ENERGY-EFFICIENT TRAFFIC-AWARE DETOUR TREES FOR GEOGRAPHICAL ROUTING

Lei Zhang[1], Deying Li[2] and Alvin Lim[3]

[1]Frostburg State University, Frostburg, MD, U.S.A
lzhang@frostburg.edu
[2]Renmin University of China, Beijing, P.R. China
deyingli@ruc.edu.cn
[3]Auburn University, Auburn, U.S.A.
lim@eng.auburn.edu

ABSTRACT

Tree routing is one of the detouring strategies employed in geographic routing to help find a detour for a packet to leave a local minimum. The effectiveness of tree routing depends on the quality of the pre-constructed routing trees. Existing tree construction methods build trees in a top-down and centralized fashion and do not consider the traffic pattern and residual energy of the network. Therefore is likely to create trees with poor routing performance. In this paper, we propose a novel routing tree, namely Energy-Efficient Traffic-Aware Detour Tree, which is constructed completely in a bottom-up fashion, with the consideration of both traffic load and residual energy. The simulation shows that our detour tree rarely encounters the problem of conflicting hulls, has much higher average path residual energy and throughput than other detour trees, leading to a better routing performance.

KEYWORDS

Detour tree, Geographic routing, Energy-efficient traffic-aware routing

1. INTRODUCTION

Wireless network routing can be roughly divided into two categories: proactive routing and reactive routing [1-7]. Maintaining updated routing information to any other nodes in the network introduces high overhead in proactive routing. Longer response time and poor scalability due to expensive network flooding are the major drawbacks of reactive routing. Though hybrid of reactive and proactive routing [8, 9] is designed to balance the tradeoff between both routing protocols, it still suffers from high complexity.

As a special kind of wireless network routing protocol, geographical routing [10-14] has several advantages over traditional routing algorithms. In general, geographic routing is simple, efficient, and scales better. As demonstrated in [14, 15], the routing state maintained by each node depends only on the local network density, but not the network size. Recently, geographic routing algorithms have also been widely applied to data-centric applications [16, 17]. Even when physical locations are not available, geographic routing can still be applied using virtual coordinates [18, 19].

Geographic routing forwards a packet in a *greedy* manner whenever possible and makes use of localized geographical location information. Thus, it is able to avoid most of the communication and storage overhead. Each packet carries the position of its destination. Each node in the network is assumed to know its own location and its neighbor's location. So, a node can always forward a packet to its neighbor that is geographically closest to the destination, so long as that neighbor

exists. This improves the routing scalability. Local minimum may exist where no neighbor is closer to the destination. In such case, greedy forwarding would fail, and a *detour strategy* must be applied to continue making progress toward the destination.

Different geographical routing protocols have different detour strategies to guarantee packet delivery. Flooding is used in [10] for finding a detour when a packet reaches a local minimum. This detour strategy is expensive especially in large networks. In [11, 12], the network topology is first reduced to a planar graph in a distributed manner [20, 21], and then a certain heuristic traversal is applied on the planar graph to find a detour. This detour strategy has an advantage over flooding in terms of the communication overhead, but it requires several impractical assumptions to work correctly, such as the unit disk link model and a flat network topology. Cross-Link Detection Protocol (CLDP) proposed in [14] solved the problem of these impractical assumptions. However, it still introduces extra overhead caused by "probe" packets for planarization.

To avoid this problem, [13, 22] construct a few pre-constructed routing trees to search for a detour. These simple and effective detour trees do not require impractical assumptions and still can achieve really good routing performance. [13] introduces the idea of convex hull to tree routing. In essence, each node keeps a convex hull of all nodes in the sub-tree rooted at the node. When a packet reaches a local minimum, it is forwarded to the child node whose convex hull contains the location of the destination. If no such child is available, the packet is forwarded to the parent node. In the worst case scenario, if there are overlapping or conflicting convex hulls, which means multiple children's convex hulls contain the location of the destination, they will be explored in turn until a node closer to the destination than the current local minimum is found, where greedy forwarding takes over again. For this reason, reducing the number of conflicting hulls will certainly remove routing ambiguity and improve the routing efficiency.

The detour tree strategy does not rely on aforementioned assumptions and has been shown in [13, 22] to perform well compared to other geographical routing protocols [11, 12]. However, the success of the tree-based detour largely depends on the quality of the pre-constructed tree. In [13], several different types of routing trees are studied and the minimum path tree is suggested to be the better choice. However, in their study, the root of a tree is always fixed. This places a serious constraint to the construction of the 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. Additionally, two important factors: residual energy distribution and traffic pattern of the network are not taken into account in their study. In their case, if two geographically close nodes with traffic volume are placed logically far apart in the tree, it would not only create bottlenecks, but also increase energy consumption.

In this paper, we propose a different routing tree algorithm to find a detour out of dead-end nodes. It is built completely in a distributed bottom-up fashion. This allows the number of conflicting hulls to be reduced significantly. In addition, the network residual energy distribution and traffic patterns are considered in the detour tree construction. We call this detour tree Energy-Efficient Traffic-Aware Detour Tree (ETDT). ETDT makes detour tree routing more efficient. The simulation results show that our detour routing tree rarely encounters the problem of conflicting hulls and has a much higher average path residual energy and throughput compared to other routing trees.

