Geographic Routing with Low Stretch in d-dimensional Spaces[*]

Simon S. Lam and Chen Qian
Department of Computer Science
The University of Texas at Austin
Austin, Texas 78712

TR-10-03 January 19, 2010
Revised, April 21, 2010

ABSTRACT

Geographic routing is attractive because the routing state needed per node is independent of network size. We present a novel geographic routing protocol with several major advances over previous geographic protocols. First, our protocol achieves an average routing stretch close to 1. Second, our protocol can be used for nodes located in d-dimensional Euclidean spaces ($d \geq 2$). Third, node locations are specified by coordinates which may be accurate, inaccurate, or arbitrary. Conceptually, our routing structure consists of a Delaunay triangulation (DT) overlay on an *arbitrary connectivity graph*. We refer to the structure as a *multi-hop DT*. Greedy routing in a correct multi-hop DT provides guaranteed delivery.

We present join, leave, failure, maintenance, and initialization protocols, named *MDT protocols*, for constructing and maintaining a multi-hop DT using *soft states*. The join protocol is proved to be correct for serial joins. When a system is under churn, nodes may join, leave, and fail concurrently. Our experiments show that MDT's routing success rate is close to 100% for systems under churn and node states converge to a correct multi-hop DT after churn. MDT is scalable to large networks. We present performance comparisons of MDT versus several geographic (and one non-geographic) routing protocols for nodes in 2D and 3D.

1. INTRODUCTION

Geographic routing (also known as location-based or geometric routing) is a promising approach to scalable routing in large networks. Most geographic routing protocols have been designed for nodes in a 2D plane with accurate location information. In reality, many network applications run on nodes located in 3D spaces [1, 3, 8, 9]. Furthermore, node location information may be highly inaccurate or simply unavailable. In the latter case, d-tuple virtual coordinates ($d \geq 2$) can be used to specify node locations for geographic routing in d-dimensional virtual spaces [23, 27].

In this paper, we present a novel geographic routing protocol for a network of nodes in a d-dimensional Euclidean space,[1] for integer $d \geq 2$. Network nodes are identified by their locations specified by coordinates. The graph of nodes and physical links, assumed to be connected, will be referred to as the *connectivity graph*.

Delaunay triangulation (DT) [11] has a long history and many applications in different fields of science and engineering. For nodes (points) in a 2D plane, Bose and Morin proved that greedy routing in a DT always finds a given destination node [4]. Lee and Lam [17, 18] extended this result and proved that given a destination location ℓ in a d-dimensional space, $d \geq 2$, greedy routing in a DT always finds a node that is closest to ℓ. The above results hold for node locations specified by accurate, inaccurate, or arbitrary coordinates.

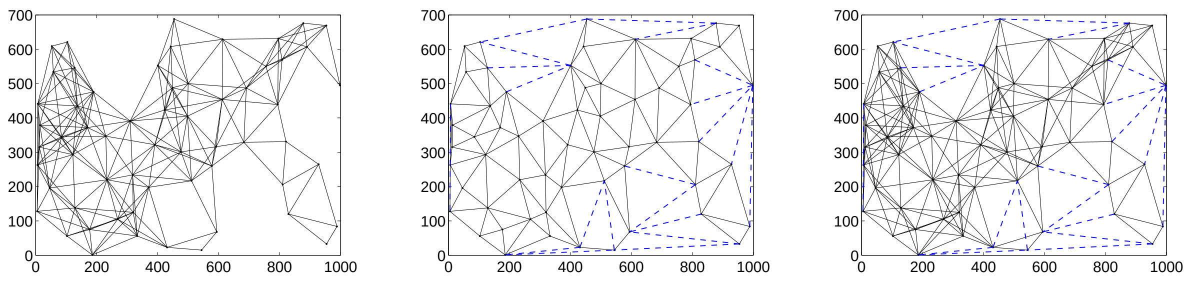
DT has not been successfully applied to wireless routing in the past due to the following problem: Two neighbors in a DT may not be able to communicate directly with each other for various reasons, e.g., there is an obstacle between them, the distance between them exceeds the radio transmission range, etc. That is, for most wireless networks, the DT graph is not a subgraph of the connectivity graph, as illustrated in Figures 1(a)-(b) where dashed lines are DT edges between nodes that are not connected by physical links. To solve this problem, we have designed and evaluated a protocol suite, named MDT, for a dynamic set of nodes to construct and maintain a correct multi-hop DT overlay on an arbitrary connectivity graph. In a multi-hop DT, two nodes may be multi-hop neighbors which communicate via a virtual link, i.e., a path provided by *soft-state* forwarding tables[2] in nodes along the path

Even though the design of MDT was initially motivated by wireless networks, we note that MDT can be used for geographic routing in wireline networks also. This is because MDT routing has been designed to run correctly in any connected graph of nodes and physical links in a d-dimensional space, for $d \geq 2$.

We have proved that for a given destination location ℓ, MDT routing in a correct multi-hop DT provides *guaran-*

[*]Research sponsored by National Science Foundation grant CNS-0830939.

[1]Hereafter, whenever we say "d-dimensional space", we refer to a d-dimensional Euclidean space.
[2]Inspired by Ethernet switch tables but implemented differently.



(a) Connectivity graph (b) DT graph (c) MDT graph

Figure 1: An illustration of connectivity, DT, and MDT graphs of a set of nodes in 2D

teed delivery to a node that is closest to ℓ. MDT routing can achieve a very *low routing stretch* because its routing structure is "rich" in links. MDT routing uses all of the physical links with additional virtual links connecting multi-hop DT neighbors (dashed lines in Figure 1(c)). In comparison, other protocols use only physical links and, in recovery mode, a subset of physical links [5, 14, 15].

The MDT suite consists of protocols for routing, join, leave, failure, maintenance, and system initialization. The MDT join protocol has been proved correct for a single join. (Thus it constructs a correct multi-hop DT when nodes join serially.) The maintenance protocol enables concurrent joins at system initialization. The join and maintenance protocols are sufficient for a system under churn to provide a routing success rate close to 100% and for node states to converge to a correct multi-hop DT after churn. The leave and failure protocols are used to improve accuracy and reduce communication cost.

MDT protocols are *communication efficient* because MDT does not use flooding to discover DT neighbors. MDT's search technique is also not limited by a maximum hop count (needed in scoped flooding used by many wireless routing protocols) and is guaranteed to succeed when the existing multi-hop DT is correct.

The balance of this paper is organized as follows. In Section 2, we provide an overview of related work. In Section 3, we present concepts, definitions, and our model assumptions. In Section 4, we present the MDT routing protocol and a theorem stating that it provides guaranteed delivery in a correct multi-hop DT. In Section 5, we present join, maintenance, initialization, leave, and failure protocols and a theorem that the join protocol is correct for a single join. We present experimental results to demonstrate MDT's speed to construct a correct multi-hop DT for a large number of nodes in 3D at system initialization. We evaluate the performance of MDT routing from experiments for nodes in 3D with inaccurate coordinates and randomly placed obstacles, without churn and with churn. In Section 6, we present experimental results to compare the routing performance of MDT with geographic protocols designed for 2D and 3D and a non-geographic protocol, VRR [6]. Lastly, we compare the communication costs of constructing a correct multi-hop MDT versus two other graph construction algorithms [15, 20]. We conclude in Section 7.

2. RELATED WORK

Routing research has a vast literature. We limit this review mostly to related work on the geographic approach. Almost all geographic routing protocols have been designed for nodes located in a 2D plane using greedy routing. For a general connectivity graph, greedy routing may be "stuck" at a node that is a local minimum, i.e., it is closer to the destination than any of its neighbors. When a packet is stuck at a node, two of the earliest protocols, GFG [5] and GPSR [14], use the idea of face routing to move the packet out of the local minimum. These protocols provide guaranteed delivery for a planar graph. If the connectivity graph is not planar, a planarization algorithm (such as GG [12] or RNG [26]) is used to disallow some links such that the nodes and remaining links form a planar subgraph. For GG and RNG algorithms to successfully construct a connected planar subgraph, it is required that the original connectivity graph satisfies the unit disk graph model and node location information be accurate.

The DT of a set of nodes in a 2D plane is a graph that has been shown to be a good spanner with a constant stretch factor [7]. Furthermore, greedy routing in a DT graph provides guaranteed delivery for nodes in a 2D plane [4]. However, some edges in the DT graph may be arbitrarily long and exceed the radio transmission range. The restricted DT graph proposed in [13] and the *k*-localized DT graph proposed in [21] are approximations of the true DT graph. They were shown to be good spanners with constant stretch factors. However, being DT approximations, they do not provide guaranteed delivery. In another approach [28], the need to solve the problem of long DT edges is obviated by constraining node locations such that the DT graph of the nodes is a subgraph of the connectivity graph.

In each of the geographic routing protocols cited above, the unit disk graph model is assumed. If node locations are specified by inaccurate coordinates (e.g., due to localization errors), the subgraphs constructed by GG and RNG have cross links and are no longer planar. As a result, face routing may not move packets out of local minima, resulting in routing failures. Some fixes to reduce the number of routing failures are presented in [24].

For a practical wireless network with obstacles between nodes and using real radios, the assumption of a unit disk graph model cannot be justified. A quasi unit disk graph model is proposed in [16]. Kim et al. [15] took a major step away from the unit disk graph model. They proposed the CLDP protocol which, given a connected graph, produces a subgraph in which face routing would not cause routing failures. When stuck at a local minimum, GPSR routing uses the subgraph produced by CLDP instead of by GG or RNG.

Leong et al. proposed the GDSTR protocol [20] which can also be used for any connectivity graph. A packet is routed greedily until it is stuck at a local minimum. The packet is then routed in a distributed spanning tree until it reaches a point where greedy routing can again make progress.

For a network of nodes without location information, No-Geo [23] was proposed to use node locations specified by virtual coordinates which are constructed to reflect the underlying connectivity. A packet is forwarded by greedy routing based on virtual coordinates. When a packet is stuck at a local minimum, expanding ring search is used to find a way out.

All of the geographic protocols referenced above were designed for routing in 2D. For routing in 3D, there is no analog of face routing in 2D. Durocher et al. [8] showed that there is no local routing protocol that provides guaranteed delivery, even with the assumptions of a unit ball graph model and accurate location information. For geographic routing in 3D, GRG [9] uses greedy routing with randomized recovery to lead packets out of local minima. A routing approach using d-tuple virtual coordinates, for $d \geq 5$, was proposed in [27].

We mention just a few references in the non-geographic routing literature. VRR [6] uses random unsigned integers to identify nodes and organize them in a virtual ring. Each node maintains a virtual neighbor set and physical neighbor set. VRR sets up and maintains a routing path between each pair of virtual neighbors. VRR routing table entries are maintained as hard states. Note that MDT is similar to VRR in maintaining virtual links to DT neighbors, but MDT forwarding table entries are maintained as soft states.

