculate the receiving power, which can be used to infer the BER the experiment results presented in [7]. Without loss of generality, we adopt the error model in [7] as an exponential equation, $0.4 \times exp(-0.323 \times Pr)$, where Pr is the receiving power. In this equation, BER increases quickly as the receiving power decreases. For instance, when Pr is 30 *dB*, BER is 2×10^{-5}, and when Pr is 40 *dB*, BER is 10^{-6}.

2) MAC Layer Configurations: As suggested in [6], the CSMA/CA with ARQ is adopted as the medium access control mechanism for our underwater simulations. The RTS/CTS handshaking is removed because it causes serious delays in a low rate and high delay underwater environment. To battle with high BER, the stop-n-wait ARQ (i.e., link layer ACK [27]) is used in our underwater simulations. If a node does not receive ACK within the period of the ACK timeout after it transmits a packet, it assumes the packet is lost and retransmits the packet. A packet can be retransmitted as needed up to a prefixed number of times. To simplify the simulation design, the forward error correction [27] is not implemented and the packet error rate (PER) is approximated as the product of BER and packet size. In other words, if any bit in a packet is lost, the whole packet will be discarded.

Due to the special physical characteristics of underwater acoustic links, the MAC parameters are adjusted accordingly. For instance, the value of the slot time in the CSMA/CA backoff mechanism has to account for the propagation delay of the physical layer. Since the speed of sound in water is approximately 1500 m/s, the slot time should be much longer than that in wireless media. The MAC parameters in our simulations are set similar to what are used in [22]. The inter frame spacing and the slot time are both set to be 0.18 second. The minimum and maximum contention window sizes are set to be 4 and 32, respectively. To accommodate the high propagation delay of acoustic signals, the duration of the ACK timeout is set to be 5 seconds and the number of maximum link retransmissions is set to 10.

5.2. Simulation Results and Analysis

To evaluate routing protocols, six different metrics, including Number of Conflicting Convex Hulls, Average Path Hops, Average Path Distance, Average Throughput, Delivery Rate, and Energy Consumption, are used. Their definitions are provided as follows.

1) Number of Conflicting Convex Hulls: The number of conflicting convex hulls is calculated by the conflict that happened between sibling clusters.
2) Average Path Hops: The average path hops is defined as the average number of hop between source and destination pairs.
3) Average Path Distance: the average path distance is defined as the sum of the Euclidean distance of links in the path from source to destination.

4) Average Throughput: The average throughput is defined as the smallest available link bandwidth along the path from source and destination.

5) Delivery Rate: The delivery rate is defined as the ratio of the total number of received packets to the total number of generated packets.

6) Energy Consumption: The energy consumption is calculated by the ns-2 underwater module [10]. The unit of energy is a Joule.