The remainder of the paper is organized as follows. In Section 2, we review the different types of spanning trees. In Section 3, we propose our detour tree and the distributed algorithm for constructing it. In Section 4, we present the simulation results. Finally, we provide the conclusion and future research directions in Section 5.

2. RELATED WORK

In this section, we review several existing geographical detour methodologies including planarization and detour tree algorithms. Detour tree methods have several advantages over planarization detour methods. In detour tree strategy, data packets are delivered toward their destinations along a path in the pre-constructed spanning tree topology. A spanning tree of a connected network is defined as a tree that contains all the nodes in the network. For a given network, there are many different spanning trees. In this section, we not only point out the drawbacks of planarization detour, but also investigate several spanning tree algorithms and discuss their applications in wireless networks routing.

2.1. Geographical Routing Algorithms

2.1.1 Planarization

In geographical routing, detouring packets out of a void area is a major research issue. There are a numbers of solutions available for solving this problem. GPSR [11]combines face routing with right hand rule to traverse the perimeter of void area. This traversal is not so efficient if the correct face cannot be found quickly. In the worst case scenario, all the faces would have been traversed before the last face is found to be the correct one. GOAFR [23] improves GPSR, by adopting adaptive face routing (AFR) to detour. AFR adjusts the boundary of a traverse ellipse area around the face and chooses an optimal value to reach the destination. Both GPSR and GOAFR depend on planarization to support face routing. Planarization assumes that the connectivity between nodes can be described by unit graphs. [14, 24] discovered that unit graph assumption cannot always be satisfied in reality. Instead, a distributed Cross-Link Detection Protocol (CLDP) has been proposed by [14] to planarize the network and solve the problem of GPSR and GOAFR. However, CLDP has to pay the expensive price of extra overhead caused by probe packets for planarization.

2.1.2. Search Trees

Detour tree methods are alternative to planarization. In the routing tree detour strategy, data packets are delivered toward their destinations along a path in the pre-constructed tree topology. Depth-first search (DFS) and breadth-first search (BFS) are two principal algorithms for traversing a connected network and creating routing trees [25].

In the DFS algorithm, the starting point of a 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. 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 node is reached, i.e., a node without adjacent unvisited node, is encountered. At a dead-end node, the tree construction method backs up one link to the node where it came from and tries to continue visiting unvisited node from there. Eventually, it halts after backing up to the starting node, with the latter being a dead-end node. By then, all the nodes in the same connected components as the starting node have been visited.

The BFS algorithm, on the other hand, visits the neighboring unvisited nodes as wide as possible until no such node is available. 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.

While DFS tries 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.

2.1.3. Minimum Spanning Trees

A spanning tree of a connected network is defined as a tree that contains all the nodes in the network. Clearly, both DFS and BFS trees are spanning trees of the original network. While a network can have many spanning trees, a minimum spanning tree has the smallest total weight of links among all spanning trees of the network. The minimum spanning tree for a network can be constructed by using either the Prim's or the Kruskal's algorithm [25]. If additional nodes and links may 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 [25].

2.1.4. Minimum Path Trees

A minimum path tree optimizes the spanning trees in another fashion. It first selects a node at the extreme end of the network as the root. When building the tree, each node chooses the neighbor with the minimum number of hops to the root as its parent. If a node has a choice between multiple neighboring nodes that are the same number of hops from the root, the geographical closest node is chosen. The shorter links in the tree construction process reduce the occurrences of crossing links, and result in a tree with sub-trees that are more clustered together, thereby reducing the probability of intersections between convex hulls and creating fewer conflicting 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 case if the root is poorly chosen, it may lead to a large number of conflicting hulls.

2.2. Common Issues of Existing Tree Algorithms

The above tree construction methods all follow the top-down approach. They usually require centralized knowledge about the entire network, and therefore are difficult to develop a distributed algorithm to construct the trees. 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 low reliability. Clearly, distributed algorithms are more preferred in practice. In addition, these methods construct the trees according to the node locations, but ignore the traffic load and its tight relationship with residual energy. Therefore, some nodes lack residual energy but are fully congested with high traffic volume, while other links with full residual energy end up with no traffic. The reason is that the tree is not constructed with the full use of relationship between residual energy and traffic distribution. Consequently, the result routing is neither optimal nor efficient.

In the following, we will develop a greedy spanning tree construction algorithm that considers the location, the residual energy and traffic pattern of the network with the intention of minimizing the number of the conflicting hulls, optimizing the energy usage and balancing the traffic load of the entire network.