Another recent protocol, S4 [22] which, using ideas from BVR [10] and compact routing [25], provides a worst-case routing stretch of 3 and an average stretch close to 1. S4 requires a routing state per node of $O(\sqrt{N})$ which is very good for non-geographic routing protocols, but it is not independent of N like geographic protocols.

The prior work most relevant to this paper is by Lee and Lam [17, 18, 19]. Their protocols for constructing and maintaining a correct distributed DT of nodes in a d-dimensional space ($d \geq 2$) provide a basis for our work in this paper. Their protocols, however, were designed with the assumption that every DT node can directly communicate with every other DT node. For arbitrary connectivity graphs, direct communication between every pair of neighbors in a DT is impossible.

3. CONCEPTS AND DEFINITIONS

Consider a set S of nodes in a d-dimensional space ($d \geq 2$). Each node in S is identified by its location specified by coordinates. There is at most one node at each location. The Delaunay triangulation of S, denoted by $DT(S)$, is a graph whose vertices are nodes in S.[3] When we say node u *knows* node v, node u knows node v's coordinates. Coordinates may be accurate, inaccurate, or arbitrary.

3.1 Distributed DT

Definition 1. A distributed DT of a set S of nodes is specified by $\{< u, N_u > | u \in S\}$, where N_u represents the set of u's neighbor nodes, which is locally determined by u.

Definition 2. A distributed DT is **correct** if and only if for every node $u \in S$, N_u is the same as the neighbor set of u in $DT(S)$.

Using protocols in [18, 19], each node, $u \in S$, finds a set C_u of nodes (C_u includes u). Then u computes $DT(C_u)$ locally to determine its set N_u of neighbor nodes. Note that C_u is local information of u while S is global knowledge. For the extreme case of $C_u = S$, u is guaranteed to know its neighbors in $DT(S)$. However, the communication cost for each node to acquire knowledge of S would be very high. A *necessary and sufficient condition* for a distributed DT to be *correct* is that for all $u \in S$, C_u includes all neighbor nodes of u in $DT(S)$. This theorem was presented in [18] with a proof published later in [19].

3.2 Model assumptions

Two nodes connected by a physical link are said to be *physical neighbors*. Each link is *bidirectional*. The connectivity graph may be arbitrary as long as it is a connected graph. To simplify protocol descriptions, we assume that each link provides reliable message delivery. (In a practical implementation, additional mechanisms such as ARQ should be used to ensure reliable message delivery.) We assume a fail-stop model. When a node fails, it becomes silent.

3.3 Multi-hop DT

A multi-hop DT is specified by $\{< u, N_u, F_u > | u \in S\}$, where F_u is a soft-state forwarding table, and N_u is u's neighbor set which is derived from information in F_u. The multi-hop DT model generalizes the distributed DT model by relaxing the requirement that every node in S be able to communicate directly with each of its DT neighbors. In a multi-hop DT, two nodes that are multi-hop neighbors communicate via a path provided by forwarding tables in nodes along the path.

[3]See [11]. Familiarity with the $DT(S)$ definition and algorithms for computing the $DT(S)$ graph is not needed for reading this paper.

For a node u, each entry in its forwarding table F_u is a 4-tuple $< source, pred, succ, dest >$, which is a sequence of nodes with *source* and *dest* being the source and destination nodes of a path, and *pred* and *succ* being node u's predecessor and successor nodes in the path. In a tuple, *source* and *pred* may be the same node; also, *succ* and *dest* may be the same node. A tuple is used by u for message forwarding from *source* to *dest* or from *dest* to *source*. For a specific tuple t, we use $t.source$, $t.pred$, $t.succ$, and $t.dest$ to denote the corresponding nodes in t.

For ease of exposition, we assume that a tuple and its "reverse" are inserted in and deleted from F_u as a pair. For example, $< a, b, c, d >$ is in F_u if and only if $< d, c, b, a >$ is in F_u. (In practice, only one tuple is stored with each of its two endpoints being both source and destination.) A tuple in F_u with u itself as the source is represented as $< -, -, succ, dest >$, which does not have a reverse in F_u.

For an example of a forwarding path, consider the multi-hop DT in Figure 2(c). The DT edge between nodes g and i is a virtual link; messages are routed along the paths, $g-e-h-i$ and $i-h-e-g$, using the following tuples: $< -, -, e, i >$ in node g, $< g, g, h, i >$ in node e, $< g, e, i, i >$ in node h, and $< -, -, h, g >$ in node i.

For a node u, its physical neighbors can be in one of three stages of the join process:

1) For a physical neighbor v that has booted up but has not yet joined the DT, the tuple, $< -, -, v, - >$, is stored in F_u. (These tuples are stored in and deleted from F_u by a link management protocol which will not be explicitly specified.)

2) A physical neighbor v that has sent a join request and received a join reply from a DT node[4] will notify all of its physical neighbors to change their tuple for v from $< -, -, v, - >$ to $< -, -, v, v >$ which indicates that v has found a closest node guaranteed to be v's neighbor in the global DT. We use P_u to denote the set $\{v \mid < -, -, v, v > \in F_u\}$. Each node in P_u is referred to as a *physical neighbor of u attached* to DT. Note that when a node first becomes a physical neighbor attached to DT, it is not a DT node. It becomes a DT node later after it has finished its join protocol execution. A multi-hop DT is correct only if all nodes have become DT nodes. If a node in P_u receives a message to forward before its join is finished, the message is queued to be forwarded after the node becomes a DT node.

3) A physical neighbor in a correct multi-hop DT is a DT node but may not be a DT neighbor, e.g. nodes a and h in Figure 2(c) are physical but not DT neighbors. On the other hand, a DT neighbor may not be a physical neighbor, e.g., nodes a and j in Figure 2(c). A node v that is in both P_u and N_u of node u is said to be a *one-hop DT neighbor* of u or, simply, a *one-hop neighbor*.[5]

[4]A *DT node* is one that has finished its join protocol execution.

Tuples in F_u are maintained as **soft states**. Each tuple is *refreshed* whenever there is packet traffic (e.g., application data or keep-alive message) between its endpoints. A tuple that is not refreshed will be deleted when its timeout occurs.

Definition 3. A multi-hop DT of S, $\{< u, N_u, F_u > \mid u \in S\}$, is **correct** if and only if the following conditions hold: i) the distributed DT of S, $\{< u, N_u > \mid u \in S\}$, is correct; and ii) for every neighbor pair (u, v), there exists a unique k-hop path between u and v in the forwarding tables of nodes in S, where k is finite.

The systems we consider are sometimes under churn when nodes join, leave, and fail concurrently. To define a metric for quantifying the accuracy of a multi-hop DT, we consider a node to be *in-system* from when it has finished joining until when it starts leaving or has failed. Let $MDT(S)$ denote a multi-hop DT of a set S of in-system nodes. Let $N_c(MDT(S))$ be the total number of correct neighbor entries and $N_w(MDT(S))$ be the total number of wrong neighbor entries in the forwarding tables of all nodes. A neighbor v in N_u is correct when u and v are neighbors in $DT(S)$ and wrong when u and v are not neighbors in $DT(S)$. Let $N_{edges}(DT(S))$ be the number of edges in $DT(S)$. Let $N_{np}(MDT(S))$ be the number of edges in $DT(S)$ that do not have forwarding paths in the multi-hop DT of S. The accuracy of $MDT(S)$ is defined to be:

$$\frac{N_c(MDT(S)) - N_w(MDT(S)) - 2 \times N_{np}(MDT(S))}{2 \times N_{edges}(DT(S))} \qquad (1)$$

It is straightforward to prove that the accuracy of $MDT(S)$ is 1 (or 100%) if and only if the multi-hop DT of S is correct.

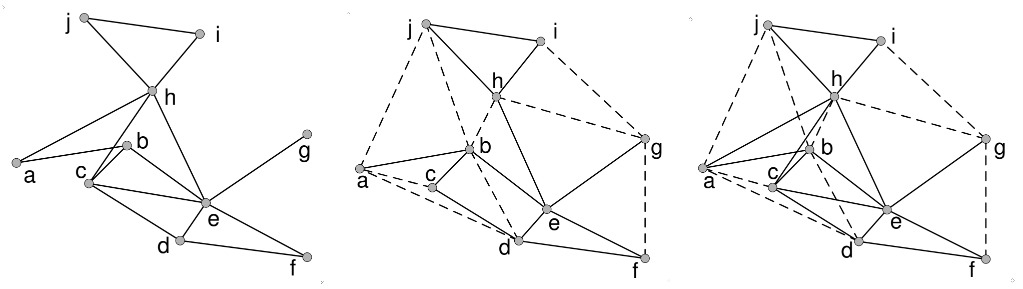
4. MDT ROUTING PROTOCOL

The MDT routing protocol pseudocode is shown in Figure 3. Consider a node u that has received a data message m to route. Node u first compares its own location with the message's destination location. If they are *not* equal, it calls $Routing(m)$ which calls $Get\_Next(m.dest, m.relay)$.

When Get_Next runs, it first checks these two cases: (i) a node in P_u exists at the destination location; (ii) there is a relay node to forward the message to. If neither (i) nor (ii) applies, then it performs *greedy routing* to determine the next-hop node. Conceptually, when a packet is stuck at a local minimum, MDT routing moves the packet along a virtual link to a multi-hop DT neighbor that is closest to the destination location. There are three possible outcomes:

a) Node v, a physical neighbor in P_u is closer to the destination location than any node in $P_u \cup \{u\}$ (line 9). Node u then transmits m directly to v.

[5]We use "neighbor" to refer to a DT neighbor. A node knows only neighbors in its locally computed DT. If the multi-hop DT is correct, then local DT neighbors are the same as neighbors in $DT(S)$.



(a) Connectivity graph of ten nodes (b) DT graph of ten nodes (c) MDT graph of ten nodes

Figure 2: Graphs for the join protocol example

```
Data message format
Node u stores message m with the format:
m = <m.dest, m.source, m.relay, m.data> in a local data
structure, where m.dest is the destination location,
m.source is the source node, m.relay is the relay node, and
m.data is the payload of the message.

Routing(m): Node u receives message m to route
        // u ≠ m.dest
1.      v ← Get_Next(m.dest, m.relay)
                // m.relay may be changed when Get_Next returns
2.      if v ≠ null then
3.          Transmit m to v
4.      else
5.          exit   // u is closest to m.dest
6.      end if

Get_Next(dest, relay): Node u finds the next-hop node
1.      if there exists v | v ∈ P_u and v = dest then
2.          return v   // a physical neighbor attached to DT exists at dest
3.      end if
4.      if relay ≠ null and relay ≠ u then
                        // forward message to the relay
5.          t ← tuple in F_u such that t.dest = relay
6.          return t.succ
7.      end if
            // perform greedy routing in multi-hop DT
8.      v ← node in P_u ∪ {u} closest to dest
9.      if v ∈ P_u then   // v is a physical neighbor attached to DT
10.         relay ← null
11.         return v
12.     else   // u is closer to dest than any node in P_u
13.         v ← node in N_u ∪ {u} closest to dest
14.         if v ∈ N_u then   // v is a multi-hop DT neighbor
15.             t ← tuple in F_u such that t.dest = v
16.             relay ← v
17.             return t.succ
18.         else   // u is the node closest to dest
19.             return null
20.         end if
21.     end if
```

Figure 3: MDT routing protocol at node *u* for a multi-hop DT

b) Node v, a multi-hop DT neighbor, is closer to the destination location than any node in $N_u \cup \{u\}$ (line 14). Node u writes v into the relay field of the message (line 16), and looks up its forwarding table to get the successor node in the path to v (line 17). (A correct multi-hop DT guarantees that a path from u to v exists in the forwarding tables of u and nodes along the path to v.)

c) Node u is closer to the destination location than any neighbor in $P_u \cup N_u$ (line 18).

