Geographic Routing in d-dimensional Spaces with Guaranteed Delivery and Low Stretch

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

TR-10-03 January 19, 2010
Revised, October 29, 2010

ABSTRACT

Almost all geographic routing protocols have been designed for 2D. We present a novel geographic routing protocol, named MDT, for 2D, 3D, and higher dimensions with these properties: (i) guaranteed delivery for any connected graph of nodes and physical links, and (ii) low routing stretch from efficient forwarding of packets out of local minima. The guaranteed delivery property holds for node locations specified by accurate, inaccurate, or arbitrary coordinates. The MDT protocol suite includes a packet forwarding protocol together with protocols for nodes to construct and maintain a distributed MDT graph for routing. We present the performance of MDT protocols in 3D and 4D as well as performance comparisons of MDT routing versus representative geographic routing protocols for nodes in 2D and 3D. Experimental results show that MDT provides the lowest routing stretch in the comparisons. Furthermore, MDT protocols are specially designed to handle churn, i.e., dynamic topology changes due to addition and deletion of nodes and links. Experimental results show that MDT's routing success rate is close to 100% during churn and node states converge quickly to a correct MDT graph after churn.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols—*Routing Protocols*

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Geographic Routing, Delaunay Triangulation

1. INTRODUCTION

Geographic routing (also known as location-based or geometric routing) is attractive because the routing state needed for greedy forwarding at each node is independent of network size. Almost all geographic routing protocols have been designed for nodes in 2D. In reality, many wireless applications run on nodes located in 3D [20, 1, 6, 7]. Furthermore, node location information may be highly inaccurate or simply unavailable.

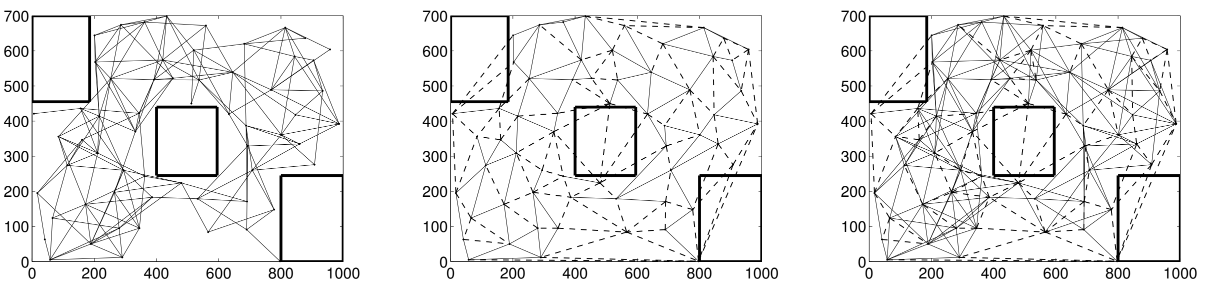
Consider a network represented by a connected graph of nodes and physical links (to be referred to as the *connectivity graph*). Greedy forwarding of a packet may be stuck at a *local minimum*, i.e., the packet is at a node closer to the packet's destination than any of the node's directly-connected neighbors. Geographic routing protocols differ mainly in their recovery methods designed to move packets out of local minima. For general connectivity graphs in 3D, face routing methods designed for 2D [3, 11, 12] are not applicable. Furthermore, Durocher et al. [6] proved that there is no "local" routing protocol that provides guaranteed delivery, even under the strong assumptions of a "unit ball graph" and accurate location information. Thus, designing a geographic routing protocol that provides guaranteed delivery in 3D is a challenging problem.

We present in this paper a novel geographic routing protocol, MDT, that provides guaranteed delivery for a network of nodes in a d-dimensional space, for $d \geq 2$. (Only Euclidean spaces are considered in this paper.) The guaranteed delivery property is proved for node locations specified by arbitrary coordinates; thus the property also holds for node locations specified by inaccurate coordinates or accurate coordinates. We show experimentally that MDT routing provides a routing (distance) stretch close to 1 for nodes in 2D and 3D when coordinates specifying node locations are accurate.[1] When coordinates specifying node locations are highly inaccurate, we show that MDT routing provides a low routing (distance) stretch relative to other geographic routing protocols. Nodes may also be arbitrarily located in a virtual space with packets routed by MDT using the coordinates of nodes in the virtual space (instead of their coordinates in physical space). In this case, MDT routing still provides guaranteed delivery but the distance stretch in physical space may be high.

Geographic routing in a virtual space is useful for networks without location information or networks in which the routing cost between two directly-connected neighbors is neither a constant nor proportional to the physical distance between them (such as, ETT [5]). For example, a 4D virtual space can be used for geographic routing of nodes physically located in a 3D space. The extra dimension makes it possible to assign nodes to locations in the virtual space such that the Euclidean distance between each pair of nodes in the virtual space is a good estimate of the routing cost between them. The design of a positioning system to embed routing costs in a virtual space is a challenging problem for wireless networks without any-to-any routing support and beyond the scope of this paper. The problem is solved in a companion paper [21] where we show how to (i) make use of MDT protocols to embed routing

Technical report revised March 18, 2011; an abbreviated version to appear in *Proceedings ACM SIGMETRICS*, San Jose, CA, June 2011

[1]Routing and distance stretch are defined later.



(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

costs in virtual spaces (such as 4D), and (ii) extend MDT routing to optimize end-to-end path costs for any additive routing metric.

MDT was designed to leverage the guaranteed delivery property of Delaunay triangulation (DT) graphs. For nodes in 2D, Bose and Morin proved that greedy routing in a DT always finds a given destination node [2]. Lee and Lam [13, 14] generalized their result and proved that in a d-dimensional Euclidean space ($d \geq 2$), given a destination location ℓ, greedy routing in a DT always finds a node that is closest to ℓ.

Figure 1(a) shows a 2D space with three large obstacles and an arbitrary connectivity graph. Figure 1(b) shows the DT graph [8] of the nodes in Figure 1(a). In the DT graph, the dashed lines denote DT edges between nodes that are not connected by physical links. The MDT graph of the connectivity graph in Figure 1(a) is illustrated in Figure 1(c). By definition, the MDT graph includes every physical link in the connectivity graph and every edge in the DT graph. In MDT routing, when a packet is stuck at a local minimum of the connectivity graph. the packet is next forwarded, via a virtual link, to the DT neighbor that is closest to the destination. In short, the recovery method of MDT is to forward greedily in the DT graph which is guaranteed to succeed.

In this paper, we present MDT protocols for a set of nodes to construct and maintain a correct *multi-hop DT* (formal definition in Section 2). In a multi-hop DT, two nodes that are neighbors in the DT graph communicate directly if there is a physical link between them; otherwise, they communicate via a virtual link, i.e., a path provided by soft-state forwarding tables in nodes along the path.

MDT protocols are also designed specially for networks where node churn and link churn are nontrivial concerns. For example, in a wireless community network, nodes join and leave whenever computers in the community are powered on and off. Furthermore, the quality of wireless links may vary widely over time for many reasons (e.g., fading effects, external interference, and weather conditions). Link quality fluctuations cause dynamic addition and deletion of physical links in the connectivity graph used for MDT routing.