3. ENERGY-EFFICIENT TRAFFIC-AWARE DETOUR TREE

3.1. Design of Energy-Efficient Traffic-Aware Detour Tree

As described in Section 2, the existing routing trees suffer from a number of issues that will cause the tree construction as well as the tree routing to be inefficient. To address these issues, we propose to build the tree in a distributed bottom-up fashion. The notations and terms used for the remainder of the paper are defined as follows. A cluster is defined to be a set of nodes. Each cluster has a cluster head (or simply head if 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 nodes in the cluster. The distance between two clusters *C1* and *C2*, denoted as *dist (C1, C2)*, is defined as the Euclidean

distance between their centers. If at least one node in a cluster is a neighbor of a node in the other cluster, the two clusters are neighboring clusters. Notice that in our definition the distance between neighboring clusters can be greater than the transmission range of a node. To take into account the energy distribution and traffic tendency, we assume each node learns about its neighbor's *average residual energy* and *average available bandwidth* from their past history. To make the problem simple, we assume the initially energy for each node in the network is same and each transmission consumes the same mount of energy. The virtual energy of two neighboring clusters *C1* and *C2*, denoted as *energy(C1,C2)*, is defined as the sum of all residual energy of the two clusters' nodes. 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. The virtual bandwidth-energy product (which we will simply call virtual product) is defined as the product value of virtual bandwidth and virtual energy. Notice that the virtual energy of two neighboring clusters is an estimation of their actual average residual energy, while the virtual bandwidth between two neighboring clusters is an estimation of the actual average available bandwidth between them. The reason to use the virtual value instead of the actual average available value is because it requires much lower computational and communication overhead to compute the virtual values after the merge of clusters. The head of a cluster maintains the cluster information, including the center of the cluster, the size of the cluster, and the virtual values as well as the link with the largest virtual product of each neighboring cluster.