We have proved Theorem 1 which states that MDT routing in a correct multi-hop DT provides guaranteed delivery. A proof of the theorem is presented in the Appendix.

THEOREM 1. *Consider a correct multi-hop DT of a finite set S of nodes in a d-dimensional Euclidean space. Given a location ℓ in the space, the MDT routing protocol succeeds to find a node in S closest to ℓ in a finite number of hops.*

5. MDT PROTOCOL SUITE

In addition to the routing protocol, MDT includes join, leave, failure, and maintenance protocols, which make use of basic protocol steps in [18, 19] for a distributed DT. There are, however, major innovations in MDT protocol design:

1) *Construction of forwarding paths*: In addition to constructing and maintaining a distributed DT, join and maintenance protocols in MDT insert tuples into forwarding tables and update some existing tuples to correctly construct paths between multi-hop neighbors. Leave, failure and maintenance protocols in MDT construct a new path between two multi-hop neighbors whenever the previous path between them has been broken due to a leave or failure. Protocol design to perform these tasks poses significant challenges.

2) *Soft state versus hard state*: For protocols in [18, 19], each node, say u, stores nodes it knows in its candidate set C_u. Nodes in C_u are maintained as hard states. The neighbor set, N_u, is derived from computing $DT(C_u)$ locally. For MDT, each node, say u, stores tuples in its forwarding table F_u. Tuples in F_u are maintained as soft states. The neighbor set N_u is derived from computing $DT(C_u^*)$ where $C_u^* = \{u\} \cup \{v \mid v = t.dest,\ t \in F_u\}$. Note that when a tuple's timeout occurs because it has not been refreshed by its endpoints, the tuple's destination is removed from C_u^*.

3) *Leave and failure notifications*: For protocols in [18, 19],

when a node, say v, leaves or is detected to have failed, each node in N_v is notified of its leave/failure by a unicast message. The leave/failure notification is also propagated to nodes that are not in N_v by a greedy reverse path broadcast (GRPB) protocol. Since MDT uses soft states, nodes not in N_v do not have to be notified of v's departure.

4) *System initialization protocols*: MDT includes two initialization protocols for constructing a correct multi-hop DT for a large number of nodes, one for serial joins and the other for concurrent joins.

In our protocol descriptions to follow, we keep the candidate set notation, C_u, for node u to store newly learned nodes. The C_u notation is kept for two reasons. First, having C_u makes it easier to understand MDT protocols and relate them to protocols in [18, 19]. Second, node u needs a place to temporarily store newly learned nodes and storing them in C_u is as good as any alternative. In MDT protocols, however, nodes in C_u are *soft states*. A node in C_u is deleted if (i) it does not become the destination of a tuple in F_u within a timeout period, or (ii) it is $t.dest$ for a tuple t that has not been refreshed and is deleted from F_u.

5.1 Join protocol

We begin by describing the basic steps of the join protocol in [17, 18].[6] Consider a new node, say w. It boots up and discovers its physical neighbors. If one of the physical neighbors is a DT node (say v) then w sends a join request to v. The join request is forwarded by greedy routing to a DT node (say z) closest to w. Node z sends a join reply to w which then sends a neighbor-set request to z for mutual neighbors of w and z in $DT(C_z)$.

When w receives the neighbor-set reply from z, w adds the mutual neighbors (if any) to its candidate set, C_w, and computes its neighbor set, N_w. If w finds new neighbors in N_w, w sends neighbor-set requests to them for mutual neighbors. The joining node w repeats the above process recursively until it cannot find any more new neighbor in N_w. At this time w has successfully joined and become a DT node.

Path construction to closest node. For a multi-hop DT, a node w can join when it has a physical neighbor v that is a DT node.[7] Node w joins by sending a join request to node v. MDT routing is used to forward the join request to node z that is closest to w. A forwarding path between w and z is constructed as follows. When w sends the join request to v, it stores the tuple $< -,-,v,v >$ in its forwarding table. Subsequently, suppose an intermediate node (say u) receives the join request from a one-hop neighbor (say v) and forwards it to a one-hop neighbor (say e), the tuple $< w,v,e,e >$ is stored in F_u.

When node z receives the join request of w from a one-hop neighbor (say d), it stores the tuple $< -,-,d,w >$ in its forwarding table for the reverse path. The join reply is forwarded along the reverse path from z to w using tuples stored when the join request traveled from w to z earlier. Additionally, each such tuple is updated with z as an endpoint. For example, suppose node x receives a join reply from z to w from its one-hop neighbor e. Node x changes the tuple $< e,e,*,w >$ in F_x to $< z,e,*,w >$, where $*$ denotes any node already in the tuple.

After node w has received the join reply, it notifies each of its physical neighbors that w is now attached to DT and they should change their tuple for w from $< -,-,w,- >$ to $< -,-,w,w >$.

Physical-link shortcuts. The join reply message, at any node along the path from z to w (including node z), can be transmitted directly to w if node w is a physical neighbor (i.e., for message m, there is a tuple t in the forwarding table such that $t.succ = m.dest$). If such a physical-link shortcut is taken, the path previously set up between z and w is changed. Tuples with z and w as endpoints stored by nodes in the abandoned portion of the previous path will be deleted because they will not be refreshed by the endpoints.

A physical-link shortcut can also be taken when other messages in MDT join, maintenance, leave, and failure protocols (to be presented) are forwarded, but they require the stronger condition, $t.succ = t.dest = m.dest$, that is, the shortcut can be taken only if $m.dest$ is a physical neighbor attached to DT.

Path construction to multi-hop DT neighbors. For a multi-hop DT, the join protocol needs to construct a forwarding path between the joining node w and each of its multi-hop neighbors. After node w has attached itself to DT, it sends neighbor-set requests and receives neighbor-set replies. When w learns a new node y from the join reply or a neighbor-set reply sent by some node, say x, how does w route a neighbor-set request to y that is more than one hop away? Our solution is to include a **relay** field in the neighbor-set request message. Node w sends a neighbor-set request to x, with x as the relay and y as the destination. Note that a forwarding path has already been established between w and x. Also, since x and y are DT neighbors, a forwarding path exists between x and y (assuming that w is joining a correct multi-hop DT). As the neighbor-set request is forwarded and relayed from w to y, tuples with w and y as endpoints are stored in forwarding tables of nodes along the path from w to y. The forwarding path that has been set up between w and y is then used by y to return a neighbor-set reply to w.

A pseudocode specification of the MDT join protocol is presented in the Appendix. Theorem 2 states that the MDT join protocol is correct for a single join. A proof of the theorem is presented in the Appendix.

THEOREM 2. *Let S be a set of nodes and w be a join-*

[6] We do not follow the ACE join protocol [19] because its correctness proof requires the general position assumption [11] and the assumption that the joining node is located within the convex hull of the existing DT.

[7] If node w has only physical neighbors, it will not start the join protocol until it hears from a physical neighbor who is attached to DT, e.g., it receives a token from such a node at system initialization.

ing node that is a physical neighbor of at least one node in S. Suppose the existing multi-hop DT of S is correct, w joins using the MDT join protocol, and no other node joins, leaves, or fails. Then the MDT join protocol finishes and the updated multi-hop DT of $S \cup \{w\}$ is correct.

Join protocol example. We present an illustration of the join protocol using Figure 2. Initially, let node a be a new node. The other 9 nodes are maintaining a correct multi-hop DT. When node a boots up, it discovers two physical neighbors, namely, nodes b and h, both of which are DT nodes. Node a transmits a join request to node b and stores the tuple $< -,-,b,b >$ in F_a. Node b runs MDT routing and transmits the join request to node c; it also stores the tuple $< a,a,c,c >$ in F_b. MDT routing *guarantees* to find a DT node closest to node a because the existing multi-hop DT is correct. The closest node happens to be c in this example. Node c stores the tuple $< -,-,b,a >$ in F_c and sends a join reply to a by transmitting it to b. Node b gets the join reply and transmits it to a (node b does not have to update its tuple $< a,a,c,c >$ in this particular example). When a gets the join reply, it updates F_a by replacing $< -,-,b,b >$ with $< -,-,b,c >$. Node a is now attached to DT and it notifies its physical neighbors, b and h.

Node c being closest to node a is guaranteed to be a neighbor of a in the DT of all ten nodes. Node a then tries to find all of its neighbors in the DT by first sending a neighbor-set request to c. Its tuple $< -,-,b,c >$ indicates that the request should be sent to b which then transmits it to c. When c gets the request, it computes its local DT to determine nodes that are mutual neigbors of c and a, which are nodes d and b. (See Figure 2(b).) Node a then sends neighbor-set requests to b and d. Node d replies that c is a mutual neighbor. Node b replies that c and j are mutual neighbors. Node a then sends a neighbor-set request to node j.

To establish a forwarding path between a and j, note that node a learns of node j from node b. A forwarding path is already established between a and b. Also, because the existing multi-hop DT is correct, a unique forwarding path exists between node b and node j, which is $b-e-h-j$. Therefore, node a sends a neighbor-set request to j by specifying b as the relay node in the message. The request is first sent to b which then forwards it to e on the $b-e-h-j$ path. At every node along the way, a tuple with endpoints a and j is stored in the node's forwarding table.

Note that the path, $a-b-e-h-j$, is very long. When the neighbor-set reply from j travels back via h, node h searches F_h and finds that node a is a physical neighbor (see Figure 2(c)). Node h then transmits j's reply directly to node a. (This is an example of a *physical-link shortcut*.) Subsequently, nodes a and j will select and refresh only the path $a-h-j$ between them. Tuples previously stored in nodes b, e, and h for endpoints a and j will be deleted upon timeout. Lastly, from j's reply, a learns that b is the only mutual neighbor of itself and j. Since a does not have any more new neighbor to query, its join protocol execution terminates and it becomes a DT node.