The MDT protocol suite consists of protocols for forwarding, join, leave, failure, maintenance, and system initialization. The MDT join protocol was 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. Experimental results show that MDT constructs a correct multi-hop DT very quickly using concurrent joins. 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 is communication efficient because MDT does not use flooding to discover multi-hop 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 idea of using virtual links in MDT is conceptually simple. It was, however, a major challenge to design protocols to *correctly* construct and repair forwarding paths between multi-hop DT neighbors without the use of flooding. Lastly, since MDT routing is designed to run correctly in any connected graph of nodes and physical links, it is possible to use MDT for geographic routing in wireline networks.

1.1 Related work

There were several prior proposals to apply DT to geographic routing. None of them addressed the underlying technical issue that the DT graph of a wireless network is, in general, not a subgraph of its connectivity graph. In [23], requirements are imposed on the placement of nodes and links in 2D such the DT graph is a subgraph of the connectivity graph. In other approaches, the restricted DT graph [10] and the k-localized DT [18] are both approximations of the DT graph. These graphs were shown to be good spanners with constant stretch factors. However, being DT approximations, they do not provide guaranteed delivery. Furthermore, they were designed for nodes in 2D with connectivity graphs restricted to unit disk graphs. (A *unit disk graph* requires that a physical link exists between two nodes if and only if the distance between them is within a given radio transmission range.)

Many geographic routing protocols have been designed for nodes in 2D based upon greedy forwarding. Two of the earliest protocols, GFG [3] and GPSR [11], use face routing to move packets out of local minima. These protocols provide guaranteed delivery for a planar graph. If the connectivity graph is not planar, a planarization algorithm (such as GG [9] or RNG [22]) is used to construct a connected planar subgraph. Successful construction requires that the original connectivity graph is a unit disk graph and node location information is accurate. Both assumptions are unrealistic.

Kim et al. [12] proposed CLDP which, given any connectivity 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. CLDP was designed to provide guaranteed delivery for nodes in 2D under the assumption that there are no degenerate link crossings caused by exactly colinear links [12].

Leong et al. proposed GDSTR [16] which provides guaranteed delivery for any connectivity graph. Initially, nodes exchange messages to compute and store a distributed spanning tree. Each node also computes and stores a convex hull of the locations of all of its descendants in the subtree rooted at the node; the resulting tree is called a *hull tree*. Subsequently, a packet is routed greedily until it is stuck at a local minimum. For recovery, the packet is routed upwards in the spanning tree until it reaches a point where greedy routing can again make progress.

GHG [19] and GRG [7] are geographic protocols designed for 3D. GHG assumes a unit-ball graph and accurate location information, which are unrealistic assumptions. GRG uses random recovery which is inefficient and does not provide guaranteed delivery. Aside from MDT, there is one other geographic routing protocol that provides guaranteed delivery for general connectivity graphs in 3D, namely, GDSTR-3D [24]. For recovery, GDSTR-3D uses two distributed hull trees while MDT uses a distributed DT graph. GDSTR-3D, designed for sensor networks, assumes a static network topology; the protocol has no provision for any dynamic topology change.

1.2 Outline

The balance of this paper is organized as follows. In Section 2, we present concepts, definitions, and model assumptions. In Section 3, we present the MDT forwarding protocol. In Section 4, we present join, maintenance, and initialization protocols. In Section 5, we present an experimental performance evaluation of MDT in 3D and 4D. We also present experimental results to demonstrate MDT's resilience to node churn and link churn. In Section 6, we present performance comparisons of MDT with geographic routing protocols designed for 2D and 3D. We conclude in Section 7.

2. CONCEPTS AND DEFINITIONS

A triangulation of a set S of nodes (points) in 2D is a subdivision of the convex hull of nodes in S into non-overlapping triangles such that the vertices of each triangle are nodes in S. A DT in 2D is a triangulation such that the circumcircle of each triangle does not contain any other node inside [8]. The definition of DT can be generalized to a higher dimensional space using simplexes and circum-hyperspheres. In each case, the DT of S is a graph to be denoted by $DT(S)$.

Consider a set S of nodes in a d-dimensional space, for $d \geq 2$. Each node in S is identified by its location specified by coordinates. There is at most one node at each location. When we say node u *knows* node v, node u knows node v's coordinates. A node's coordinates may be accurate, inaccurate, or arbitrary (that is, its known location may differ from its actual location). In Section 2.1, we present the definition of a *distributed* DT and a key result from Lee and Lam [14, 15] that we need later.

2.1 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 $DT(S)$.

To construct a correct distributed DT, 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* [14, 15] for a distributed DT to be *correct* is that for all $u \in S$, C_u includes all neighbor nodes of u in $DT(S)$.

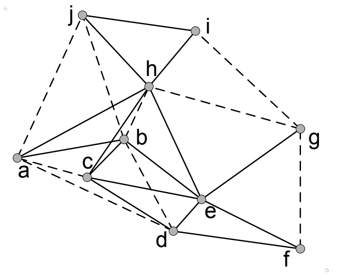


Figure 2: MDT graph of 10 nodes

2.2 Model assumptions

Two nodes directly connected by a physical link are said to be *physical neighbors*. Each link is bidirectional. In our protocol descriptions, each link is assumed to provide reliable message delivery.[2] The graph of nodes and physical links may be arbitrary so long as it is a connected graph. We provide protocols to handle dynamic topology changes. In particular, new nodes may join and existing nodes may leave or fail.[3] Furthermore, new physical links may be added and existing physical links that have become error-prone are deleted.

2.3 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 neighbors. (We will use the term "neighbor" to refer to a DT neighbor.) In a multi-hop DT, the neighbor of a node may not be a physical neighbor; see, for example, nodes i and g in Figure 2.

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 in F_u 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 fact, 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 MDT graph in Figure 2. 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.

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 messages) between its endpoints. A tuple that is not refreshed will be deleted when its timeout occurs.

Definition 3. A multi-hop DT of S, $\{< u, N_u, F_u > | u \in S\}$, is

[2]Only links that are reliable and have an acceptable error rate are included in the connectivity graph.

[3]When a node fails, it becomes silent.

Table 1: MDT forwarding protocol at node u

	CONDITION	ACTION
1.	$u = m.dest$	no need to forward (node u is at destination location)
2.	there exists a node v in P_u and $v = m.dest$	transmit to v (node v is at destination location)
3.	$m.relay \neq null$ and $m.relay \neq u$	find tuple t in F_u with $t.dest = m.relay$, transmit to $t.succ$
4.	there exists a node v in $P_u \cup \{u\}$ closest to $m.dest$, $v \neq u$	transmit to node v (greedy step 1)
5.	there exists a node v in $N_u \cup \{u\}$ closest to $m.dest$, $v \neq u$	find tuple t in F_u with $t.dest = v$, transmit to $t.succ$ (greedy step 2)
6.	conditions 1-5 are all false	no need to forward (node u is closest to destination location)

correct if and only if the following conditions hold: i) the distributed DT of S, $\{< u, N_u > | u \in S\}$, 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.

For a dynamic network in which nodes and physical links may be added and deleted, we 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.