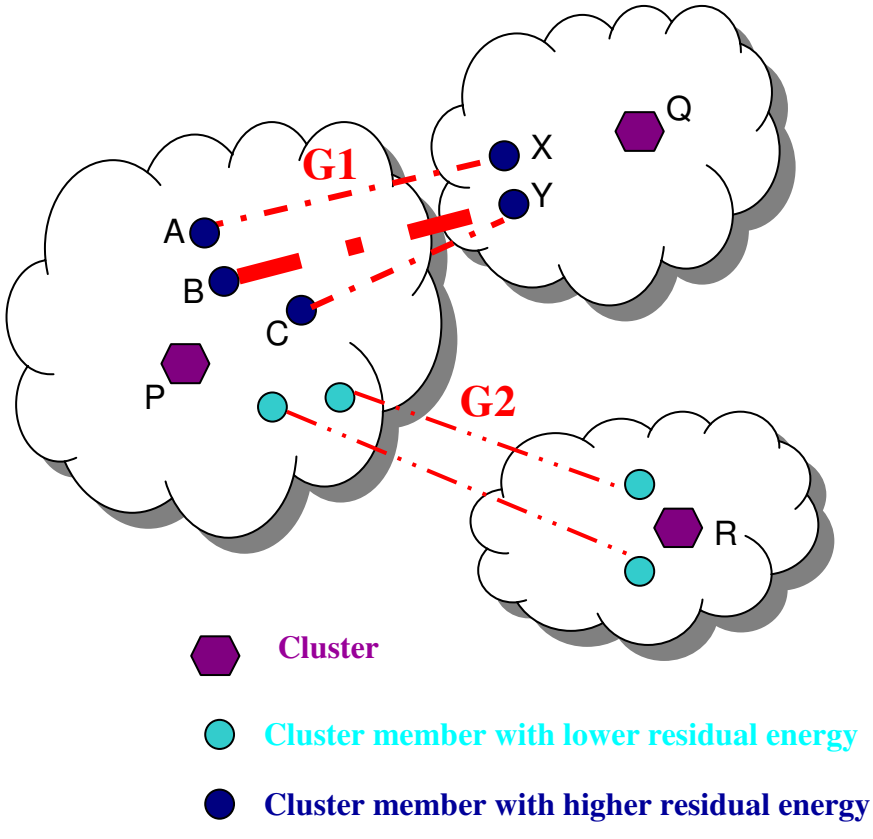

Figure 1. An Example of Clusters Combination

Table 1. Energy-Efficient Traffic-Aware Detour 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_∀i G_i
      set target cluster as C_T and timer T as T_max/G_T
      set state to comparing}
while (true) {
    if ( state = comparing) {
            repeat {
          reduce timer Tvalue
          if ( receive an update message from cluster C_A)
                    update information about C_A
       }until (T expires ) or ( receive a m-request)
       if ( Texpires )
              send a m-request to C_T  and set state to requested
       else {
             set target cluster C_T  to the one sending m-request
             send a m-response to C_T and set state to 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 comparing
      else
            send a commit message and set state to committed
}else if ( state = responded) {
            wait for T_merge for the commit message
            if (T_merge expires before receiving the commit message)
                  rollback C_T and T, and set state to comparing
            else
            set state to 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 comparing
    else {
            add the link with highest available bandwidth to C_T in the tree
            compute the gravity G_i to its neighboring clusters C_i
            find the largest gravity  G_T = max_∀i G_i
            set timer T as T_max/G_T
            set state to comparing}}}}
```

The basic idea of our Energy-Efficient Traffic-Aware Detour Tree algorithm is that each pair of

neighboring clusters has a *gravity*, which is computed according to a function of the square of their distance and the virtual product. The larger the gravity value of two neighboring clusters the earlier they will be merged together. The process will keep repeating until there is only one cluster left or zero gravity between any pair of neighboring clusters. For instance, in Figure 1, the cluster with cluster head P has two neighboring clusters Q and R. The virtual bandwidth between the cluster with head P and that with head Q is the sum of their bandwidth between neighboring pairs (A, X), (B, Y), and (C, Y). The virtual energy of cluster P and Q is the sum of residual energy of nodes A, B, C, X, and Y. The size of a cluster C is the number of nodes in C, and is denoted as |C|. In the case of Figure 1, it is obvious that the gravity of cluster P and cluster Q is greater than the gravity of cluster P and cluster R. Thus, the neighboring clusters P and Q will merge with each other first.

The cluster head can be in one of the following four states: *comparing*, *requested, responded,* and *committed*. To form the detour tree, each cluster head runs the detour tree algorithm described in Table 1. In the algorithm, *Tmax* is the longest timer for a beacon period that a node is allowed to set. Initially, each node is treated as a one-node cluster and the target cluster is the neighbor with the strongest gravity to itself. The timer is set to be *Tmax* divided by the gravity value to the target cluster. When the timer of a cluster head expires, the event of sending a merge request is triggered. The merge request including the size of the cluster is sent to the head of its target cluster in order to solicit a merge response. A merge response will be returned, if the target cluster decides to commit the merge. After exchanging merge request and response, 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 its new cluster head and turns into a regular cluster member. In each merge, the head of the merged cluster includes the link with the highest product of available energy and bandwidth between two original clusters to connect them. The algorithm will naturally form a detour tree, which is an Energy-Efficient Traffic-Aware Detour Tree (ETDT). According to our algorithm, the cluster head is not necessarily the root of the tree.

The center of the cluster is calculated by averaging the locations of nodes in the cluster. Similarly, by calculating the virtual values of the original cluster and its adjacent cluster, the virtual product of the newly merged clusters can be determined. The link between the original cluster and the target cluster is the link with the maximum virtual product.

After two clusters are merged into a larger one, the head of the resulting cluster will send to the head of each neighboring cluster an update message containing the size of the merged cluster and its virtual product. A cluster head receiving an update message from a neighboring cluster will update the information about its neighboring cluster, but will not reset its timer for entering the comparing state. This is to ensure that the timer of a cluster head will eventually expire.

It is worth noting that before all clusters are merged into one, each cluster already has its own energy-efficient traffic-aware detour tree. It is used by neighboring cluster heads to exchange the merge request/response as well as the cluster and update information. These data packets are routed through tree routing. After the detour tree construction is completed, every node in the tree knows its convex hull and packets can be forwarded based on the corresponding convex hull.