5.2 Maintenance protocol

For a system under churn, when nodes join, leave, and fail concurrently, node states may be incorrect. For a distributed DT to be correct, each node must know all of its neighbors in the global DT. To satisfy this condition, each node (say u) queries some of its neighbors to see if they know mutual neighbors that u does not know. The MDT maintenance protocol uses basic steps from the ACE join and maintenance protocols [19]. More specifically, node u selects a subset V of neighbors such that every simplex in $DT(C_u)$ including u also includes one node in V. Node u then sends a neighbor-set request to each node in V. When node u finds new neighbors in the neighbor-set replies, node u sends a neighbor-set request to each new neighbor x that satisfies the following condition:

C1. x is a vertex of a simplex in $DT(C_u)$ that includes u and does not include any node that has been sent a neighbor-set request.

Node u keeps sending neighbor-set requests until it cannot find any more new neighbor in N_u that satisfies C1. Node u then sends neighbor-set notifications to neighbors in N_u that have not been sent neighbor-set requests (these notifications do not require neighbor-set replies). The protocol code for constructing forwarding paths between node u and each new neighbor is the same as in the MDT join protocol.

If after sending a neighbor-set request to a node, say v, and a neighbor-set reply is not received from v within a timeout period, the node is deemed to have failed. Node u sends a failure notification about v to inform each node in u's updated neighbor set. These notifications are unnecessary since MDT uses soft states; they are performed to speed up convergence of node states.

Each node runs the maintenance protocol independently, controlled by a timeout value T_m. After a node has finished running the maintenance protocol, it waits for time T_m before starting the maintenance protocol again. The value of T_m should be set adaptively. When a system has a low churn rate, a large value should be used for T_m to reduce communication cost.

If every node runs the maintenance protocol repeatedly, the *node states converge to a correct multi-hop DT* because neighbors in a DT are connected by neighbor relations. A node can find all of its neighbors by following the neighbor relations [17].

5.3 Initialization protocols

We design two system initialization protocols to construct a correct multi-hop DT for a large set of nodes. As before, we assume that there is a link management protocol that enables each node to discover its physical neighbors. Also the graph of nodes and physical links is connected.

Serial joins by token passing. Starting with a one-node DT, other nodes join serially using the join protocol. The ordering of joins is controlled by the passing of a single token from one node to another. The token passing protocol ensures that the token visits every node in the set.

Concurrent joins by token broadcast. Starting from a one-node DT, other nodes join concurrently using the join and maintenance protocols. The ordering of joins is controlled by a token broadcast protocol. Initially, a token is installed in a selected node, which configures its state as a one-node DT. When a node has a token, it runs the join protocol once (except the selected node) and then the maintenance protocol, controlled by the timeout value T_m. It also sends a token to each physical neighbor that is not known to have joined the multi-hop DT (i.e., it is not a physical neighbor from which a token has been received and not $t.dest$ for some tuple t in its forwarding table). Each token is sent after a random delay uniformly distributed over time interval $[1, \tau]$, where τ is in seconds. If a node receives more than one token, any duplicate token is discarded. The token broadcast protocol provides at least one token to every node to start its join process.

5.4 Performance of MDT protocols

Evaluation methodology. Our performance criteria are routing success rate, routing stretch, resilience to churn, as well as storage and communication costs associated with routing. Since MDT can be used for a variety of networks, it is beyond the scope of this paper to evaluate metrics (e.g., throughput and end-to-end latency) that depend on link characteristics and congestion. Hence, we evaluate MDT protocols using a packet-level discrete-event simulator in which every protocol message created is routed and processed hop by hop from its source to destination. Queueing delays at a node, link errors, and transmission interference are not simulated. Instead, message delivery times from one node to the next are sampled from a uniform distribution over a specified time interval. To evaluate scalability of MDT, we performed experiments for up to 1,300 nodes in 3D. To evaluate resilience to churn, we performed experiments for churn rates up to 100 nodes/minute for 300-node networks in 3D.

Inaccurate coordinates. For each simulation experiment, we first locate nodes randomly in a 2D or 3D space. We then generate coordinates for these nodes, such that they have location errors specified by an **error ratio**, e, which is defined to be the ratio of the average location error to the average distance between physical neighbors.

Random Graph model. Given a set of nodes, connectivity graphs in our experiments are generated using a Random Graph model specified by two parameters, a *connection probability* p and a *transmission range* R. Two nodes that are more than R distance apart are not connected by a physical link. Two nodes that are less than or equal to R distance apart are connected by a physical link with probability p. With probability $(1-p)$, a physical link is missing between two nodes that are within transmission range of each other. For a wireless network, the probability $1-p$ is used to model *randomly placed obstacles* that block transmissions between pairs of nodes. The use of a limited R value challenges MDT protocols to correctly construct forwarding paths between DT neighbors that are far apart. In designing our experiments, the R value was varied to generate connectivity graphs with average node degrees comparable to those used in prior work [15, 20].

Note that this model generalizes the Bernoulli random graphs model in [15]. (If R is specified to be larger than the maximum distance between any pair of nodes, we get the model in [15].) These models are general in the sense that for a given set of nodes and R value, any possible connectivity graph may be generated with nonzero probability.

5.4.1 *Constructing a correct multi-hop DT at system initialization*

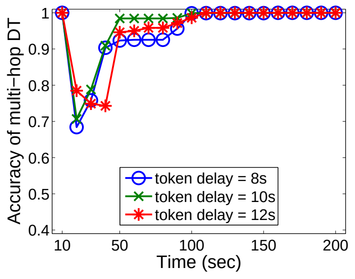
In Figure 4, we show simulation results from two sets of experiments for concurrent joins using token broadcast. In each experiment, 300 nodes are randomly distributed in a $800 \times 800 \times 800$ 3D space with transmission range $R = 325$. Connectivity graphs are generated from the Random Graph model with $e = 1$ and $p = 0.5$ (i.e., both inaccurate coordinates and randomly placed obstacles); the average node degree (number of physical neighbors per node) is 15.5. The first set of experiments is for low-speed networks in which one-hop message delays are sampled from 100 ms to 200 ms (average = 150 ms), with a maintenance protocol timeout duration of 1 minute. The second set of experiments is for high-speed networks in which one-hop message delays are sampled from 10 ms to 20 ms (average = 15 ms), with a maintenance protocol timeout duration of 10 seconds.

In the legend of Figure 4,“token delay” is maximum token delay τ. In every experiment, note that accuracy of the multi-hop DT is low initially when many nodes are joining at the same time. However, accuracy improves and converges to 100% accuracy quickly. In all experiments, after each node's initial join, the node ran the maintenance protocol only once or twice by the time 100% accuracy was achieved.

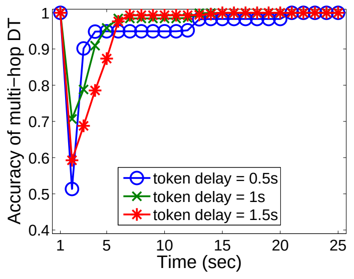
For the same parameter values and connectivity graphs as those in Figure 4, we ran simulations in which nodes joined serially controlled by token passing. The multi-hop DT at the end of every join was correct (as stated by Theorem 2). In each experiment, the time taken to construct a correct multi-hop DT for all 300 nodes was between 10 and 20 times the convergence time in Figure 4. The tradeoff is that the number of protocol messages used by serial joins was a small fraction of the number of protocol messages used by concurrent joins. A comparison of the communication costs of serial joins and concurrent joins to construct correct MDT graphs as well as those of two other graph contruction algorithms for wireless routing is shown in Figure 13 and discussed in Section 6.3.

5.4.2 *MDT routing performance*

We evaluated the performance of MDT routing by sim-

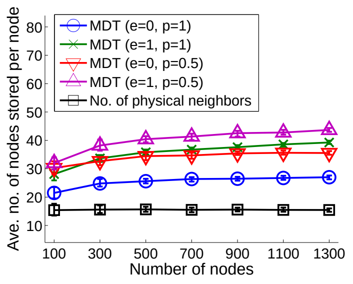


(a) Ave. message delay = 150 ms

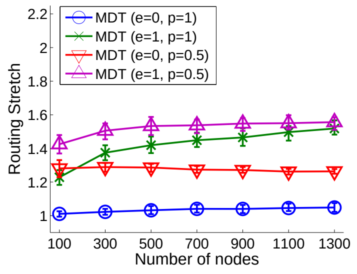


(b) Ave. message delay = 15 ms

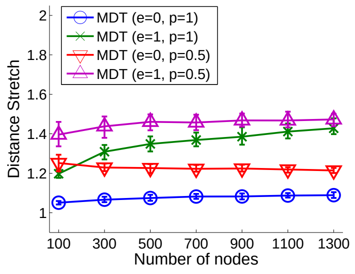
Figure 4: Accuracy vs. time for concurrent joins in 3D (location error ratio $e = 1$, connection probability $p = 0.5$)



(a) Storage cost vs. N



(b) Routing stretch vs. N



(c) Distance stretch vs. N

Figure 5: MDT routing performance for nodes in 3D

ulation experiments for nodes in 3D for the following four cases:

- unit disk model ($e = 0$, $p = 1$),
- inaccurate coordinates only ($e = 1$, $p = 1$),
- randomly placed obstacles only ($e = 0$, $p = 0.5$),
- both inaccurate coordinates and randomly placed obstacles ($e = 1$, $p = 0.5$).

The routing stretch value of a pair of nodes, s and d, in a multi-hop DT of S is defined to be the ratio of the number of hops in the MDT route to the number of hops in the shortest route (in hops) between s and d. The **routing stretch** of the multi-hop DT is defined to be the average of the routing stretch values of all source-destination pairs in S. The *distance stretch* of the multi-hop DT is defined similarly with distance replacing number of hops as metric.

In Figure 5, we present results from simulation experiments for a varying number (N) of nodes with transmission range $R = 250$ for $p = 1$, and $R = 325$ for $p = 0.5$. The 3D space size increases with N such that the average node degree (number of physical neighbors per node) is maintained at approximately 15.5. A correct multi-hop DT was first constructed at the beginning of each experiment. Routing success rate was observed to be 100% in every experiment.

In Figure 5(a), the storage cost of a node is the average number of other nodes whose coordinates have to be stored in the node.[8] From the figure we observe that the storage cost (per node) increases slowly, with the rate of increase trending to zero as N becomes large. The introduction of inaccurate coordinates ($e = 1$) and randomly placed obstacles ($p = 0.5$) requires more storage per node.

In Figures 5(b)-(c), both routing stretch and distance stretch are close to 1 for the unit disk model. Inaccurate coordinates and randomly placed obstacles increase both the routing stretch and distance stretch of MDT routing. However, we observed that inaccurate coordinates and randomly placed obstacles do not affect the guaranteed delivery property of MDT routing (Theorem 1).