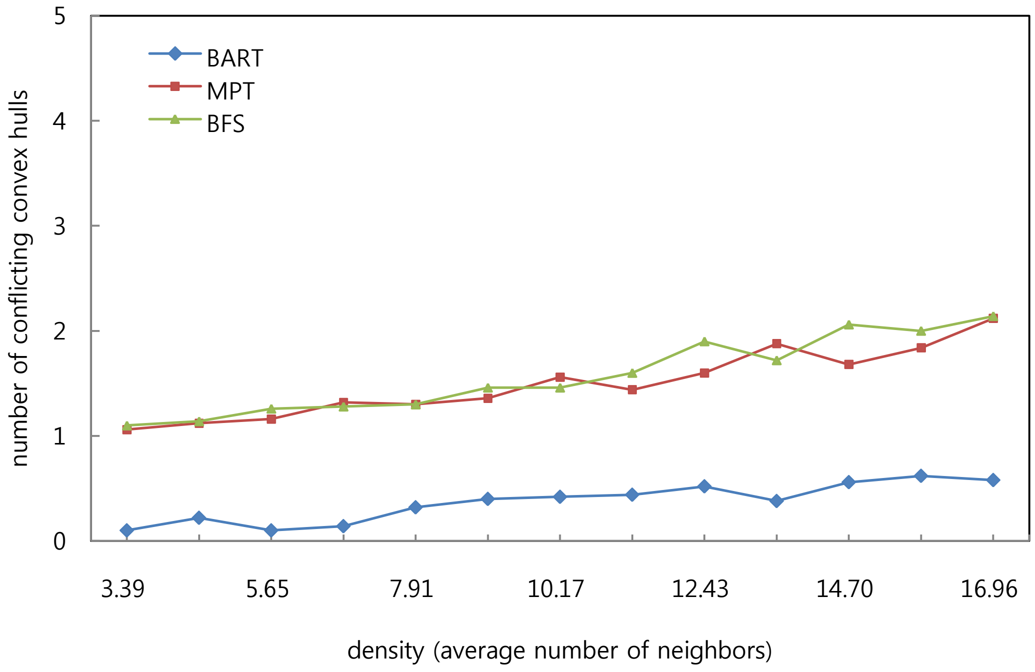

Figure 5.1 Number of Conflicting Convex Hulls

The first metric is the average number of conflicting hulls in the spanning tree. After the spanning trees are produced by the three algorithms, we calculate the 3-D convex hulls of the subtrees and count the number of conflicting convex hulls among siblings. Each point in Figure 5.1 represents the average of the number of conflicting hulls in 50 different topologies under the same network density. More conflicting hulls result in a more complicated routing. As shown in Figure 5.1, the number of conflicting hulls of BART is less than one for all density levels and is consistently smaller than the other methods. This is because BART is built from bottom-up. Closer clusters are merged first so that the convex hulls between siblings have little chance of overlapping. Therefore, it results in a geographically "compact" tree.

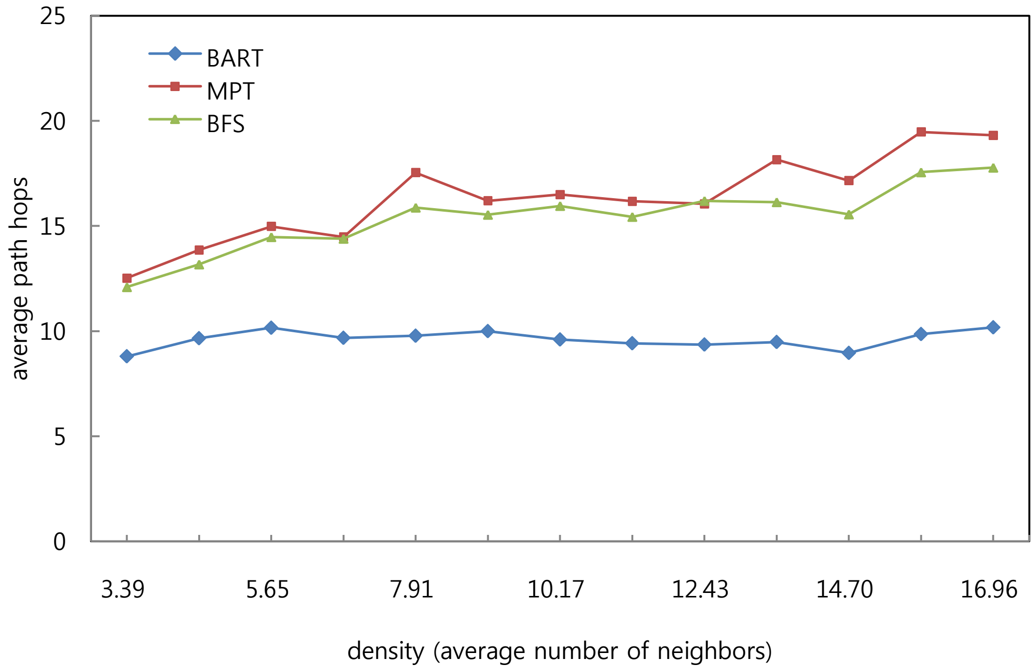

Figure 5.2 Average Path Hops

The second metric is the average path hops. For each generated network topology, 2 source-destination pairs are randomly selected to perform routing. Each point in Figure 5.2 represents the average of 100 source-destination pairs from the 50 different network topologies for a given network density. A smaller number of path hops means that the end-to-end delay is lower, so the traffic can reach the destination faster. While MPT is formed by the shortest paths from each node to the root, its average path hops may not be the lowest because the root may be neither the source nor the destination. As shown in Figure 5.2, BART has the smallest average path hops. In BART, the distance between neighboring clusters is one of the major factors to determine the merge of clusters. The resulting tree therefore has the smallest average path hops.