Our energy-efficient traffic-aware detour tree algorithm is distributed. Cluster merging are localized operations, which do not necessarily require knowledge of the entire network. Compared to centralized algorithms, our distributed detour algorithm does not involve route discovery broadcast or dissemination of the route decision to the entire network. Clearly, our distributed algorithm is more efficient in practice. It not only avoids expensive broadcast storms and bottleneck events, but also significantly balance energy cost and improves efficiency.

3.2. Implementation of the Algorithm

There are two main design issues of this algorithm: What is an appropriate gravity function and how to deal with concurrent merge requests. In the following subsections, we address each of them separately.

3.2.1. The Gravity Function

To minimize the possibility of having conflicting hulls between siblings, the closer clusters should be given stronger gravity so that they are merged earlier. In addition, to alleviate the energy unbalance and bottleneck issues, the neighboring clusters with higher virtual product should be merged earlier so that the traffic between them will go through fewer links in the tree, while at the same time, energy utilization will be more optimal. Given two neighboring clusters *C1* and *C2*, these observations lead us to set the gravity function between them as followings:

$$gravity(C_1, C_2) = \frac{bandwidth(C_1, C_2) \times energy(C_1, C_2)}{dist(C_1, C_2)^2} \quad (1)$$

In the case where two nodes are neighbors of each other, similar to what has been defined in IEEE 802.11b specification [26], they will exchange beacons periodically. The beacon exchange implies that there is non-zero traffic between any pair of neighboring nodes. To guarantee beacon exchanges, non-zero residual energy of each individual node is required. Through this beacon exchange, nodes can also learn about the traffic and energy consumption patterns of their neighbors, which is necessary for detour tree construction. ETDT is appropriate for applications which maintain constant traffic patterns and stable energy cost for a certain period of time or where the change is gradual. To optimize transmission, we investigate the relationship between traffic pattern, residual energy and clusters' distance. The neighboring nodes with higher residual energy can support heavier traffic load between them. The shorter the distance between two clusters the greater the saving in transmission energy and indirectly facilitate higher traffic load between the clusters. From this analysis, it is obvious that in the gravity function, both clusters' distance and residual energy can better improve transmission. Putting those three factors, i.e. bandwidth, residual energy, and distance together into the gravity function for building detour trees will surely optimize overall performance of detour tree routing, since the gravity function determines the quality of detour tree and also the performance of detour tree routing. The Energy-Efficient Traffic-Aware Detour Tree (ETDT) is constructed according to this gravity function (Equation 1), under the assumption that there will be non-zero gravity between any pair of neighboring nodes. Consequently, if the network is connected, after a series of merges, all nodes should eventually be integrated into one cluster.

Since this detour routing tree is built based on network conditions at a certain time period, it needs to be reconstructed regularly. The length of reconstruction period depends on each individual application, for instance, when half of the nodes around a void area have the energy level reduced to only 50% of their initial energy, when the traffic pattern in the network has changed and when a new traffic pattern has lasted for a certain amount of time. Those events may trigger a detour tree reconstruction.

3.2.2. Concurrent Merge Requests

It is likely that, before the merge process finishes, more than one clusters' timer expire. In this case, there will be multiple merge requests sent out concurrently in the network. If the source and destination of these requests are different, these merges can be performed in parallel. Otherwise, the merge must be performed in a specific sequential order to avoid creating inconsistent states.

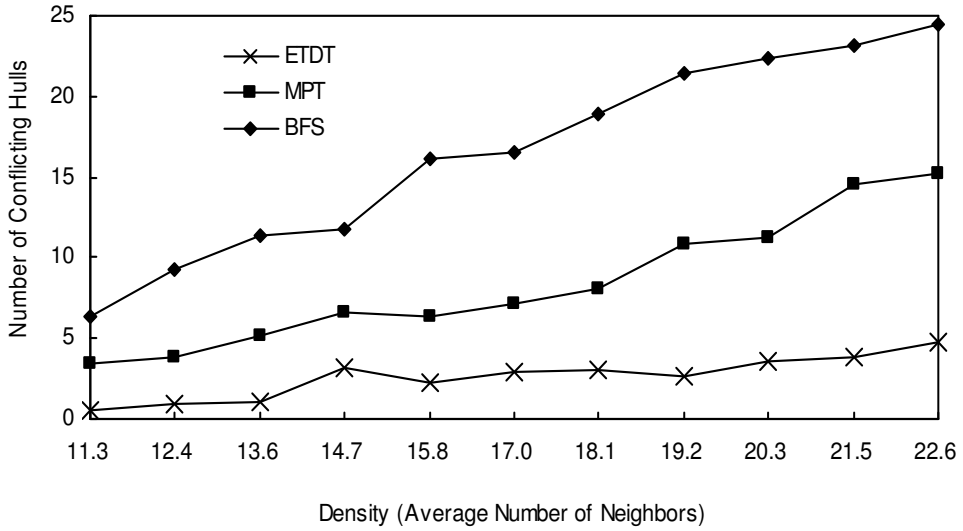

Figure 2. Number of Conflicting Convex Hulls under different Network Densities

In Table 1, when the head of a cluster decides to combine with another cluster, it asks for permission by sending out the merge request to the target cluster. These cluster heads will go through a process similar to the two-phase commit protocol [27]. The one sending out the merge request will go through the *requested* and *committed* states; while the other one will go through the *responded* and *committed* states. Furthermore, we regard the statements associated with the *committed* state as one atomic action. By incorporating these protection mechanisms, a cluster head will be involved in at most one merge request.