Terminology. For a node u, a physical neighbor v that has just booted up is represented in F_u by the tuple $< -, -, -, v >$. A physical neighbor v that has sent a join request and received a join reply from a DT node is said to be a *physical neighbor attached to the DT*. It is represented in F_u by $< -, -, v, v >$. We use P_u to denote u's set of physical neighbors attached to the DT. A node in P_u will become a DT node when it finishes executing the join protocol.

3. MDT FORWARDING PROTOCOL

The key idea of MDT forwarding at a node, say u, is conceptually simple: For a packet with destination d, if u is not a local minimum, the packet is forwarded to a physical neighbor closest to d; else, the packet is forwarded, via a virtual link, to a multi-hop DT neighbor closest to d.

For a more detailed specification, consider a node u that has received a data message m to forward. Node u stores it 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. Note that if $m.relay \neq null$, message m is traversing a virtual link.

The MDT forwarding protocol at a node, say u, is specified by the conditions and actions in Table 1. To forward message m to a node closest to location $m.dest$, the conditions in Table 1 are checked sequentially. The first condition found to be true determines the forwarding action. In particular, line 3 is for handling messages traversing a virtual link. Line 4 is greedy forwarding to physical neighbors. Line 5 is greedy forwarding to multi-hop DT neighbors.

The following theorem, which states that MDT forwarding in a correct multi-hop DT provides guaranteed delivery, is proved in the 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 forwarding protocol succeeds to find a node in S closest to ℓ in a finite number of hops.*

4. MDT PROTOCOL SUITE

In addition to the forwarding protocol, MDT includes *join*, *maintenance*, *leave*, *failure*, and *initialization* protocols. The join protocol is designed to have the following correctness property: Given a system of nodes maintaining a correct multi-hop DT, after a new node has finished joining the system, the resulting multi-hop DT is correct. This property ensures that a correct multi-hop DT can be constructed for any system of nodes by starting with one node, say u with $F_u = \emptyset$ initially, which is a correct multi-hop DT by definition, and letting the other nodes join the existing multi-hop DT serially.

Two nodes are said to join a system *concurrently* if their join protocol executions overlap in time. When two nodes join concurrently, the joins are *independent* if the sets of nodes whose states are changed by the join protocol executions do not overlap. For a large network, two nodes joining different parts of the network are likely to be independent. If nodes join a correct multi-hop DT concurrently and independently using the MDT join protocol, the resulting multi-hop DT is also guaranteed to be correct.

The maintenance protocol is designed to repair errors in node states after concurrent joins that are dependent, after nodes leave or fail, after the addition of physical links, and after the deletion of existing physical links (due to, for example, degraded link quality). Experimental results show that join and maintenance protocols are sufficient for a system of nodes to recover from dynamic topology changes and their multi-hop DT to converge to 100% accuracy.

MDT includes leave and failure protocols designed for a single leave and failure, respectively, for two reasons: (i) 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 before the node leaves or fails. (For failure recovery, each node u pre-stores the recovery information in a selected neighbor which serves as u's monitor node.) Thus having leave and failure protocols allows the maintenance protocol, which has a higher communication cost, to run less frequently than otherwise. (ii) 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 protocol to run. The leave and failure protocols are presented in the Appendix.

For a multi-hop DT, in addition to constructing and maintaining a distributed DT, join and maintenance protocols insert tuples into forwarding tables and update some existing tuples to *correctly* construct paths between multi-hop neighbors. Leave, failure, and maintenance protocols construct a new path between two multi-hop neighbors whenever the previous path between them has been broken due to a node leave/failure or a link deletion.

4.1 Join protocol

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 to join the existing DT.[4] In the MDT join protocol, a node uses the basic search technique of Lee and Lam [13, 14] to find its DT neighbors. First, greedy forwarding of w's join request finds w's closest DT neighbor. Subsequently, w sends a neighbor-set request to every new neighbor it has found; each new neighbor replies with a set of w's neighbors in its local view. The search terminates when node w finds no more new neighbor in the replies. The MDT join protocol also constructs a forwarding path between w and every one of its multi-hop DT neighbors. A more detailed protocol description follows.

Finding the closest node and path construction. Node w joins by sending a join request to node v with its own location as the destination location. MDT forwarding is used to forward the join request to a DT node z that is closest to w (success is guaranteed by Theorem 1). 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 existing 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 the 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 the MDT join, maintenance, leave, and failure protocols 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 the DT.

Finding DT neighbors. Node w, after receiving the join reply from node z, sends a neighbor-set request to z for neighbor information. At this time, C_z, the set of nodes known to z includes both w and z. Node z computes $DT(C_z)$, finds nodes that are neighbors of w in $DT(C_z)$, and sends them to w in a neighbor-set reply message.

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

When w receives the neighbor-set reply from z, w adds neighbors in the reply (if any) to its candidate set, C_w, and updates its neighbor set, N_w, from computing $DT(C_w)$. If w finds new neighbors in N_w, w sends neighbor-set requests to them for more neighbor information. 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.

Nodes in C_u, the set of nodes known to a node u, are maintained as hard states in distributed DT protocols [13, 14]. In MDT protocols, nodes in C_u are maintained as soft states. More specifically, tuples in F_u are maintained as soft states. By definition, $C_u = \{u\} \cup \{v \mid v = t.dest,\ t \in F_u\}$. A new node in C_u is deleted if it does not become the destination of a tuple in F_u within a timeout period. Also, whenever a tuple t is deleted from F_u, its endpoints are deleted from C_u.

Path construction to multi-hop DT neighbors. The MDT join protocol also constructs a forwarding path between the joining node w and each of its multi-hop neighbors. Whenever w learns a new node y from the join reply or a neighbor-set reply sent by some node, say x, node w sends a neighbor-set request to x, with x as the *relay* and y as the destination (that is, in neighbor-set request m, $m.relay = x$ and $m.dest = y$.) 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 (given 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.

Example. Let node a in Figure 2 be a joining node. Suppose a has found b, c, and d to be DT neighbors and it has just learned from b that j is a new neighbor. Node a sends a neighbor-set request to j with b indicated in the message as the relay. Because the existing multi-hop DT (of 9 nodes) is correct, a unique forwarding path exists between node b and node j, which is $b-e-h-j$. After receiving the message, b forwards it to e on the $b-e-h-j$ path. At b and every node along the way to j, a tuple with endpoints a and j is stored in the node's forwarding table. 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 attached to the DT (see Figure 2). 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 no new neighbor other than b, c, and d. Without any more new neighbor to query, a's join protocol execution terminates and it becomes a DT node.

A pseudocode specification of the MDT join protocol and a proof of Theorem 2 are presented in the Appendix.

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.*

4.2 Maintenance protocol

The MDT maintenance protocol for repairing node states is designed for systems with frequent addition and deletion of nodes and physical links. For a distributed DT to be correct, each node must know all of its neighbors in the global DT. Towards this goal, each node (say u) runs the maintenance protocol by first querying a subset of its neighbors, one for each simplex including u in $DT(C_u)$. More specifically, node u selects the smallest subset V of neighbors such that every simplex including u in $DT(C_u)$ includes one node in V. Node u then sends a neighbor-set request to each node in V. A node z that has received the neighbor-set request adds u to C_z and computes $DT(C_z)$. Node z then sends a neighbor-set reply containing neighbors of u in $DT(C_z)$ to u.