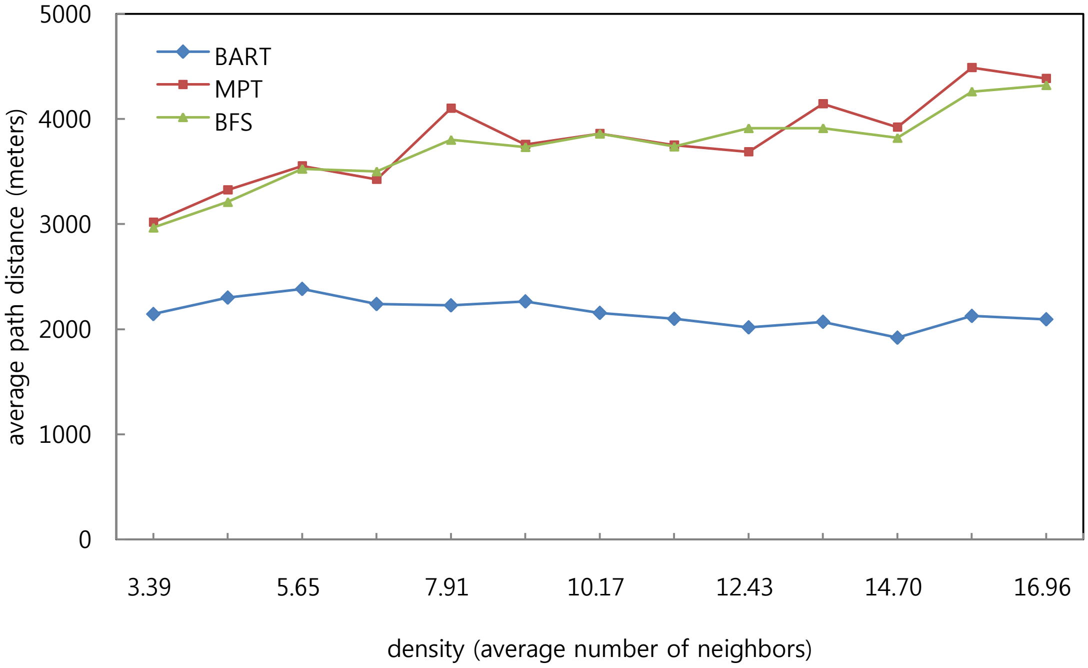

Figure 5.3 Average Path Distance

The third metric is the average path distance. A smaller average path distance means less energy is spent on each routing, thus prolonging the lifetime of nodes in the network. Similar to the average path hops, each point in Figure 5.3 is an average of 100 source-destination pairs from 50 different network topologies for a given network density. Compared with Figure 5.2, the result of Figure 5.3, taken between same densities, reflects very similar results. Otherwise, it shows that the average path hops is highly related with the average path distance. In Figure 5.3, it is clear that BART has the smaller average path distance than BFS and MPT. Recall that BART has clearly smaller average path hops, so the results of the average path distance again confirm that the average link length in BART is shorter. In underwater sensor networks, a shorter link may be preferred since it has the advantage of higher transmission success rates.