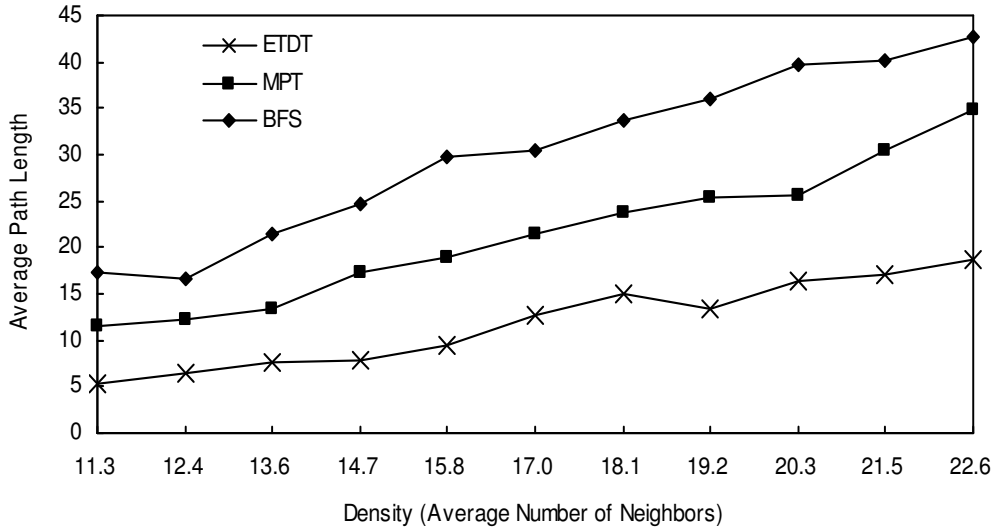

Figure 3. Average Path Length under different Network Densities

4. PERFORMANCE EVALUATION

In order to compare the performance of Energy-Eficient Traffic-Aware Detour Tree (ETDT) with other routing tree methods, such as Breadth First Search Tree (BFST) and Minimum Path Tree (MPT), we implement ETDT in Matlab. To show the capability of tree routing, the simulations are carried out in 3D networks.

4.1. Simulation Settings

In the simulation experiment, the transmission range is 300 meters. We randomly generate the topologies by placing nodes in a 3D cube with each side of length 1000 meters, according to a uniform distribution. The cube is wrapped around at both ends of each dimension to eliminate the edge effect. The total number of nodes placed in the cube ranges from 100 to 200, which corresponds to network densities from 8.5 to 17 neighbors per node. For each network density, we generate 10 topologies and use them to evaluate the performance of the three detour routing trees. For each pair of neighbors, the residual energy and available bandwidth of each node is randomly generated in the range from 1 and 500 with a uniform distribution. The initial energy of each node is 500.

4.2. Simulation Results and Analysis

Four metrics are used to evaluate the performance of routing trees. The first metric is the average number of conflicting hulls in the routing tree. After the routing trees are produced by the three algorithms, we calculate the convex hulls of the sub-trees and found the number of conflicting convex hulls among siblings. Each result in Figure 2 is the average of the number of conflicting hulls in 10 different topologies under the same network density. As discussed in Section I, more conflicting hulls result in more complicated routing. As shown in Figure 2, the number of conflicting hulls of ETDT is close to zero for all density levels and is consistently smaller than the other methods. The reason is that ETDT is built in a bottom-up fashion. Closer clusters are merged first so that the convex hulls between siblings are more clustered and have slim chance of overlapping. Therefore, it results in a geographically “compact” tree and gets rid of routing ambiguity.

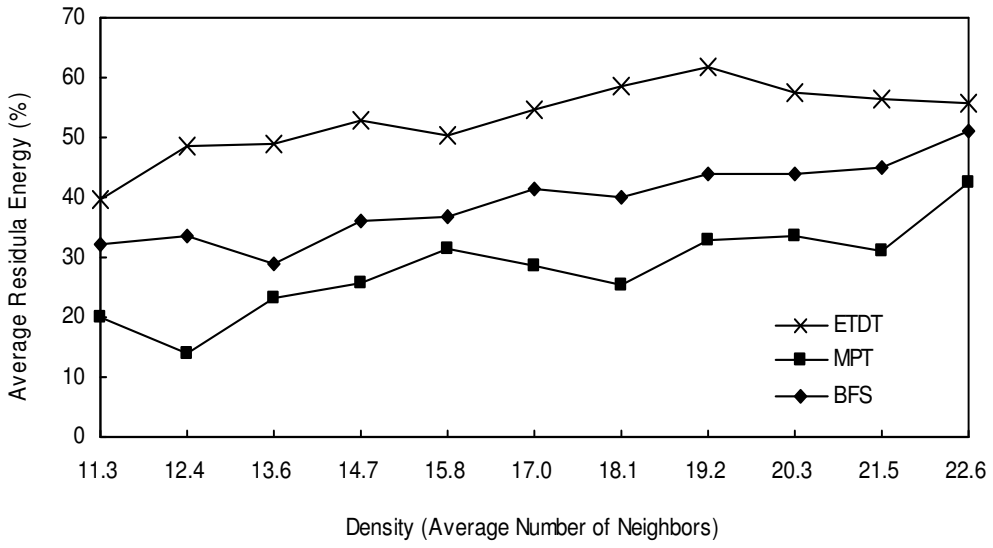

Figure 4. Average Residual Energy under different Network Densities

The second metric is the average path hops of the routing path, which is defined as the average

number of hops between different source and destination pairs. Figure 3 shows the average path length in term of hop numbers under different network densities. Ten source and destination pairs are randomly selected to perform routing for each generated network topology. For a given network density, each result in Figure 3, is obtained by averaging 100 source-destination pairs on top of 10 different network topologies. A smaller number of average path hops means less energy cost on each routing, which is more energy efficient and prolongs the system lifetime. In addition, a smaller number of path hops also means that the traffic can reach the destination faster. This will help reduce the delay and improve the system throughput. For the other case, the shortest paths from each node to the root forms MPT. The hops count between source and destination may not be the lowest because neither the source nor the destination may be the root. As shown in Figure 3, ETDT has the smallest average path hops. MPT has a lower average path hops than BFST, because it considers the location of the nodes. However, it seems that the location information is not fully utilized. In ETDT, the merging of clusters is determined by the distance and bandwidth between neighboring clusters. Thus, the resulting tree has lowest average path hops.

The third metric is the average path residual energy. A link residual energy is the sum of two vertex nodes' residual energy. On the routing path from a source to a destination, the smallest residual energy of any link in the path is defined as the average path residual energy. The larger the average path residual energy the lower is the chance of a network partition. Similar to the average path length, for a given network density, Figure 4 shows the average path energy of 100 source-destination pairs on top of 10 different network topologies. Figure 4 shows the ratio of average path residual energy to its initial energy under different network densities. Since the initial energy is same for all of the nodes, the higher the ratio the higher is average path residual energy. ETDT has the largest average path residual energy because clusters are merged with consideration for their residual energy. By merging clusters with more residual energy earlier, ETDT includes links with more residual energy in the detour tree and supports larger transmission traffic. An interesting observation is that although the average path residual energy of BFST is lower than ETDT, it is higher than MPT because a node in BFST includes all unvisited neighbors as its children during tree construction. This indirectly allows BFST to include links with more residual energy in the tree than MPT.