Node u adds new nodes found in each neighbor-set reply to C_u; it then computes $DT(C_u)$ to get N_u. If u finds a new neighbor, say x, in N_u, node u sends a neighbor-set request to x if x satisfies the following condition:

C1. The simplex in $DT(C_u)$ that includes both u and neighbor x does not include any node to which u has 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 announce u's presence and do not require 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, node v 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 each 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 [13]. (See results from our system initialization experiments in Section 5.3 and churn experiments in Section 5.6.)

4.3 Initialization protocols

Serial joins by token passing. Starting from one node, 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.

Concurrent joins by token broadcast. Starting from one node, 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. When a node has a token, it runs the join protocol once (except the selected node) and then the maintenance protocol repeatedly, 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. 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.

5. PERFORMANCE EVALUATION

5.1 Methodology

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. We will not evaluate metrics that depend on congestion, e.g., end-to-end throughput and latency. Hence, queueing delays at a node are not simulated. Instead, message delivery times from one node to the next are sampled from a uniform distribution over a specified time interval. Time-varying wireless link characteristics and interference problems are modeled by allowing physical links to be added and deleted dynamically.

Creating general connectivity graphs. To create general connectivity graphs for simulation experiments, a physical space in 3D (2D) is first specified. ***Obstacles*** are then placed in the physical space. The number, location, shape, and size of the obstacles are constrained by the requirement that the unoccupied physical space is not disconnected by the obstacles. (Any real network environment can be modeled accurately if computational cost is not a limiting factor.) ***Nodes*** are then placed randomly in the unoccupied physical space. Let R denote the radio transmission range. ***Physical links*** are then placed using the following algorithm: For each pair of nodes, *if* the distance between them is larger than R or the line between them intersects an obstacle, there is no physical link; *else* a physical link is placed between the nodes with probability p. We refer to p as the *connection probability* and $1-p$ as the *missing link* probability. If a graph created using the above procedure is disconnected, it is not used. Note that to replicate the connectivity graph of a real network, missing links between neighbors can be specified deterministically rather than with probability $1-p$.

Inaccurate coordinates. The known coordinates of a node may be highly inaccurate [17] because some localization methods have large location errors. In our experiments, after placing nodes in the physical space, their "known" coordinates are then generated with randomized location errors. The location errors are generated to satisfy a *location error ratio*, e, which is defined to be the ratio of the average location error to the average distance between nodes that are physical neighbors. We experimented with location error ratios from 0 to 2.

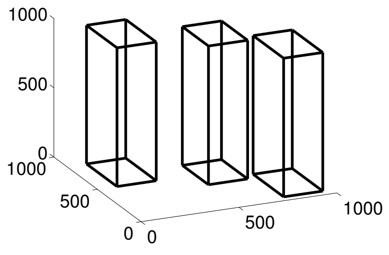
Definitions. The routing stretch value of a pair of nodes, s and d, in a multi-hop DT of S is defined to be the ratio of the number of physical links in the MDT route to the number of physical links in the shortest route in the connectivity graph between s and d. The ***routing stretch*** of the multi-hop 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 physical links as metric.

5.2 Design of experiments

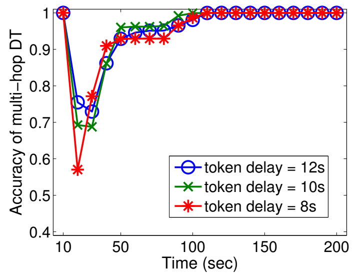
Our simulation experiments were designed to evaluate geographic routing in the most challenging environments. In general, everything else being the same, the challenge is bigger for a higher dimensional space, larger obstacles, a higher missing link probability, a lower node density, a larger network size, or larger node location errors. Furthermore, we performed experiments to evaluate MDT's resilience to dynamic topology changes at very high churn rates. In the geographic routing literature, no other protocol has been shown to meet all of these challenges.

Our simulator enables evaluation of geographic routing protocols in the most challenging environments. In the simulator, any connectivity graph can be created to represent any real network environment with obstacles of different shapes and sizes. The connectivity graphs created as described above have properties of real wireless networks, unlike unit-disk and unit-ball graphs used in prior work on geographic routing.[5] We experimented with obsta-

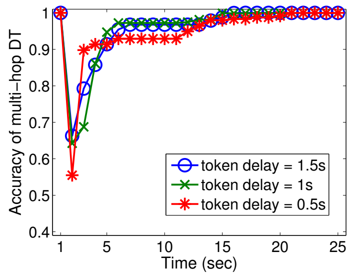
[5]In a very recent paper on 3D routing, unit-ball graphs were still used for simulation experiments [24].



(a) Obstacles in a 3D space



(b) Ave. message delay = 150 ms



(c) Ave. message delay = 15 ms

Figure 3: Accuracy vs. time for concurrent joins in 3D

cles of different shapes and sizes, and nodes with large location errors or arbitrary coordinates in 2D, 3D, and 4D. In this paper, we present experimental results for large obstacles, such as those shown in Figure 3(a), because large obstacles are more challenging to geographic routing than small ones; these very large obstacles may represent tall buildings in an outdoor space or large machinery in a factory. Between neighbors that are in line of sight and within radio transmission range, we experimented with a missing link probability as high as 0.5.

Node density is an important parameter that impacts geographic routing performance. We present experiments for node density of 13.5 for 3D and 9.7 for 2D. When we scale up the network size in a set of experiments, we increase the space and obstacle sizes to keep node density approximately the same. For experiments with different missing link probabilities, we vary the radio transmission range to keep node density approximately the same. We found that node densities lower than 13.5 for 3D and 9.7 for 2D would result in many disconnected graphs for spaces with large obstacles and a missing link probability of 0.5. The values of node density we used for experiments are relatively low compared with prior work on geographic routing. We also conducted experiments for higher node densities which resulted in better MDT performance, thus allowing us to conclude that MDT works well for a wide range of node densities.

5.3 System initialization experiments

We have performed numerous experiments using our initialization protocols. In every experiment, a correct multi-hop DT is constructed. Concurrent joins can do so much faster than serial joins but with a higher message cost (see Figure 10 for message cost comparison).

Figures 3(b)-(c) show results from two sets of experiments using concurrent-join initialization. In each experiment, the physical space is a $1000 \times 1000 \times 1000$ 3D space, with three large obstacles, placed as shown in Figure 3(a). The size of one obstacle is $200 \times 300 \times 1000$. Each of the other two is $200 \times 350 \times 1000$ in size. The obstacles occupy 20% of the physical space. Connectivity graphs are then created for 300 nodes using the procedure described in Section 5.1 for radio transmission range $R = 305$ and link connection probability $p = 0.5$. The *average node degree*, i.e., number of physical neighbors per node, is 13.5.[6] The (known) coordinates of the nodes are inaccurate with location error ratio $e = 1$.

The first set of experiments is for low-speed networks with one-hop message delays sampled from 100 ms to 200 ms (average = 150 ms) and a maintenance protocol timeout duration of 60 seconds.

[6]In 3D, a node density of 13.5 is fairly low and realistic.