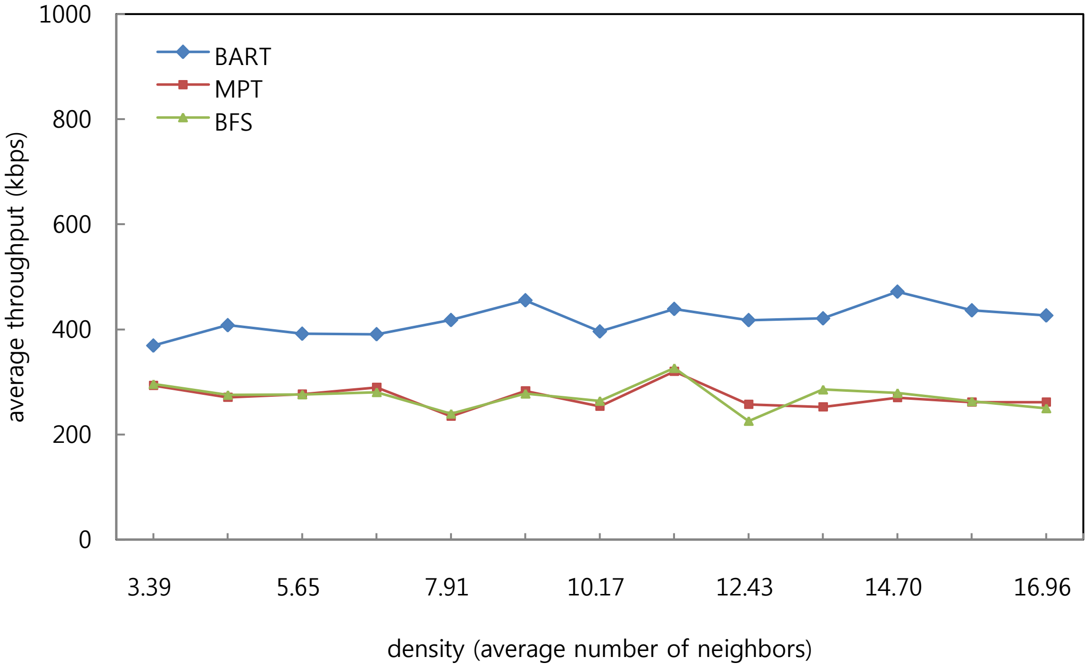

Figure 5.4 Average Path Throughput

The fourth metric is the average path throughput of the routing path. The larger the path throughput is, the less likely that the traffic will be congested. Again, each point in Figure 5.4 represents an average of 100 source-destination pairs from 50 different network topologies for a given network density.

As shown in Figure 5.4, BART has the largest path throughput because clusters of nodes are merged based on the inter-cluster available bandwidth as well as their distances. By merging clusters with higher bandwidth between them earlier, BART provides more bandwidth for routing along in the tree.