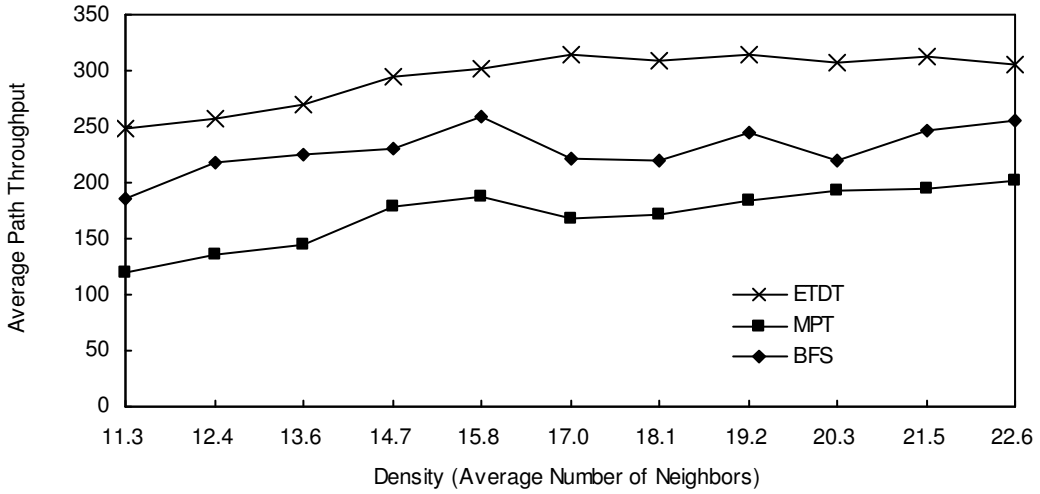

Figure 5. Average Path Throughput under different Network Densities

Our last evaluation metric is average path throughput. On a routing path from a source to a

destination, the smallest available bandwidth of any link in the path is defined as average path throughput. The larger the path throughput the lower is the chance of a bottleneck. For a given network density, Figure 5 shows the average path throughput of 100 source-destination pairs on top of 10 different network topologies and under different network densities. ETDT has the largest path throughput because clusters of nodes are merged based on the available inter-cluster bandwidth. By merging clusters with more bandwidth between them earlier, ETDT includes links with more bandwidth in the detour tree. Also, although the average path throughput of BFST is lower than ETDT, it is higher than MPT because a node in BFST includes all unvisited neighbors as its children during tree construction. This indirectly allows BFST to include links with more available bandwidth in the tree than MPT.

5. CONCLUSION AND FUTURE WORK

Because of its light-weight and efficient routing algorithm, geographic routing has been applied to many different applications of wireless sensor networks. The planarization detour method requires several impractical assumptions to work correctly. In [7], spanning tree routing has been shown to be a good alternative detour strategy for geographic routing. However, the quality of the pre-constructed spanning trees determines the performance of the spanning tree routing. Our proposed Energy-Efficient Traffic-Aware Detour Tree not only considers network traffic patterns, but also network traffic and residual energy. It is beneficial to consider the relationships between clusters' distance, residual energy and traffic pattern when building the detour routing tree since it improves routing performance compared to other routing tree algorithms, such as MPT and BFST. Moreover, this detour tree algorithm is distributed. Clusters merging are localized operations, which do not require the knowledge of the entire network, thus reducing network traffic. Unlike centralized algorithms, our distributed detour algorithm does not need to send information from all nodes to a centralized node, and then disseminate the routing decision to the entire network. Obviously, our distributed detour algorithm is more efficient in practice. This distributed algorithm does not cause expensive broadcast storms when building the detour tree and improves performance in the geographical routing detour mode. The simulation results confirm that the spanning tree detour routing based on ETDT achieves a much better routing performance in terms of the number of conflicting hulls, average path hops, average path residual energy and average path throughput.

Up to now no pre-constructed spanning tree is designed to be adaptive to network mobility. Even with slight changes of network topology, the whole routing tree may need to be reconstructed. In theory, when a link in ETDT is broken due to movements of nodes, the node at one end can do a local search to find a possible alternative link to reconnect the disconnected component. This option will be further investigated in our future work.

ACKNOWLEDGEMENTS

We express the special appreciation to both Dr. Min-Te Sun, assistant professor of Department of Computer Science and Software Engineering at Auburn University and Dr. Chunlei Liu, Associate Professor of department of Mathematics and Computer Science at Valdosta State University for their constructive criticism and helpful suggestions on this research.

REFERENCES

1. Perkins, C.E. and P. Bhagwat. *Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers*. in *Proceedings of the conference on Communications architectures, protocols and applications SIGCOMM '94*. 1994. London, United Kingdom.
2. Johnson, D.B. *Routing in Ad Hoc Networks of Mobile Hosts*. in *Proceedings of the Workshop on Mobile Computing Systems and Applications*. 1994. Santa Cruz, CA.
3. Johnson, D.B. and D.A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, in *Mobile Computing*, T. Imielinski and H. Korth, Editors. 1996, Kluwer Academic Publishers. p. 153-181.
4. Perkins, C.E. and E.M. Royer. *Ad hoc On-Demand Distance Vector Routing*. in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*. 1999. New Orleans, LA.
5. Vincent, D.P. and M.S. Corson. *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. in *Proceeding of IEEE Conference on Computer Communications, INFOCOM*. 1997. Kobe, Japan.
6. Clausen, T., et al. *Optimized Link State Routing Protocol*. in *Proceeding of IEEE International Multi Topic Conference 2001: technology for the 21st century*. 2001. Lahore University of Management Sciences, Pakistan.