The second set of experiments is for high-speed networks with one-hop message delays sampled from 10 ms to 20 ms (average = 15 ms) and a maintenance protocol timeout duration of 10 seconds.

In the legend of Figures 3(b)-(c),"token delay" is maximum token delay τ. In each 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% quickly. In all experiments, after each node's initial join, the node had run the maintenance protocol only once or twice by the time 100% accuracy was achieved.

5.4 MDT performance in 3D

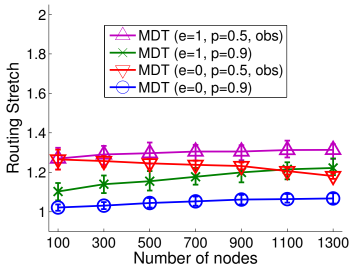
We evaluated the performance of MDT routing for 100 to 1300 nodes in 3D. We present results from four different sets of experiments using connectivity graphs created in a 3D space with and without obstacles, for node locations specified by accurate and inaccurate coordinates. There are four cases:

- accurate coordinates ($e = 0$), few missing links ($p = 0.9$), no obstacle
- inaccurate coordinates ($e = 1$), few missing links ($p = 0.9$), no obstacle
- accurate coordinates ($e = 0$), many missing links ($p = 0.5$), large obstacles (*obs*)
- inaccurate coordinates ($e = 1$), many missing links ($p = 0.5$), large obstacles (*obs*)

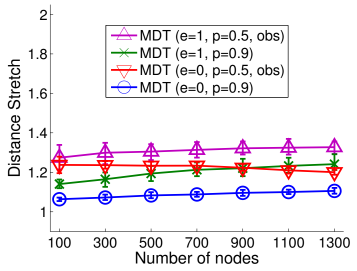
For 300 nodes, dimensions of the physical space and obstacles are the same as in Figure 3(a). For a smaller (or larger) number of nodes, dimensions of the physical space and obstacles are scaled down (or up) proportionally. For each *obs* experiment, the three obstacles are randomly placed in the horizontal plane. $R = 305$ is used for $p = 0.5$ and $R = 250$ is used for $p = 0.9$ such that the average node degree is approximately 13.5. At the beginning of each experiment, a correct multi-hop DT was first constructed. *Routing success rate was 100% in every experiment* and is not plotted.

Figures 4(a)-(b) show that both routing stretch and distance stretch versus network size are close to 1 for the easy case of accurate coordinates ($e = 0$), few missing links ($p = 0.9$), and no obstacle. Either inaccurate coordinates ($e = 1$) or many missing links ($p = 0.5$) and large obstacles (*obs*) increase both the routing stretch and distance stretch of MDT routing. Note that both the routing and distance stretch of MDT remain low as network size becomes large.[7]

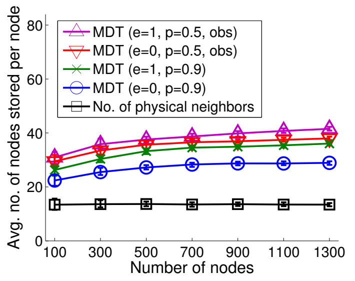
[7]Distance stretch is almost the same as routing stretch (except in 4D experiments for which physical distance is not meaningful) and will not be shown again.



(a) Routing stretch vs. N

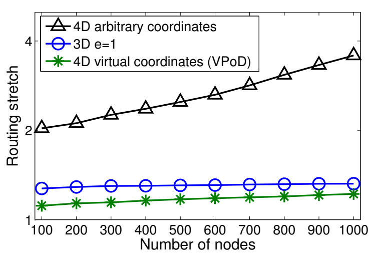


(b) Distance stretch vs. N

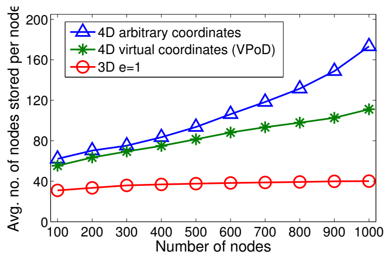


(c) Storage cost vs. N

Figure 4: MDT performance in 3D (average node degree=13.5)



(a) Routing stretch vs. N



(b) Storage cost vs. N

Figure 5: MDT performance in 3D and 4D (average node degree=13.5, *p*=0.5, obstacles)

Storage cost. The most important routing information stored in a node is the set of nodes it uses for forwarding; the known coordinates of each node in the set are stored in a *location table*. We use 4 bytes per dimension for storing each node's coordinates (e.g., 12 bytes for a node in 3D); this design choice is intended for very large networks. The coordinates of a node are used as its global identifier. Each node is also represented by a 1-byte local identifier in our current implementation. The location table stores pairs of global and local identifiers (e.g., 13 bytes per node for nodes in 3D). In the *forwarding table*, local identifiers are used to represent nodes in tuples. To illustrate MDT's storage cost in bytes, consider the case of 1300 nodes, $e = 1$, and $p = 0.5$ with obstacles. The average location table size is 540.2 bytes. The average forwarding table size is 88.8 bytes. The average location table size is 86% of the combined storage cost. We found that this percentage is unchanged for all network sizes (100 - 1300) in each set of experiments, indicating that the forwarding table size is also proportional to the number of distinct nodes stored.

In this paper, the storage cost is measured by the average number of distinct nodes a node needs to know (and store) to perform forwarding. This represents the storage cost of a node's minimum required knowledge of other nodes. This metric, unlike counting bytes, requires no implementation assumptions which may cause bias when different routing protocols are compared. Figure 4(c) shows the storage cost per node versus network size. As expected, either inaccurate coordinates ($e = 1$) or many missing links ($p = 0.5$) and large obstacles require more storage per node due to the need for more multi-hop DT neighbors. For comparison, the bottom curve is the average number of physical neighbors per node.

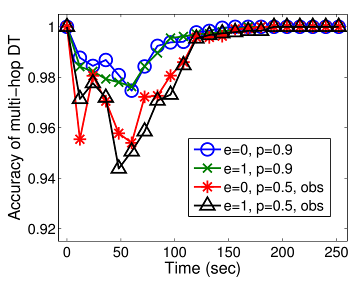
Varying obstacle locations. Each data point plotted in Figures 4(a)-(c) is the average value of 50 simulation runs for 50 different connectivity graphs each of which was created from a different placement of the obstacles. Also shown as bars are the 10th and 90th percentile values. Observe that the intervals between 10th and 90th percentile values are small for all data points. (These intervals are also small in experimental results to be presented in Figures 5 and 8-11 and will be omitted from those figures for clarity.) The small intervals between 10th and 90th percentile values demonstrate that varying obstacle locations has negligible impact on MDT routing performance.

Varying number and size of obstacles. Aside from varying the locations of obstacles, we also experimented with varying the number and size of obstacles. In particular, we repeated the experiments in Figure 4 for 6 obstacles and also for 9 obstacles. In each such experiment, the fraction of physical space occupied by obstacles was kept at 20%. We found the resulting changes in MDT's routing stretch, distance stretch, and storage cost to be too small to be visible when plotted in Figures 4.[8] However, when we increased the fraction of physical space occupied by obstacles from 20% to 30%, the resulting increases in MDT's routing and distance stretch were significant (about 6%).