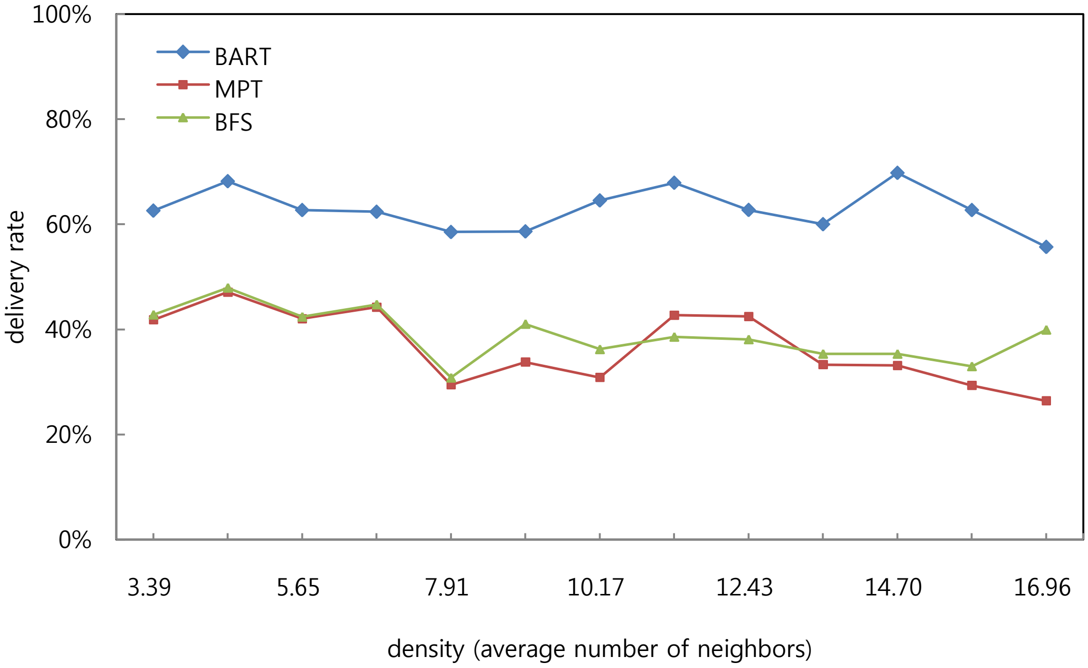

Figure 5.5 Delivery Rate

The fifth metric is the delivery rate. Figure 5.4 illustrates the delivery rate of different routing protocols with respect to the network density. The larger delivery rate means the higher success rate to transfer the packet from source to destination. As observed, BART achieves a much higher delivery rate compared with MPT and BFST. This is due to the fact that BART has an efficient detouring strategy based on the routing tree concentrated on the distance and the bandwidth. Recall in Figure 5.2 BART is always the lowest average path hop.

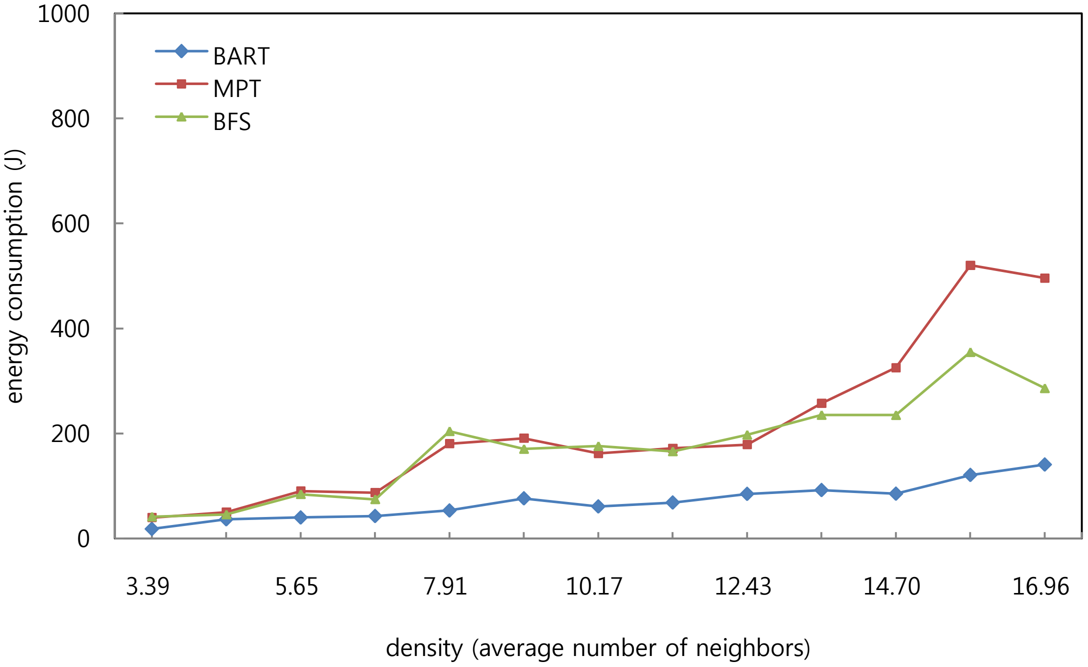

Figure 5.6 Energy Consumption

The last metric is the energy consumption. Figure 5.6 demonstrates the energy consumption of different routing protocols with respect to the network density in the scenario. As can be seen in Figure 5.6, less energy is consumed by BART than MPT and

BFST because BART delivers the packet to designation with shorter path hops. According to the increased density, energy consumption grows gradually.