Simulation methodology. In Figure 5, each data point plotted is the average value of 50 simulation runs for 50 connectivity graphs generated from the Random Graph model. For each simulation run, the 90th percentile and 10th percentile values are also plotted as bars above and below the average value. Almost all of the intervals between 90th and 10th percentile values are very small. Such small intervals between 90th and 10th percentile values are typical of all simulation results to be presented in the balance of this paper. For the sake of clarity, we will omit 90th and 10th percentile values in other figures. Note that each data point plotted will still be the average value of 50 simulation runs (with the exception of transient behaviors from churn experiments

[8] In MDT, each node is identified globally by its coordinates. Within a node, locally defined identifiers are used to represent nodes in the forwarding table to reduce storage cost.

shown in Section 5.6).

5.5 Leave and failure protocols

Join and maintenance protocols are sufficient for a system of nodes to recover from churn and their multi-hop DT to converge to 100% accuracy. It is however desirable for MDT to include leave and failure protocols designed for a single leave and failure, respectively, for the following reasons:

1) A departed node has almost all recovery information in its state to inform its neighbors how to repair their states. Such recovery information is not available to the maintenance protocol and would be lost if not provided by a leave or failure protocol when the node leaves or fails. Thus having leave and failure protocols in MDT allows the maintenance protocol, which has a higher communication cost, to run less frequently than otherwise.

2) Concurrent join, leave and failure occurrences in different parts of a large network are often independent of each other. After a leave or failure, node states can be quickly and effectively repaired by leave and failure protocols without waiting for the maintenance timeout to occur.

Leave protocol. Consider a node u that leaves gracefully. When node u's neighbors update their states, it is not sufficient for a neighbor v to delete u from C_v and N_v. This is because v may have a new neighbor z that was not a neighbor of v before u's departure and v does not know z after u's departure. However, such a node z is always a neighbor of u prior to u's departure (Lemma 10 in [17]). Therefore node u can notify neighbor v that u is leaving and provide v with the following information:

1. v's neighbor set N_v^u in $DT(N_u)$,[9] and
2. a graph $G = < V, E >$, where the set of vertices $V = N_u$, and the set of edges, $E = \{(v,z) \mid v, z$ are neighbors in $DT(N_u)$ and F_u does not contain a tuple with v and z as endpoints$\}$.

We use *vertex* to refer to a node in graph G and *route* to refer to a path in graph G connecting two vertices. Note that all vertices in G are DT nodes. Edges in G connect neighbors in the multi-hop DT of S. By the definition of G, none of these edges uses u as a node in its forwarding path.

After receiving a leave notification, v computes a route in G to every node z in its updated neighbor set. *Suppose* such a route exists in G between v and z. Node v sends to z a path-recover message along the route as follows: The path-recover message is relayed by vertices along the route. Two adjacent vertices in the route, being neighbors in the multi-hop DT of S, are connected by a physical link or a forwarding path. At every hop along the route from v to z, a tuple with v and z as endpoints is stored, thus establishing a forwarding path between v and z. The leave protocol is highly efficient for repairing node states after a leave.

[9]Note that u is not in N_u.

For some rare cases, the leave protocol may not be able to repair all node states after a leave for two reasons. First, the leaving node u may be an articulation point of the connectivity graph. Second, even if u is not an articulation point, it is possible that v and some neighbor z are disconnected in G because the forwarding paths of all routes between them in the $DT(N_u)$ graph use node u to forward messages. In this case, node v exits the leave protocol and immediately runs the maintenance protocol to repair node states. (A pseudocode specification of the MDT leave protocol is presented in the Appendix.)

Failure protocol. The failure protocol is similar to the leave protocol and almost as efficient. The key idea is that every node u prepares recovery information for its neighbors in case u fails. The recovery information includes, for each neighbor v, its neighbor set N_v^u in $DT(N_u)$ after u's departure as well as the graph G in the leave protocol. Node u selects one of its neighbors (say m) as its monitor node and sends to m the recovery information for every node in u's neighbor set. (The recovery information is updated by u whenever there is a change in N_u.) The monitor node m periodically probes u to check that u is alive. When m detects failure of u, m sends to each of u's former neighbors its recovery information prepared by u.

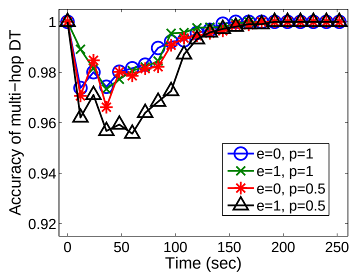
5.6 MDT performance for systems under churn

We performed a large number of experiments to evaluate the performance of MDT protocols for systems of nodes under churn. For each experiment, there are 300 nodes initially maintaining a correct multi-hop DT. The **churn rate** is defined to be the rate at which new nodes join the system, which is equal to the rate at which existing nodes leave or fail from the system. In each experiment, each departing node is randomly selected (with probability 0.5) to be a graceful leave or a failure. Nodes are randomly distributed in a $800 \times 800 \times 800$ 3D space. In each experiment, churn begins at time 0 and ends at time 60 seconds.

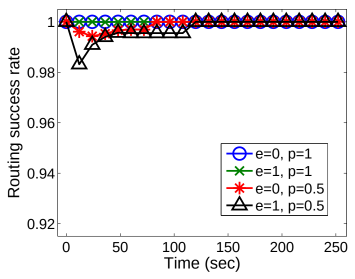
Both Figures 6 and 7 are from experiments for low-speed networks where one-hop message delays are sampled from [100 ms, 200 ms].

Figures 6(a)-(c) are for the four cases of accurate or inaccurate coordinates ($e = 0$ or 1) with or without randomly placed obstacles ($p = 0.5$ or 1). The transmission range is $R = 250$ for $p = 1$, and $R = 325$ for $p = 0.5$; the average node degree is 15.5. The maintenance timeout value is 60 seconds for all three figures. The churn rate is 100 nodes/minute in Figures 6(a)-(b) and varies in Figure 6(c). Figure 6(a) shows the accuracy of the multi-hop DT versus time. The accuracy returns to 100% quickly after churn. Figure 6(b) shows the routing success rate versus time. The success rate is close to 100% during churn and returns to 100% quickly after churn. Figure 6(c) shows the communication cost (per node per second) versus churn rate.

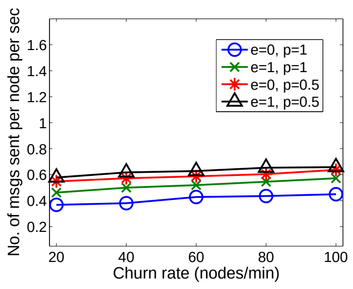
When nodes have inaccurate coordinates ($e = 1$) or there are randomly placed obstacles ($p = 0.5$), note that the accu-



(a) Churn rate = 100 nodes/min.

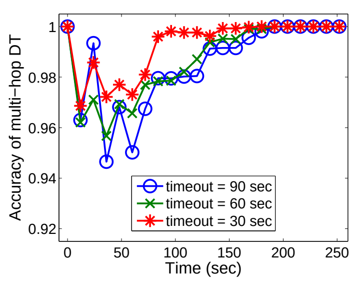


(b) Churn rate = 100 nodes/min.

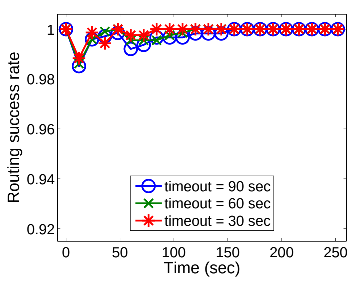


(c) Communication cost vs. churn rate

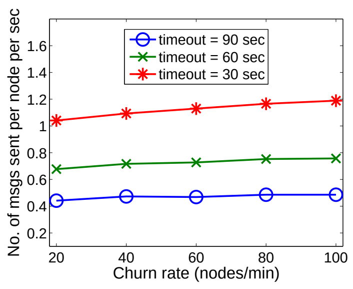
Figure 6: MDT performance for systems in 3D under churn (ave. message delay $= 150$ ms, timeout $= 60$ sec.)



(a) Churn rate = 50 nodes/min.



(b) Churn rate = 50 nodes/min.



(c) Communication cost vs. churn rate

Figure 7: MDT performance for systems in 3D under churn (ave. message delay$= 150$ ms, $e = 1$, $p = 0.5$)

racy in Figure 6(a) and the success rate in Figure 6(b) are slightly lower, and the communication cost in Figure 6(c) is slightly higher. However, the convergence times to 100% accuracy in Figure 6(a) and to 100% success rate in Figure 6(b) are almost the same for the four cases.

Figures 7(a)-(c) are for three maintenance timeout values (30, 60, and 90 seconds) for systems with both inaccurate coordinates and randomly placed obstacles ($e = 1, p = 0.5$). The churn rate is 50 nodes/minute in Figures 7(a)-(b) and varies in Figure 7(c). Figure 7(a) shows the accuracy of the multi-hop DT versus time. The accuracy returns to 100% quickly after churn. Figure 7(b) shows the routing success rate versus time. The success rate is close to 100% during churn and returns to 100% quickly after churn. Figure 7(c) shows the communication cost (per node per second) versus churn rate.

Observe from Figure 7(c) that a decrease in the timeout value to 30 seconds causes a large increase in communication cost. On the other hand, increasing the churn rate has only minor impact on communication cost. For a timeout value of 60 seconds or more, the communication cost is quite low (less than 0.7 message sent per node per second).

By Little's Law, for 300 nodes and a churn rate of 100 nodes/minute, the average lifetime of a node is 300/100 = 3 minutes, which represents a very high churn rate for most practical systems.

The above experiments were repeated for high-speed networks where one-hop message delays are sampled from [10 ms, 20 ms]. The results are shown in Figures 8(a)-(c) and Figures 9(a)-(c). The maintenance timeout value is 60 seconds in Figures 8(a)-(c). The accuracy of multi-hop DT and routing success rate in these experiments are better than those in Figures 6(a)-(b) and Figures 7(a)-(b) for low-speed networks, even though higher churn rates are used (up to 120 nodes/minute). The communication costs (per node per second) are about the same.

6. PERFORMANCE COMPARISON

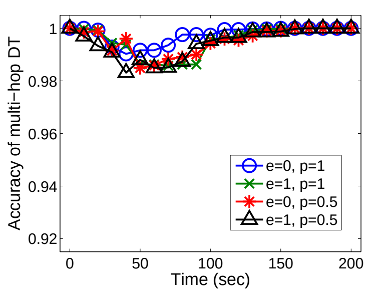
6.1 MDT compared with GDSTR and GPSR on GG, RNG, and CLDP in 2D space

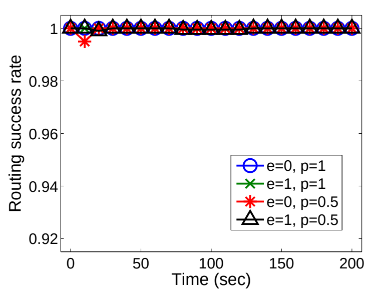
The geographic routing protocols, GPSR running on GG, RNG, and CLDP graphs [14, 15], and GDSTR [20] were designed for routing in 2D. We implemented these protocols in our simulator.[10] We compare the performance of MDT routing with these protocols for 300 nodes in a 1000×1000 2D space.