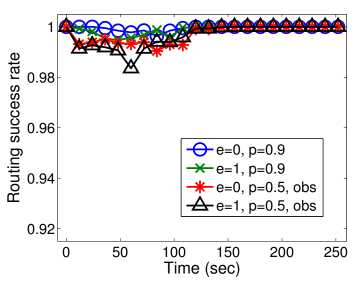
5.5 MDT performance in 4D

To illustrate how MDT can be used in 4D, consider the connectivity graphs created for the set of experiments in Figure 4 with many missing links ($p = 0.5$) and large obstacles. Suppose the nodes have *no location information*. We experimented with two cases: (i) Each node assigns itself an arbitrary location in a 4D space and sends its (arbitrary) coordinates to its physical neighbors. These coordinates are used by MDT protocols to construct and maintain a multi-hop DT as well as for routing. (ii) After a multi-hop DT has been constructed by the nodes using the initial (arbitrary) coordinates, each node then runs the VPoD protocol [21] to iteratively compute a better virtual position in the 4D space. VPoD is a virtual positioning protocol that does not require

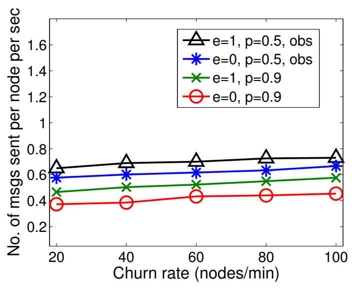
[8]Performance measures from experiments for 9 obstacles are smaller than those from experiments for 3 obstacles by less than 0.5%.



(a) Churn rate = 100 nodes/min.

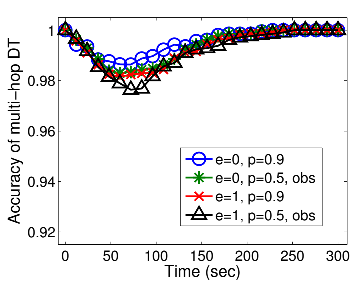


(b) Churn rate = 100 nodes/min.

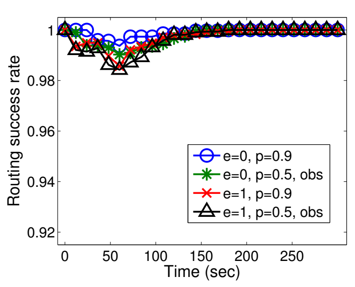


(c) Communication cost vs. churn rate

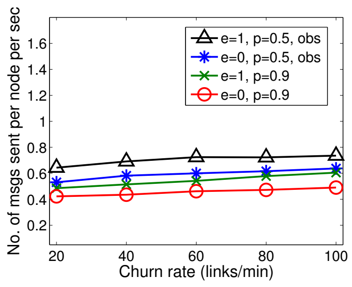
Figure 6: MDT performance under node churn (ave. message delay $= 150$ ms, timeout $= 60$ sec.)



(a) Churn rate = 100 links/min.



(b) Churn rate = 100 links/min.



(c) Communication cost vs. churn rate

Figure 7: MDT performance under link churn (ave. message delay $= 150$ ms, timeout $= 60$ sec.)

any node location information, any special nodes (such as, beacons and landmarks), nor the use of flooding. VPoD makes use of a multi-hop DT for routing support and link costs between physical neighbors for nodes to compute virtual positions. Any *additive routing metric* can be used for link costs in VPoD. For the results presented in Figure 5, we used 1 (hop) as the routing metric between two physical neighbors. (Each data point plotted in Figure 5 is the average value from 50 experiments.)

For comparison, we have also plotted the results for MDT routing using inaccurate coordinates ($e = 1$ case from Figure 4). Figure 5(a) on routing stretch, plotted in logarithmic scale, shows that MDT routing using 4D virtual coordinates is better than using inaccurate coordinates in 3D. Figure 5(b) on storage cost shows that MDT routing using inaccurate coordinates in 3D is better than using 4D virtual coordinates. In both figures, MDT routing using arbitrary coordinates has the worst performance. Routing success rate was 100% in every experiment and is not shown.

5.6 Resilience to Churn

We performed a large number of experiments to evaluate the performance of MDT protocols for systems under churn, with 300 nodes in a $1000 \times 1000 \times 1000$ 3D physical space. Like the experiments used to evaluate MDT routing stretch in Figure 4, four sets of experiments were performed using connectivity graphs created with and without three large obstacles, for node locations specified by accurate and inaccurate coordinates. The average node degree is kept at approximately 13.5 for every experiment.

In a *node churn* experiment, the rate at which new nodes join is equal to the churn rate; the rate of nodes leaving and the rate of nodes failing are each equal to half the churn rate. In a *link churn* experiment, the churn rate is equal to the rate at which new physical links are added and the rate at which existing physical links are deleted. In each experiment, the 300 nodes initially maintain a correct multi-hop DT. Churn begins at time=0 and ends at time=60 seconds.

Figure 6 presents results from node churn experiments for low-speed networks where one-hop message delays are sampled from [100 ms, 200 ms]. The maintenance timeout value is 60 seconds. 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.

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

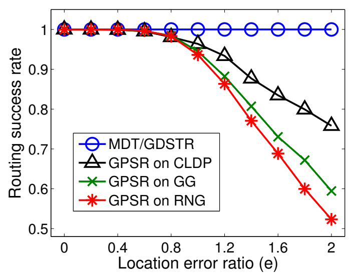
Figure 7 presents results from link churn experiments for low-speed networks with a maintenance timeout value of 60 seconds. Figure 7(a) shows the accuracy of the multi-hop DT versus time. The accuracy returns to 100% quickly after churn. Figure 7(b) shows the routing success rate versus time. The success 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.

Note that the convergence times to 100% accuracy in Figures 6(a) and 7(a) and to 100% success rate in Figures 6(b) and 7(b) are almost the same for the four cases. These results are typical of all churn experiments performed.

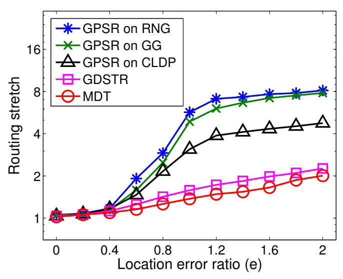
6. PERFORMANCE COMPARISON

6.1 Comparison of 2D protocols

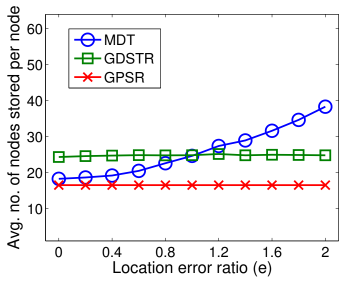
The geographic routing protocols, GPSR running on GG, RNG, and CLDP graphs [11, 12], and GDSTR [16] were designed for