CHAPTER 6

CONCLUSION AND FUTURE WORK

Geographic routing is a powerful algorithm well-known to be light-weight and efficient in wireless sensor networks, but when it is adopted for underwater sensor networks, most detouring strategies fail because their assumptions do not apply to the environment of such networks. In [15], spanning tree routing has been shown to be a good detouring strategy alternative for geographic routing without the need of those assumptions.

However, the performance of the spanning tree routing is largely dependent on the quality of the pre-constructed spanning trees with a top-down fashion. In this thesis, the Bandwidth-Aware Routing Tree (BART) algorithm is presented. Compared with the existing spanning trees, the proposed BART makes better use of the location and bandwidth information in the network. Simulation results confirm that the spanning tree routing based on BART achieves a better routing performance in terms of the number of conflicting hulls, average path hops, average path distance, average path throughput, delivery rate, and energy consumption.

Currently there is no pre-constructed spanning tree designed to be adaptive to network mobility. Slight movement changes of network topology may completely change the routing tree. Because of the distributed nature of BART, when a link is broken due to environment interference of node movement, theoretically the affected nodes can do a local search to find alternative links to reconnect network without reconstructing the entire tree. This option will be investigated in our future work.

BIBLIOGRAPHY

[1] Akyildiz, I. F., Pompili, D., and Melodia, T., “Challenges for efficient communication in underwater acoustic sensor networks,” ACM SIGBED Review, vol. 1, pp. 3 - 8, Jul. 2004.

[2] Akyildiz, I. F., Pompili, D., and Melodia, T., “State of the Art in Protocol Research for Underwater Acoustic Sensor Networks,” to appear in ACM Mobile Computing and Communication Review, 2007.

[3] Berkhovskikh, L., and Lysanov, Y., “Fundamentals of Ocean Acoustics,” Springer, 1982.

[4] Bose, P., Morin, P., Stojmenovic, I., and Urrutia, J., “Routing with guaranteed delivery in ad hoc wireless networks,” Proc. the Sixth ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM), pp. 48?55, Aug. 1999.

[5] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., Introduction to Algorithms, the MIT Press; 2nd edition, Sep. 2001.

[6] Creber, R. K., Rice, J. A., Baxley, P. A., and Fletcher, C. L., “Performance of Undersea Acoustic Networking using RTS/CTS Handshaking and ARQ Retransmission,” vol. 4, pp. 2083–2086, Nov. 2001.

[7] Freitag, L., Grund, M., Singh, S., Partan, J., Koski, P., and Ball, K., “The WHOI Micro-Modem: An Acoustic Communications and Navigation System for Multiple Platforms,” in Proceedings of MTS/IEEE OCEANS 2005, vol. 2, pp. 1086–1092, Sep. 2005.

[8] Garbriel, K. and Sokal, R., “A new statistical approach to geographic variation analysis,” Systematic Zoology, 18 (1969), pp. 259278.

[9] Hahn, M., “Undersea Navigation via a Distributed Acoustic Communication Network, ” Master’s thesis of Naval Postgraduate School, Jun. 2005.

[10] Harris, A. F. III, and Zorzi, M. “Modeling the Underwater Acoustic Channel in ns2,” in Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools (ValueTools). ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–8, Oct. 2007.

[11] Johnson, D. B., “Routing in Ad Hoc Networks of Mobile Hosts,” Proc. of the Workshop on Mobile Computing Systems and Applications, pp. 158-163, IEEE Computer Society, Santa Cruz, CA, Dec. 1994.

[12] Karp, B. and Kung, H. T. “GPSR: Greedy perimeter stateless routing for wireless networks,” Proc. the 6th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000), pp. 243-254, Boston, MA, Aug. 2000.

[13] Kim, Y.-J., Govindan, R., Karp, B., and Shenker, S., “Geographic Routing Made Practical,” In Proc. of the USENIX Symposium on Networked Systems Design and Implementation, pp. 217 – 230, Boston, MA, May 2005.