The results in Figure 10 are for nodes with inaccurate coordinates only ($0 \leq e \leq 2$, $p = 1$) and transmission range $R = 120$. The results in Figure 11 are for nodes with randomly placed obstacles only ($0.4 \leq p \leq 1$, $e = 0$) and transmission range $R = 150$.

In Figure 10(a), the routing success rates of MDT and GDSTR are both 100% for all e values (it was 100% in every

[10]Using, as our references, [15] for CLDP, GDSTR code from www.comp.nus.edu.sg/˜bleong/geographic/, and GPSR, GG, and RNG code from www.cs.ucl.ac.uk/staff/B.Karp/gpsr/.


(a) Churn rate = 120 nodes/min.


(b) Churn rate = 120 nodes/min.

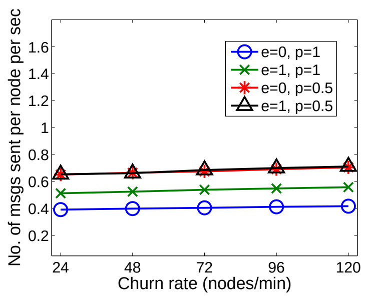
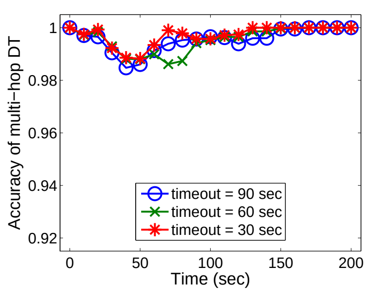
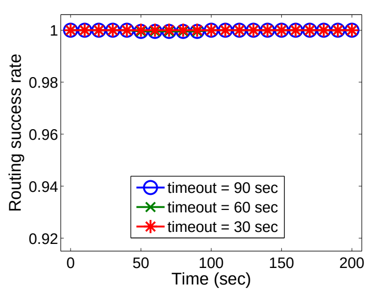

(c) Communication cost vs. churn rate

Figure 8: MDT performance for systems under churn (ave. message delay $= 15$ ms, timeout $= 60$ sec.)


(a) Churn rate = 60 nodes/min.


(b) Churn rate = 60 nodes/min.

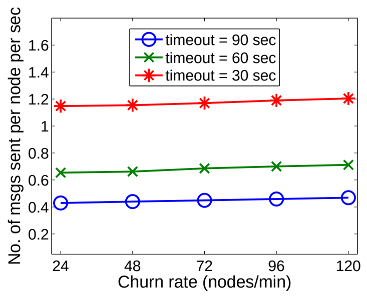

(c) Communication cost vs. churn rate

Figure 9: MDT performance for systems under churn (average message delay $= 15$ ms, $e = 1$, $p = 0.5$)

experiment). As the location error ratio (e) increases from 0, the routing success rates of GPSR running on GG, RNG, and CLDP drop off very gradually from 100%. For $e > 1$, their routing success rates drop significantly.

In Figure 11(a), the routing success rates of MDT and GDSTR are both 100% for all p values (it was 100% in every experiment.) As the connection probability p decreases from 1, the routing success rate of CLDP decreases minutely from 100% . It is 99.8% at p=0.4. The routing success rates of GG and RNG drop off very gradually from 100% as p decreases from 1. For $p < 0.7$, their routing success rates drop significantly.

In Figure 10(b), MDT has the lowest routing stretch for all e values, with GDSTR a close second, followed by GG, CLDP, and RNG. For $0 \le e \le 0.4$, the differences are small. But as e increases above 0.8, the GG, CLDP, and RNG curves increase rapidly. The MDT and GDSTR curves increase slowly as e increases from 0 to 2.

In Figure 11(b), MDT routing has the lowest routing stretch for all p values, with GDSTR a very close second, followed by GG, RNG, and CLDP. For $0.8 \le p \le 1$, the differences are small. As p decreases below 0.8, the GG, CLDP, and RNG curves increase rapidly. The MDT and GDSTR curves increase slowly as p decreases from 1 to 0.4.

To compare storage costs of the routing protocols, we count the number of nodes whose coordinates have to be stored at a node.[11] For GPSR protocols (GG, RNG, and CLDP), a node stores the coordinates of its physical neighbors only. For GDSTR, a node stores the coordinates of its physical neighbors and nodes in its convex hull and in the convex hulls of its children (if any) in the spanning tree. For MDT, a node stores the coordinates of its physical neighbors, multi-hop DT neighbors, and nodes that are endpoints of tuples in its forwarding table.

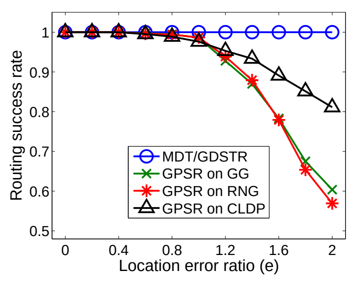
The storage costs (per node) of the protocols are shown in Figure 10(c) and 11(c). The storage cost of GPSR routing is the same as the average node degree. In Figure 10(c), the storage cost of GPSR is 12.4 nodes. As the connection probability (p) decreases from 1 to 0.4 in Figure 11(c), it decreases from 18 to 7. Note that both MDT and GDSTR require more storage costs than GPSR. The extra costs, which are not large, are incurred to achieve a routing success rate of 100%.

6.2 MDT compared with VRR and GRG in 3D space

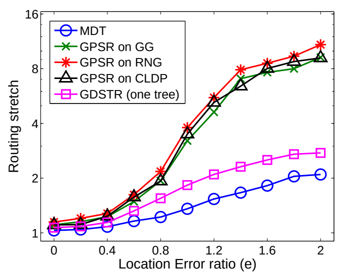
In 3D, we compare the routing performance of MDT with GRG [9], a geographic protocol designed for 3D, and the non-geographic protocol, VRR [6].

We implemented GRG in our simulator from its description in [9]. We implemented VRR for static networks (with-

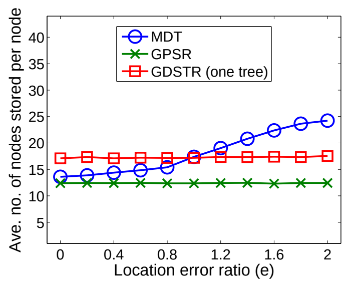
[11]Coordinates are the most important information for geographic protocols. We have ignored other storage costs that are smaller and dependent on implementation details.



(a) Routing success rate vs. e

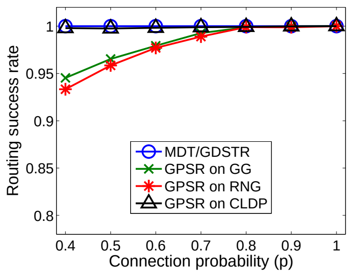


(b) Routing stretch vs. e

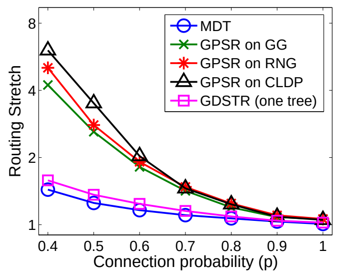


(c) Storage cost vs. e

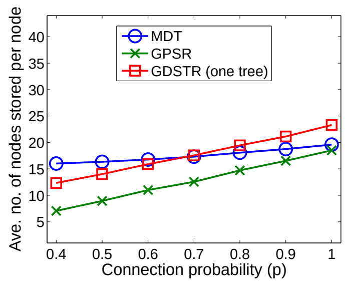
Figure 10: MDT compared with GDSTR and GPSR on GG, RNG, and CLDP in 2D for varying *e*



(a) Routing success rate vs. p



(b) Routing stretch vs. p



(c) Storage cost vs. p

Figure 11: MDT compared with GDSTR and GPSR on GG, RNG, and CLDP in 2D for varying *p*

out joins and failures).[12] Each node has 4 virtual neighbors as in [6]. Between each pair of nodes that are virtual neighbors, we used the shortest path (in hops) between them as the forwarding path (for these nodes, the routing stretch value is 1). Thus, the routing stretch curves shown in Figure 12(b) for VRR are slightly optimistic.

In these experiments, the number N of nodes is varied from 100 to 1300. As N increases, the 3D space size is increased to keep the average node degree at approximately 15.5. Connectivity graphs are generated for two cases: (i) $e = 1$ and $p = 1$ (inaccurate coordinates only) with $R = 250$, and (ii) $e = 0$ and $p = 0.5$ (randomly placed obstacles only) with $R = 325$.

In Figure 12(a), the routing success rate of GRG is well below 100%. In Figure 12(b), which is in logarithmic scale, the routing stretch of GRG is very high and increases with network size N. In Figure 12(c), the storage cost of GRG, equal to the average number of physical neighbors, is the lowest of the three protocols.

The routing performance of VRR is affected neither by inaccurate coordinates nor by randomly placed obstacles. From Figure 12(a), its routing success rate is equal to 100% (like MDT). We implemented two versions of VRR: (v1) each node additionally stores 2-hop neighbors as in [6], and (v2) nodes do not store 2-hop neighbors. In Figure 12(b), VRRv1 has a much lower routing stretch than VRRv2. However, each node in VRRv1 stores many more nodes than VRRv2 as shown in Figure 12(c).

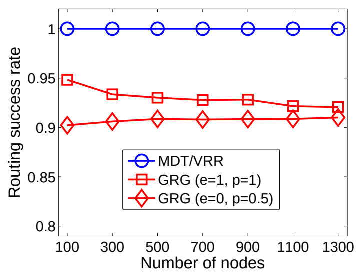
In Figure 12(a), the routing success rate of MDT is 100%. In Figure 12(b), the routing stretch of MDT is the lowest of the three protocols with the exception of one data point ($N = 100$) at which VRRv1 is slightly lower. In Figure 12(c), MDT has a larger storage cost than GRG. MDT stores about the same number of nodes as VRRv2 and a lot fewer nodes than VRRv1. However, an accurate comparison of the storage costs of MDT and VRR should consider also the sizes of (global) node identifiers in these protocols.

6.3 Communication cost comparison for graph construction

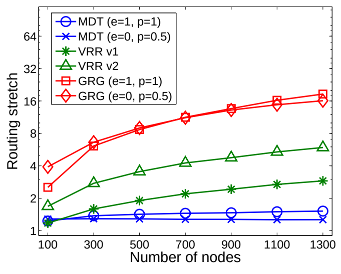
In these experiments, the number N of nodes is varied from 100 to 1300. The transmission range is $R = 200$. As N increases, the 2D space size is increased to keep the average node degree at approximately 14. Connectivity graphs are generated for nodes with inaccurate coordinates ($e = 1$) and there are randomly placed obstacles between nodes ($p = 0.5$).