(a) Routing success rate vs. *e*

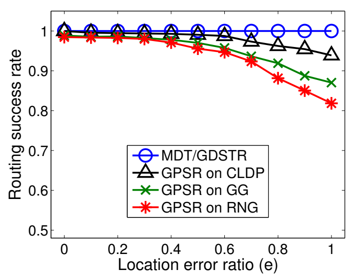


(b) Routing stretch vs. *e*

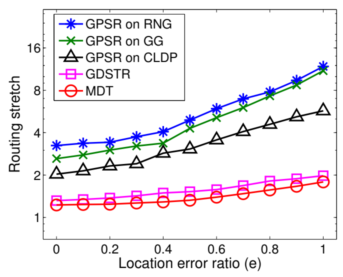


(c) Storage cost vs. *e*

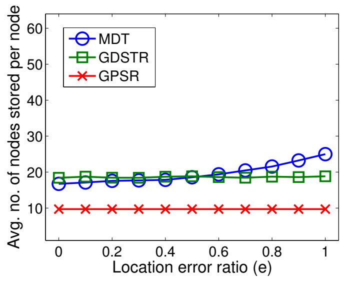
Figure 8: Performance comparison of 2D protocols (average node degree=16.5)



(a) Routing success rate vs. *e*



(b) Routing stretch vs. *e*



(c) Storage cost vs. *e*

Figure 9: Performance comparison of 2D protocols (three large obstacles, average node degree=9.7)

routing in 2D. We implemented these protocols in our simulator.[9] The experiments in Figure 8 were carried out for 300 nodes in a 1000×1000 2D space with no obstacle and few missing links ($p = 0.9$). The radio transmission range is $R = 150$. The average node degree is 16.5. The performance results are plotted versus location error ratio, from $e = 0$ (no error) to $e = 2$ (very large location errors).

The experiments of Figure 9 were carried out for 300 nodes in a 1000×1000 2D space with three randomly placed obstacles (a 200×300 rectangle and two 200×350 rectangles) and many missing links ($p = 0.5$). The radio transmission range is $R = 150$. The average node degree is 9.7. The performance results are plotted versus location error ratio, from $e = 0$ to $e = 1$.

In Figure 8(a) and Figure 9(a) the routing success rates of MDT and GDSTR are both 100% for all e values (it was 100% in every experiment). As the location error ratio (e) increases from 0, the routing success rates of RNG, GG, and CLDP drop off gradually from 100%. For $e > 0.6$ in Figure 8(a) and $e > 0.3$ in Figure 9(a), their routing success rates drop significantly.

Figure 8(b) and Figure 9(b), in logarithmic scale, show that MDT has the lowest routing stretch for all e values, with GDSTR a close second, followed by CLDP, GG, and RNG in that order. Note that routing stretch increases as e increases for all protocols.

Figure 8(c) and Figure 9(c) show storage cost comparisons. The GPSR protocols (CLDP, GG, and RNG) have the lowest storage cost, with the storage costs of GDSTR and MDT about the same.

Comparison of graph construction costs. We compare MDT's message cost to construct a correct multi-hop DT with message costs of CLDP graph construction using serial probes [12] and GDSTR hull tree construction [16]. The physical space is a 2D square with three large rectangular obstacles, occupying 20% of the physical space. There are many missing links ($p = 0.5$). Nodes have inaccurate coordinates ($e = 1$). The number N of nodes is varied from 100 to 1300. For the radio transmission range $R = 150$, the sizes of the physical space and obstacles are determined for each value of N such that the average node degree is approximately 12.

[9]Using, as our references, [12] 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/. GDSTR uses two hull trees [16]

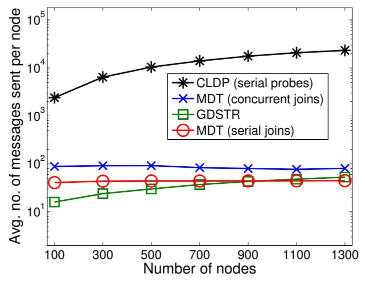
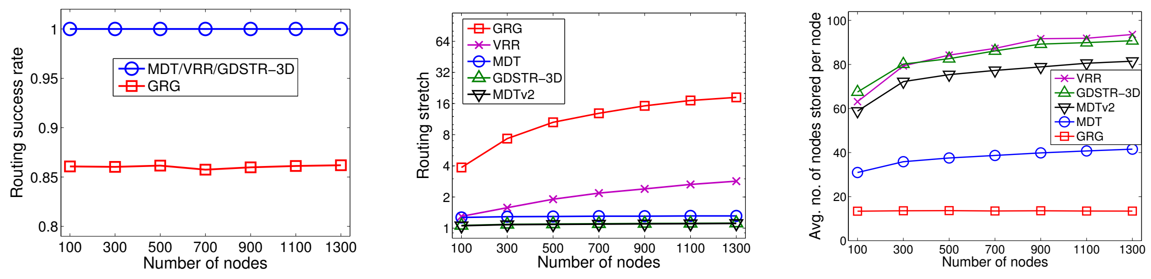


Figure 10: Initialization message cost vs. N (average node degree = 12)

In Figure 10, 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 only as one message sent. Messages sent by CLDP and MDT are unicast messages.

Figure 10 shows that with the average number of messages *sent* per node as metric, GDSTR has the best message cost performance for up to 900 nodes. For more than 900 nodes MDT (serial joins)



(a) Routing success rate vs. N (b) Routing stretch vs. N (c) Storage cost vs. N

Figure 11: Performance comparison of 3D protocols (average node degree=13.5)

has the lowest cost. CLDP has a very high cost. Note that the CLDP and GDSTR curves increase gradually with N. The MDT curves are flat.

6.2 Comparison of 3D Protocols

We compare the routing performance of MDT with GRG [7] and GDSTR-3D [24]. We implemented the basic version of GRG in our simulator. Several techniques to improve the performance of GRG are presented for *unit ball graphs* [7]. Since arbitrary connectivity graphs are used in our experiments, these techniques are not applicable and not implemented.

GDSTR-3D uses two hull trees for recovery. For each tree, each node stores two 2D convex hulls to aggregate the locations of all descendants in the subtree rooted at the node; the two 2D convex hulls approximate a 3D convex hull at each node. We implemented GDSTR-3D using its authors' TinyOS 2.x source code available at Google Sites.

Unlike other geographic protocols, each node in GDSTR-3D stores *2-hop neighbors* and uses 2-hop greedy forwarding to reduce routing stretch at the expense of a much larger storage cost per node. This performance tradeoff may not be appropriate for networks with limited nodal storage.

A non-geographic routing protocol, VRR [4], is included in the comparison. We implemented VRR for static networks without joins and failures.[10] For each pair of virtual neighbors, we used the shortest path (in hops) between them as the forwarding path (the routing stretch value is 1 between virtual neighbors). Thus, the routing stretch and storage cost results shown in Figure 11(b)-(c) for VRR are slightly optimistic. In VRR, each node also stores 2-hop neighbors for forwarding.

MDT can be easily modified to use 2-hop greedy forwarding. We present results for both MDT (which uses 1-hop greedy forwarding) and MDTv2 (which uses 2-hop greedy fowarding).

In our experiments, the number N of nodes is varied from 100 to 1300. The physical space and large obstacles are the same as the ones used in Figure 4. The average node degree was kept at approximately 13.5. Experiments were performed using connectivity graphs created for the following case: inaccurate coordinates ($e = 1$), many missing links ($p = 0.5$), and three large obstacles that occupy 20% of the physical space.