[14] Kuhn, F., Wattenhofer, R., Zhang, Y., and Zollinger, A., “Geometric ad-hoc routing: of theory and practice,” Proc. The 22nd ACM Annual Symposium on the Principles of Distributed Computing (PODC 2003), pp. 63-72, Boston, MA, Jul. 2003.

[15] Leong, B., Liskov, B., and Morris, R., “Geographic Routing Without Planarization,” Proc. of NSDI 2006, pp. 339352, May 2006.

[16] Leong, B., New Techniques for Geographic Routing, PhD thesis, MIT, 2006.

[17] Li, X.-Y., Calinescu, G., and Wan, P.-J., “Distributed construction of planar spanner and routing for ad hoc networks,” Proc. IEEE INFOCOM, vol. 3, pp. 1268-1277, Jun. 2002.

[18] Liu, H., Darabi, H., Banerjee, P., and Liu, J., “Survey of Wireless Indoor Positioning Techniques and Systems,” IEEE Trans. Systems, Man, and Cybernetics-Part C: Application and Reviews, vol.37, pp. 1067-1080, Nov. 2007.

[19] Papadimitriou, C., The Theory of Database Concurrency Control, Computer Science Press, Jul. 1986.

[20] Perkins, C. E. and Royer, E. M., “Ad-hoc on-demand distance vector routing,” Proc. the Second IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100, Feb. 1999.

[21] Perkins, C., and Bhagwat, P., “Highly dynamic destinationsequenced distance-vector routing (DSDV) for mobile computers,” Proc. ACM Conference on Communications architectures, protocols and applications (SIGCOMM), pp. 234–244, Oct. 1994.

[22] Pompili, D., Melodia, T., and Akyildiz, I. F., “Routing Algorithms for Delay-insensitive and Delay-sensitive Applications in Underwater Sensor Networks,” in

Proc. of ACM Conference on Mobile Computing and Networking (MobiCom), Los Angeles, CA, September 2006.

[23] Sidhu, G., Andrews, R., and Oppenheimer, A., “in Inside AppleTalk,” Addison-Wesley, 1989.

[24] Sozer, E. M., Stojanovic, M., and Proakis, J. G.,“Underwater Acoustic Networks,” IEEE Journal of Oceanic Engineering, Vol.25, pp. 72-83, Jan. 2000.

[25] Stojanovic, M., “Recent Advances in High-Speed Underwater Acoustic Communications,” IEEE Journal of Oceanic Engineering, vol.121, No. 2, pp.125-136, April 1996.

[26] Stojanovic, M., Proakis, J. G., and Catipovic, J., “Performance of High-Rate Adaptive Equalization on a Shallow Water Acoustic Channel,” vol. 100, no. 4, pp. 2213–2219, 1996.

[27] Tanenbaum, A. S., Computer Networks. Prentice Hall PTR; 4 edition, 2002.

[28] Toussaint, G., “The relative neighborhood graph of a finite planar set,” Pattern Recognition 12, 4 (1980), pp. 261268.

[29] Tseng, Y.-C., Ni, S.-Y., Chen, Y.-S., and Sheu, J.-P., “The Broadcast Storm Problem in a Mobile Ad Hoc Network,” Wireless Networks 8(2-3), pp. 153-167, 2002.

[30] Urick, R., “Principles of Underwater Sound,” Peninsula Pub, 1983.

[31] IEEE 802.11b Standard, http://standards.ieee.org/getieee802/download/802.11b-1999.pdf.

[32] IEEE 802.1d standard, http://standards.ieee.org/getieee802/download/802.1D-1998.pdf

[33] Information Science Institute, “The Network Simulator (ns-2),” http://www.isi.edu/nsnam/ns/.

[34] Open Shortest Path First v3, RFC 2740, 1999.

[35] University Corporation for Atmospheric Research(UCAR), “Temperature of Ocean Water,” http://www.windows.ucar.edu/tour/link=/earth/Water/temp.html.