In Figure 13, we compare MDT's message cost to construct a correct multi-hop DT with message costs of CLDP graph construction using serial probes [15] and GDSTR spanning tree construction [20]. The vertical axis is in logarithmic scale. The message cost of a protocol is the average number of messages *sent* per node (we did not account for message size differences among the protocols). Note that each GDSTR message is a broadcast message sent by a node to all of its physical neighbors and is counted as one mes-

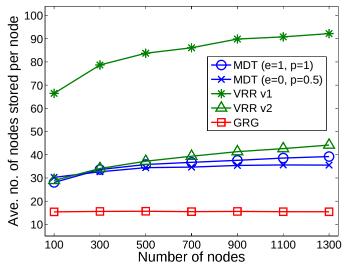
[12]With reference from www.cs.berkeley.edu/˜mccaesar/vrrcode .



(a) Routing success rate vs. N



(b) Routing stretch vs. N



(c) Storage cost vs. N

Figure 12: MDT compared with VRR and GRG in 3D

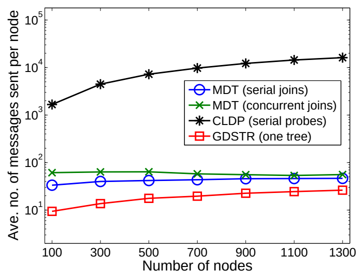


Figure 13: Initialization message cost vs. N **(**$e = 1$**,** $p = 0.5$**)**

sage sent. Messages sent by CLDP and MDT are unicast messages.

Figure 13 shows that with the average number of messages *sent* per node as metric, GDSTR has the best message cost performance, followed by MDT (serial joins), MDT (concurrent joins), and CLDP. Note the trend of each curve as N increases. The CLDP curve increases gradually with N. The GDSTR curve increases very slightly as N increases. The MDT curves are flat (they actually decrease very slightly) as N increases. (Recall that each point plotted is the average value of 50 simulation runs for 50 connectivity graphs generated from the Random Graph model.)

7. CONCLUSIONS

We have presented MDT, a novel geographic routing protocol with several major advances over previous geographic protocols. In this paper, the graph of nodes and physical links is assumed to be connected but otherwise may be arbitrary. Conceptually, the MDT routing structure is a Delaunay triangulation (DT) overlay on an *arbitrary connectivity graph*. We refer to the structure as a multi-hop DT. MDT routing achieves the *lowest routing stretch* of all routing protocols for 2D and 3D compared in this paper. This is because a multi-hop DT is rich in links (and DT is a good spanner [7]). MDT routing uses all of the physical links with additional virtual links connecting multi-hop DT neighbors.

We have proved that for a given destination location ℓ, MDT routing in a correct multi-hop DT provides *guaranteed delivery* to a node closest to ℓ. This theorem holds for nodes located in d-dimensional Euclidean spaces ($d \geq 2$) with accurate, inaccurate, or arbitrary coordinates. Experimental results show that inaccurate coordinates or missing physical links between close neighbors (e.g., due to randomly placed obstacles) impact the routing performance of MDT routing but they do not affect its guaranteed delivery property (Theorem 1).

MDT is *scalable* to a large network size N. Experimental results show that for a fixed node density, the average number of messages sent per node to construct a correct multi-hop MDT is constant (or decreases slightly) as N increases. The average storage cost per node increases slowly, with the rate of increase trending to zero as N becomes large. Lastly, the computation cost at each node is independent of N because each node only needs to compute its local DT.

We have designed and implemented join, leave, failure, maintenance, and initialization protocols for constructing and maintaining a correct multi-hop DT. The protocols use *soft states*. The join protocol is proved to be correct for serial joins. Experimental results show that a correct multi-hop DT can be constructed very quickly at system initialization using concurrent joins. Experimental results show that MDT is highly *resilient to churn*. The routing success rate of MDT is close to 100% for systems under churn. After churn, node states converge quickly to a correct multi-hop DT.

Given the above considerations, MDT routing is an attractive solution to wireless routing in d-dimensional Euclidean spaces ($d \geq 2$). Furthermore, MDT can be used for geographic routing in wireline networks as well. We believe that MDT routing can be an attractive routing solution for community/metropolitan networks [2] or infrastructure networks that support WiFi, WiMax, or cellular systems.

8. APPENDIX

Theorem 1. *Consider a correct multi-hop DT of a finite set S of nodes in a d-dimensional Euclidean space* ($d \geq 2$)*. Given a location* ℓ *in the space, the MDT routing protocol succeeds to find a node in S closest to* ℓ *in a finite number of hops.*

PROOF. We make use of the proof of Theorem 1 in [17] for a distributed DT:

1) By definition, a correct multi-hop DT of S is a correct distributed DT of S. The DT maintained by nodes in S is the same as $DT(S)$.

2) Given a correct multi-hop DT, each DT neighbor of a node u in S is either a one-hop neighbor or connected to u by a forwarding path of finite length (in hops) that exists in $\{F_v \mid v \in S\}$.

3) When a message arrives at a node, say u, that is not at the destination location, if the message is neither destined nor to be forwarded to a one-hop DT neighbor (lines 1-7),[13] node u performs greedy routing (lines 8-21). If greedy routing succeeds to find in P_u a physical neighbor v that is closer to ℓ than node u, the message is transmitted directly to v (lines 9-11); else, greedy routing is performed over the set of DT neighbors (lines 13-17).[14] From 1), the proof of Theorem 1 in [17] for a distributed DT guarantees that either node u is closest to ℓ or there exists in N_u a node v that is closer to ℓ than u. Therefore, if node u is not a closest node to ℓ, executing the greedy routing code (lines 9-17) finds a node v that is closer to ℓ than node u. From 2), sending the message from u to v is achieved in a finite number of hops if v is a multi-hop DT neighbor.

4) Any node in S (v in particular) that is closer to ℓ than u will not use greedy routing (lines 9-17) to send the message back to node u. Thus node u will not execute the greedy routing code (lines 8-21) again for this message.[15] Since every node in S executes the greedy routing code (lines 8-21) at most once and S has a finite number of nodes, together with 2) and 3), MDT routing finds a closest node in S to ℓ in a finite number of hops.

□

Theorem 2. *Let S be a set of nodes and w be a joining node that is a physical neighbor of at least one node in S. Suppose the existing multi-hop DT of S is correct, w joins using the MDT join protocol, and no other node joins, leaves, or fails. Then the MDT join protocol finishes and the updated multi-hop DT of $S \cup \{w\}$ is correct.*

PROOF. By Theorem 1, the join request of w succeeds to find a DT node (say z) closest to w, which sends back a joint reply. By Lemma 5 in [17], node z is guaranteed to be a neighbor of w in $DT(S \cup \{w\})$. A forwarding path is constructed between w and z (guaranteed by Theorem 1). Subsequently, because the multi-hop DT of S is correct, forwarding paths are constructed between w and each neighbor it sends a neighbor-set request. After receiving a request from w, each neighbor of w updates its own neighbor set to include w. They also send back replies to w. By Lemma 9 in [17], the join process finishes and N_w consists of all neighbor nodes of w in $DT(S \cup \{w\})$. Since a path has been constructed from w to every node in N_w, w and each of its neighbors in $DT(S \cup \{w\})$ can communicate with each other.[16]

By construction, two DT neighbors select only one path to use between them by refreshing only tuples stored in nodes along the selected path. Therefore, the path between each pair of neighbors in $DT(S \cup \{w\})$ is unique after the join. Each path also has a finite number of hops because (i) the path from the joining node to its closest DT node (say z) has a finite number of hops because z is found by MDT routing in a correct multi-hop DT (by Theorem 1), and (ii) the path from the joining node to each of its other DT neighbor is either a one-hop path or the concatenation of two paths, each of which has a finite number of hops. By Definition 3, the updated multi-hop DT is correct. □

Protocol pseudocode. A MDT protocol message m can be sent/forwarded by a node in three ways, which are defined in Figure 14 where $m.dest$ and $m.source$ denote the destination and source nodes in m, respectively, and $m.type$ denotes its message type. Pseudocode specifications of the MDT join and leave protocols are presented in Figures 15 and 16, respectively.

9. REFERENCES

[1] I. F. Akyildiz, D. Pompili, and T. Melodia. Underwater Acoustic Sensor Networks: Research Challenges. *Ad Hoc Networks*, 2005.

[2] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 2005.

[3] S. M. N. Alam and Z. J. Haas. Coverage and Connectivity in Three-Dimensional Networks. In *Proc. of ACM Mobicom*, 2006.

[4] P. Bose and P. Morin. Online routing in triangulations. *SIAM journal on computing*, 33(4):937–951, 2004.

[5] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. In *Proc. of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM)*, 1999.

[6] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual Ring Routing: Networking Routing Inspired by DHTs. In *Proceedings of ACM Sigcomm*, 2006.

[7] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay Graphs are Almost As Good As Complete Graphs. *Discrete & Computational Geometry*, 5, 1990.

[8] S. Durocher, D. Kirkpatrick, and L. Narayanan. On Routing with Guaranteed Delivery in Three-Dimensional Ad Hoc Wireless Networks. In *Proceedings of ICDCN*, 2008.

[9] R. Flury and R. Wattenhofer. Randomized 3D Geographic Routing. In *Proceedings of IEEE Infocom*, 2008.

[10] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon-Vector Routing: Scalable Point-to-Point Routing in Wireless Sensor Networks. In *Proc. of NSDI*, 2005.

[11] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, second edition, 2004.

[12] K. R. Gabriel and R. R. Sokal. A New Statistical Approach to Geographic Variation Analysis. *Systematic Zoology*, 1969.

[13] Line numbers refer to lines in *Get_Next* in Figure 3

[14] At this point only multi-hop neighbors need to be considered.

[15] It is however possible that this message will visit node u again with u acting as a forwarding node executing lines 4-6.

[16] Note that neighbors of w are the same in MDT as in [17, 18] even though MDT uses soft states because neighbors of w are endpoints of tuples in F_w. As long as a node remains a neighbor of w, its tuple will not be deleted from F_w.

For a node *u* and a message *m*, node *u* can send *m* out in three ways:

1. **Send**(*m*, *successor*): *m* is a message created by node *u* and it is to be sent to a destination node one or more hops away; *successor* is an input parameter.
2. **Forward**(*m*): node *u* forwards the message *m* for another node using *u*'s forwarding table.
3. **Transmit** *m* **to** *v*: node *u* transmits the message *m* directly to a physical neighbor *v*.

Send(*m, successor*): node *u* sends its message *m*

```
1.  if there exists t in $F_u$ | t.succ = m.dest and
            (m.type = JOIN_ REPLY or t.succ = t.dest) then
2.      Transmit m to m.dest    // use a shortcut
3.  else
4.      t ← <-, -, successor, m.dest>
5.      $F_u \leftarrow F_u \cup \{t\}$
6.      Transmit m to successor
7.  end if
```

Forward(*m*): node *u* forwards a message *m*

```
1.  if there exists t in $F_u$ | t.succ = m.dest and
            (m.type = JOIN_REPLY or t.succ = t.dest) then
2.      Transmit m to m.dest    // use a shortcut
3.  else
4.      t ← tuple in $F_u$ such that t.source = m.source and t.dest = m.dest
5.      Transmit m to t.succ
6.  end if
```

Figure 14: Three ways to send a message

[13] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proc. MobiHoc*, 2001.