Figure 11(a) shows that MDT (also MDTv2), GDSTR-3D, and VRR all achieve 100% routing success rate while the routing success rate of GRG is about 86%. Figure 11(b), in logarithmic scale, shows that the routing stretch of GRG is very high, the routing stretch of VRR is high for $N > 300$, and both increase with N. The routing stretch of MDTv2 is the lowest and slightly lower than that of GDSTR-3D for every network size (the differences are, however, too small to be seen in Figure 11(b)). MDT, which uses 1-hop greedy forwarding, ranks a close third.

In Figure 11(c), GDSTR-3D, VRR, MDTv2 have large per-node storage costs, because each node stores 2-hop neighbors as well as physical neighbors. The storage cost of MDTv2 is smaller than those of GDSTR-3D and VRR. Both GRG and MDT have much lower storage costs because they use 1-hop greedy forwarding. The per-node storage cost of GRG, equal to the average number of physical neighbors, is the lowest of the five protocols.

MDT versus GDSTR-3D. MDT, MDTv2, and GDSTR-3D all provide guaranteed delivery in 3D and achieve routing stretch close to 1. GDSTR-3D has a higher storage cost than MDTv2 and a much higher storage cost than MDT. One clear advantage MDT (or MDTv2) has over GDSTR-3D is that MDT is highly resilient to dynamic topology changes (both node churn and link churn) while GDSTR-3D is designed for a static topology without provision to handle any dynamic topology change. Another advantage of MDT is that it provides guaranteed delivery for nodes with arbitrary coordinates in higher dimensions ($d > 3$).

7. CONCLUSIONS

MDT is the only geographic routing protocol that provides guaranteed delivery in 2D, 3D, and higher dimensions. The graph of nodes and physical links is required to be connected, but may otherwise be arbitrary. MDT's guaranteed delivery property holds for nodes with accurate, inaccurate, or arbitrary coordinates.

Experimental results show that MDT constructs a correct multi-hop DT very quickly at system initialization. MDT is also highly resilient to both node churn and link churn. Furthermore, MDT achieves a routing stretch (also distance stretch) close to 1.

The performance of MDT scales well to a large network size (N). We observed that, as N becomes large, MDT's routing (distance) stretch and per-node storage cost converge to horizontal asymptotes. MDT does not use special nodes (such as, beacons and landmarks) that are required in many wireless routing protocols; every MDT node runs the same protocols. Each node computes its own *local DT* with computation cost dependent upon its storage cost, rather than N. Lastly, MDT's per-node communication costs for constructing and maintaining a correct multi-hop DT are fairly low and independent of N.

8. ACKNOWLEDGMENTS

This work was sponsored by National Science Foundation grant CNS-0830939. We thank Ben Leong and our shepherd, Qin Lv, for their constructive comments. We also thank Jaeyoun Kim for programming help.

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

9. REFERENCES

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

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

[3] 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.

[4] 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.

[5] R. Draves, J. Padhye, and B. Zill. Routing in Multi-radio, Multi-hop Wireless Mesh Networks. In *Proceedings of ACM Mobicom*, 2004.

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

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

[8] 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.

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

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

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

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

[13] 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, December 2006.

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

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

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

[17] M. Li and Y. Liu. Rendered Path: Range-free Localization in Anisotropic Sensor Networks with Holes. In *Proceedings of ACM Mobicom*, 2007.

[18] 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.

[19] C. Liu and J. Wu. Efficient Geometric Routing in Three Dimensional Ad Hoc Networks. In *Proceedings of INFOCOM*, 2009.

[20] D. Pompili, T. Melodia, and I. Akyildiz. Routing algorithms for delay-insensitive and delay-sensitive applications in underwater sensor networks. In *Proc. 12th Int. Conf. on Mobile Computing and Networking*, 2006.

[21] C. Qian and S. S. Lam. Greedy Distance Vector Routing. In *Proceedings of IEEE ICDCS*, June 2011.

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

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

[24] J. Zhou, Y. Chen, B. Leong, and P. Sundaramoorthy. Practical 3D Geographic Routing for Wireless Sensor Networks. In *Proceedings of Sensys*, November 2010.

10. 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 forwarding protocol succeeds to find a node in S closest to ℓ in a finite number of hops.

PROOF. 1) By definition, a correct multi-hop DT of S is a correct distributed DT of S. The distributed 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 physical 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, say m, arrives at a node, say u, if the condition in line 1, 2, or 6 in Table 1 is true, then a node closest to ℓ is found. If the conditions in lines 1-3 are all false, node u performs greedy forwarding in lines 4-5. If it succeeds to find in P_u a physical neighbor v that is closer to ℓ than node u, message m is transmitted directly to v (lines 4 in Table 1); else, greedy forwarding is performed over the set of DT neighbors (line 5 in Table 1). The proof of Theorem 1 in [13] 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 forwarding code* (lines 4-5 in Table 1) finds a node v that is closer to ℓ than node u.

4) Any other node in S that is closer to ℓ than u will not use the actions in lines 4-5 in Table 1 to send message m back to node u. It is, however, possible for message m to visit node u again in the forwarding path between two DT neighbors that are closer to ℓ than u. In this case, the condition of line 3 in Table 1 must be true for m at node u. Thus, node u executes the greedy forwarding code for message m at most once. This property holds for every node. By 2), 3), and the assumption that S has a finite number of nodes, MDT forwarding 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 a property of DT, node z, being closest to w, is guaranteed to be a neighbor of w in $DT(S \cup \{w\})$. A forwarding path is constructed between w and z. 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 [13], the join process finishes and N_w consists of all neighbor nodes of w in $DT(S \cup \{w\})$.

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 (z) has a finite number of hops (by Theorem 1), and (ii) the path from the joining node to each of its other DT neighbors 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 by a node in three ways, which are defined in Figure 12 where

```
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 13: MDT forwarding protocol at node u

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 ← F_u ∪ {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 12: Three ways to send a message

m.dest and *m.source* denote the destination and source nodes of m, respectively, and *m.type* denotes its message type. A pseudocode specification of the MDT forwarding protocol in Table 1 is presented in Figure 13. The function, $Get\_Next(m.dest, m.relay)$ for message m being routed, executes lines 2-5 of the greedy forwarding protocol in Table 1. *Get_Next* returns null if node u is closest to the destination location; else it returns a physical neighbor to transmit message m to. A pseudocode specification of the MDT join protocol is presented in Figure 14.

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 [13]). 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 $DT(N_u)$,[11] 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.

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 Figure 15.

Failure protocol. The failure protocol is similar to the leave protocol and almost as efficient. The key idea is that every node u

[11]Note that u is not in N_u.

Notation and Definitions:

When a node receives a message, it stores the message in the local data structure m. We define four 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 location (or 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 DT 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. end if
```

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

```
1.    e ← 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 u 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.       $N_w^u \leftarrow \{e \mid e$ is a neighbor of w 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 ← 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 ← 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 ← 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 $x \in N_u^{new}$ do
5.       Send(<x, NB_SET_REQ, u, relay>, successor)
6.     end for
```

Figure 14: MDT join protocol at 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. (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.

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 15: MDT leave protocol at node *u*