7. Pei, G.Y., M. Gerla, and T.W. Chen. *Fisheye state routing: a routing scheme for ad hoc wireless networks*. in *Proceeding of IEEE International Conference on Communications (ICC2000)*. 2000. New Orleans, LA.
8. Haas, Z.J. and M.R. Pearlman, *The performance of query control schemes for the zone routing protocol.* IEEE/ACM Transactions on Networking, 2001. **9**(4): p. 427 - 438.
9. Ramasubramanian, V., Z.J. Haas, and E.G. Sirer. *SHARP: a hybrid adaptive routing protocol for mobile ad hoc networks*. in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. 2003. Annapolis, Maryland, U.S.A.
10. Ko, Y.B. and N.H. Vaidya, *Location-aided routing (LAR) in mobile ad hoc networks. Wireless Networks*, 2000. **6**(4): p. 307-321.
11. Karp, B. and H.T. Kung. *GPSR: Greedy perimeter stateless routing for wireless networks*. in *Proceedings of the 6th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*. 2000. Boston, MA.
12. Kuhn, F., et al. *Geometric ad-hoc routing: of theory and practice*. in *Proceedings of the 22nd ACM Annual Symposium on the Principles of Distributed Computing (PODC 2003)*. 2003. Boston, MA.
13. Leong, B., B. Liskov, and R. Morris. *Geographic Routing Without Planarization*. in *Proceedings of the 3rd Symposium on Networked Systems Design & Implementation (NSDI 2006)*. 2006. San Jose, CA.
14. Kim, Y.J., et al. *Geographical Routing Made Practical*. in *Proceedings of the 2nd Annual Symposium on Networked Systems Design and Implementation*. 2005. Boston, MA.
15. Karp, B., *Geographic Routing for Wireless Networks*, in *Computer Science*. 2000, Harvard University: Cambridge.
16. Li, X., et al. *Multi-dimensional range queries in sensor networks.* in *Proceedings of ACM SenSys 2003*. 2003. Los Angeles, California, U.S.A.
17. Ratnasamy, S., et al. *GHT: A geographic hash table for data-centric storage in sensornets*. in *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*. 2002. Atlanta, Georgia, U.S.A.
18. Newsome, J. and D. Song. *GEM: Graph EMbedding for routing and data-centric storage in sensor networks without geographic information*. in *Proceedings of ACM SenSys 2003*. Los Angeles, California,U.S.A.

19. Rao, A., et al. *Geographic routing without location information.* in *Proceedings of the Ninth Annual International Conference on Mobile Computing and Networking, (Mobicom).* 2003. San Diego, California,U.S.A.
20. Garbriel, K. and R. Sokal, *A new statistical approach to geographic variation analysis. Systematic Zoology*, 1969. **18**: p. 259-278.
21. Toussaint, G., *The relative neighborhood graph of a finite planar set. Pattern Recognition*, 1980. **12**(4): p. 261-268.
22. Liu, S.P. and L. Cheng. *Local Tree Based Geometric Routing*. in *Proceeding of IEEE International Conference on Communications,(ICC2007)*. 2007. Glasgow, Scotland.
23. Kuhn, F., R. Wattenhofer, and A. Zollinger. *Worst-Case Optimal and Average-case Efficient Geometric Ad-hoc Routing*. in *Proceedings of the Fourth ACM International Symposium on Mobile and Ad hoc Networking & Computing (MobiHoc'03)*. 2003. Annapolis, Maryland.
24. Kim, Y.J., et al. *On the Pitfalls of Geographic Face Routing*. in *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing (DIALM-POMC'05)*. 2005. Cologne, Germany.
25. Cormen, T.H.L., C.E. Rivest, R.L. and C. Stein, *Introduction to Algorithms*. second ed. 2001.
26. IEEE, ed. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*. 2003, The Institute of Electrical and Electronics Engineers, Inc.: New York, NY. 1-89.
27. Papadimitriou, C.H., *The theory of database concurrency control*. 1986: Computer Science Press. 239.

Authors

Lei Zhang is an assistant professor of computer science, Frostburg State University, University System of Maryland. She received M.S and Ph. D of Computer Science from Auburn University in 2005 and 2008. Before she pursued her graduate study in U.S, she worked as an instructor at school of Electrical Engineering & Automation, Tianjin University, China. Her current research interests include Wireless Networks Protocols Design and Applications, Distributed Algorithm, Information Security, Data Mining and Human Computer Interaction. She is a member of IEEE/ACM. She has been a technical reviewer for numerous international journals and conferences.

Deying Li is a Professor of Computer Science, Renmin University of China, China. She received his Ph.D of computer science from City University of Hong Kong in 2004. Her research interests include wireless networking, algorithm design and analysis. She has been the associate editor of Discrete Mathematics, Algorithms and Applications (DMAA).

Alvin S. Lim is an Associate Professor of Computer Sciences, Auburn University, Auburn, AL. He received his Ph. D of computer science from The University of Wisconsin-Madison. His research interests include self-organizing networks, wireless mobile networks, high performance networks, mobile computing and databases, reliable and dynamically reconfigurable distributed systems, complex software development, parallel processing, and performance measurements and analysis. He has been the technical reviewer and editor for numerous international journals and conferences.