[14] B. Karp and H. Kung. Greedy Perimeter Stateless Routing for Wireless Networks. In *Proceedings of ACM Mobicom*, 2000.

[15] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic Routing Made Practical. In *Proceedings of USENIX NSDI*, 2005.

[16] F. Kuhn, R. Wattenhofer, and A. Zollinger. Ad-Hoc Networks Beyond Unit Disk Graphs . In *Proc. of ACM DIALM-POMC*, 2003.

[17] D.-Y. Lee and S. S. Lam. Protocol design for dynamic Delaunay triangulation. Technical Report TR-06-48, The Univ. of Texas at Austin, Dept. of Computer Sciences, October 2006.

[18] D.-Y. Lee and S. S. Lam. Protocol Design for Dynamic Delaunay Triangulation. In *Proceedings of IEEE ICDCS*, 2007.

[19] D.-Y. Lee and S. S. Lam. Efficient and Accurate Protocols for Distributed Delaunay Triangulation under Churn. In *Proceedings of IEEE ICNP*, November 2008.

[20] B. Leong, B. Liskov, and R. Morris. Geographic Routing without Planarization. In *Proceedings of USENIX NSDI*, 2006.

[21] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang. Localized Delaunay Triangulation Application in Ad Hoc Wireless Networks. *IEEE Tran. on Paral. Distr. Syst.*, 2003.

[22] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small State and Small Stretch Routing Protocol for Large Wireless Sensor Networks. In *Proceedings of USENIX NSDI*, 2007.

[23] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing without Location Information. In *Proceedings of ACM Mobicom*, 2003.

[24] K. Seada, A. Helmy, and R. Govindan. On the Effect of Localization Errors on Geographic Face Routing in Sensor Networks. In *Proceedings of IPSN*, 2004.

[25] M. Thorup and U. Zwick. Compact Routing Schemes. In *Proceedings of ACM SPAA*, 2001.

[26] G. Toussaint. The Relative Neighborhood Graph of a Finite Planar Set. *Pattern Recognition*, 1980.

[27] M.-J. Tsai, H.-Y. Yang, and W.-Q. Huang. Axis-Based Virtual Coordinate Assignment Protocol and Delivery-Guaranteed Routing Protocol in Wireless Sensor Networks. In *Proc. of INFOCOM*, 2007.

[28] G. Xing, C. Lu, R. Pless, and Q. Huang. On Greedy Geographic Routing Algorithms in Sensing-covered Networks. In *Proceedings of ACM Mobihoc*, 2004.

Notation and Definitions:

When a node receives a message, it stores the message in the local data structure *m*. We define five message formats for the MDT join protocol:

1. if *m.type* = **JOIN_REQ**, *m* = <*m.type*, *m.source*, *m.relay*>
2. if *m.type* = **JOIN_REPLY**, *m* = <*m.dest*, *m.type*, *m.source*>
3. if *m.type* = **NB_SET_REQ**, *m* = <*m.dest*, *m.type*, *m.source*, *m.relay*>
4. if *m.type* = **NB_SET_REPLY**, *m* = <*m.dest*, *m.type*, *m.source*, *m.set*>

where *m.dest* is the destination node, *m.source* the source node, *m.relay* is the relay node, and *m.set* is a set of nodes.

Join(*v*) of node *u*

```
// u is the joining node. If there is a DT node that is a physical neighbor of u,
// v ← the node ID; otherwise, v ←null
1. if $v \neq null$ then
2.   $C_u \leftarrow \{u\}, N_u \leftarrow \varnothing$
3.   $m \leftarrow$ <JOIN_REQ, u, null>
4.   Transmit m to v
5. else
6.   $C_u \leftarrow \{u\}, N_u \leftarrow \varnothing$
7. end if
```

On *u*'s receiving *m* = <JOIN_REQ, *w*, *r*> from *v*

```
1.   $e \leftarrow$ Get_Next(w, r)
         // m.relay=r may be changed when Get_Next returns
2.   if e = null then      // u is the DT node closest to w
3.     Send(<w, JOIN_REPLY, u>, v)   // successor = v
4.   else
5.     Transmit m to e
6.     $F_u \leftarrow F_u \cup \{<w, v, e, e>\}$
7.   end if
```

On *u*'s receiving *m* = <*f*, JOIN_REPLY, *w*> from *v*

```
1. if u = f then $C_u \leftarrow \{u, w\}$
2.   Send(<w, NB_SET_REQ, u, null>, v) )   // successor = v
   // inform all physical neighbors that w has joined
   // each physical neighbor will replace its tuple <-, -, u, -> with <-, -, u, u>
3. else
4.   Replace the tuple <v, v, *, f> in $F_u$ with <w, v, *, f>
                                   // * denotes any node stored
5.   Forward(m)
6. end if
```

On *u*'s receiving *m* = <*f*, NB_SET_REQ, *w*, *r*> from *v*

```
1.   if u = f then   // Case 1: u is the destination node
2.     if $w \notin C_u$ then
3.       $C_u \leftarrow C_u \cup \{w\}$
4.       $N_u \leftarrow$ neighbor nodes of u in $DT(C_u)$
5.     end if
6.     $N_w^u \leftarrow \{e \mid e$, u and w are in the same simplex in $DT(C_u)\}$
7.     Send(<w, NB_SET_REPLY, u, $N_w^u$>, v)   // successor = v
8.   else if r = null then
         // Case 2: u is between the relay and destination
9.     $t \leftarrow$ tuple in $F_u$ such that t.dest = f
10.    $F_u \leftarrow F_u \cup \{<w, v, t.succ, f>\}$
11.    Forward(m)
12.  else if u = r then   // Case 3: u is the relay node
13.    $t \leftarrow$ tuple in $F_u$ such that t.dest = f
14.    $F_u \leftarrow F_u \cup \{<w, v, t.succ, f>\}$
15.    Forward(<f, NB_SET_REQ, w, null>)
16.  else if $r \neq null$ then
         //Case 4: u is between the source and relay
17.    $t \leftarrow$ tuple in $F_u$ such that t.dest = r
18.    $F_u \leftarrow F_u \cup \{<w, v, t.succ, f>\}$
19.    Forward(m)
20.  end if
```

On *u*'s receiving *m* = <*w*, NB_SET_REPLY, *f*, N_w^f> from *v*

```
1. if u = w then
2.   $C_u \leftarrow C_u \cup N_w^f$
3.   Update_Neighbors($C_u$, $N_u$, v, f)   // successor = v, relay = f
4. else
5.   Forward(m)
6. end if
```

Update_Neighbors(C_u, N_u, *successor*, *relay*) of node *u*

```
1.   $N_u^{old} \leftarrow N_u$
2.   $N_u \leftarrow$ neighbor nodes of u in $DT(C_u)$
3.   $N_u^{new} \leftarrow N_u - N_u^{old}$   // if $N_u^{new} = \varnothing$ and there is no NB_SET_REQ
                                   // waiting for a reply, terminate join protocol
4.   for all $v \in N_u^{new}$ do
5.     Send(<v, NB_SET_REQ, u, relay>, successor)
6.   end for
```

Figure 15: MDT join protocol

Notation and Definitions:

We define two message formats for the MDT leave protocol:

1. if *m.type* = **LEAVE**, *m* = <*m.dest*, *m.type*, *m.source*, *m.set*, *m.graph*>
2. if *m.type* = **PATH_RECOVER**, *m* = <*m.dest*, *m.type*, *m.source*, *m.relay*, *m.route*>

where *m.dest* is the destination node, *m.source* is the sourc node, *m.relay* is the relay node, *m.set* is a neighbor set, *m.graph* is a graph, and *m.route* is a sequence of nodes.

The function Generate_Graph($DT(N_u)$) is used to generate the undirected graph G = <V, E>, where the set of vertices $V = N_u$, and the set of edges $E = \{(v,w) \mid v, w$ are neighbors in DT(N_u) and F_u does not contain a tuple with v and w as endpoints$\}$. We use *vertex* to refer to a node in graph G and *route* to refer to a path in graph G connecting two vertices.

Leave( ) of node *u*

```
    // node u is the leaving node
1. $G \leftarrow$ Generate_Graph($DT(N_u)$)      // note that $u \notin N_u$
2. for all $v \in N_u$ do
3.     $N_v^u \leftarrow \{w \mid w$ is a neighbor of $v$ in $DT(N_u)\}$
4.     Forward(<$v$, LEAVE, $u$, $N_v^u$, $G$>)
5. end for
```

On *u*'s receiving *m* = <*f*, LEAVE, *w*, N_f^w, *G*> from *v*

```
1.  if $u = f$ then
2.      $C_u \leftarrow (C_u \cup N_u^w) - \{w\}$
3.      $N_u \leftarrow$ neighbor nodes of $u$ in $DT(C_u)$
4.      for all $z \in N_u$ do
5.          if $z \notin P_u$ and $(u, z)$ is not in $G$ then
6.              if there is no route from $u$ to $z$ in $G$ then
7.                  exit      // call maintenance protocol
8.              else
9.                  $R \leftarrow$ shortest route in $G$ from $u$ to $z$
10.                 next $\leftarrow$ the vertex following $u$ on $R$ to $z$
11.                 successor $\leftarrow$ the physical neighbor of $u$ along $R$ to $z$
12.                 Send(<$z$, PATH_RECOVER, $u$, next, $R$ >, successor)
13.                 $F_u \leftarrow F_u \cup \{$<-, -, successor, $z$>$\}$
14.             end if
15.         end if
16.     end for
17. else
18.     Forward($m$)
19. end if
```

On *u*'s receiving <*z*, PATH_RECOVER, *w*, *r*, *R* > from *v*

```
1.  if $u = z$ then
2.      $F_u \leftarrow F_u \cup \{$<-, -, $v$, $w$>$\}$
3.  else if $u = r$ then       // u is a vertex on the route R to z
4.      next $\leftarrow$ the vertex following $u$ on $R$ to $z$
5.      $t \leftarrow$ tuple in $F_u$ such that t.dest = next
6.      $F_u \leftarrow F_u \cup \{$<$w$, $v$, t.succ, $z$>$\}$
7.      Transmit <$z$, PATH_RECOVER, $w$, next, $R$ > to t.succ
8.  else    // $u \neq r$, u is on physical path between two vertices on R
9.      $t \leftarrow$ tuple in $F_u$ such that t.dest = $r$
10.     $F_u \leftarrow F_u \cup \{$<$w$, $v$, t.succ, $z$>$\}$
11.     Transmit <$z$, PATH_RECOVER, $w$, $r$, $R$ > to t.succ
12. end if
```

Figure 16: MDT leave protocol