

# Geographic Routing in $d$ -Dimensional Spaces With Guaranteed Delivery and Low Stretch

Simon S. Lam, *Fellow, IEEE, ACM*, and Chen Qian, *Student Member, IEEE*

**Abstract**—Almost all geographic routing protocols have been designed for 2-D. We present a novel geographic routing protocol, named Multihop Delaunay Triangulation (MDT), for 2-D, 3-D, and higher dimensions with these properties: 1) guaranteed delivery for any connected graph of nodes and physical links, and 2) 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 for routing. We present the performance of MDT protocols in 3-D and 4-D as well as performance comparisons of MDT routing versus representative geographic routing protocols for nodes in 2-D and 3-D. 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 after churn.

**Index Terms**—Distributed Delaunay triangulation, geographic routing, greedy forwarding, protocol design, resilience to churn.

## I. 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 2-D. In reality, many wireless applications run on nodes located in 3-D [22], [1], [7], [8]. 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

Manuscript received July 05, 2011; revised January 11, 2012; accepted June 18, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Westphal. Date of publication September 06, 2012; date of current version April 12, 2013. This work was supported by the National Science Foundation under Grant CNS-0830939. An abbreviated version of this paper appeared in the Proceedings of the ACM SIGMETRICS Conference, San Jose, CA, June 7–11, 2011.

The authors are with the Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: lam@cs.utexas.edu; cqian@cs.utexas.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TNET.2012.2214056

general connectivity graphs in 3-D, face routing methods designed for 2-D [4], [13], [14] are not applicable. Furthermore, Durocher *et al.* [7] 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 3-D is a challenging problem.

We present in this paper a novel geographic routing protocol, Multihop Delaunay Triangulation (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 2-D and 3-D when coordinates specifying node locations are accurate.<sup>1</sup> 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 [6]). For example, a 4-D virtual space can be used for geographic routing of nodes physically located in a 3-D 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 [23] where we show how to: 1) make use of MDT protocols to embed routing costs in virtual spaces (such as 4-D), and 2) 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 2-D, Bose and Morin proved that greedy routing in a DT always finds a given destination node [3]. Lee and Lam [16] generalized their result and proved that in a  $d$ -dimensional Euclidean

<sup>1</sup>Routing and distance stretch are defined later.

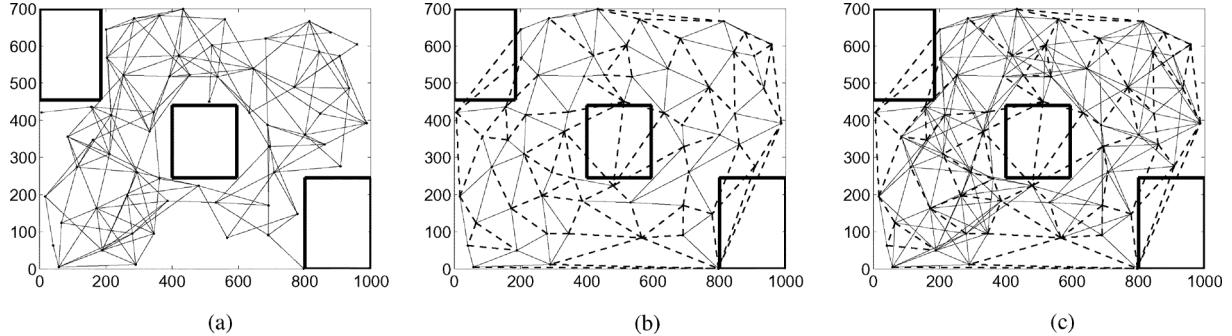


Fig. 1. Illustration of (a) connectivity, (b) DT, and (c) MDT graphs of a set of nodes in 2-D.

space ( $d \geq 2$ ), given a destination location  $\ell$ , greedy routing in a DT always finds a node that is closest to  $\ell$ .

Fig. 1(a) shows a 2-D space with three large obstacles and an arbitrary connectivity graph. Fig. 1(b) shows the DT graph [9] of the nodes in Fig. 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 Fig. 1(a) is illustrated in Fig. 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 that is guaranteed to succeed.

In this paper, we present MDT protocols for a set of nodes to construct and maintain a *correct multihop DT* (formal definition in Section II). In a multihop 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 multihop 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 proven correct for a single join. Thus, it constructs a correct multihop DT when nodes join serially. The maintenance protocol enables concurrent joins at system initialization. Experimental results show that MDT constructs a correct multihop 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 multihop 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 multihop 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 multihop DT is correct.

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 layer-2 routing in wireline networks. For example, ROME [24], an architecture for metropolitan-scale layer-2 networks, makes use of MDT routing in virtual spaces.

For a given set of nodes, under the restrictive assumption that every node can directly communicate with every other node, Lee and Lam [16], [17] presented protocols for the construction and maintenance of a correct distributed DT. These protocols cannot be used for routing in an arbitrary connectivity graph because their assumption is not satisfied. Major contributions of this paper include the definition of a correct distributed multihop DT, a new two-step greedy forwarding protocol, proofs of guaranteed delivery by the new forwarding protocol and correctness of the join protocol, as well as designing each protocol in the MDT suite to *correctly* construct/repair forwarding tables in paths between multihop DT neighbors to provide a correct distributed multihop DT.

#### A. 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 [30], requirements are imposed on the placement of nodes and links in 2-D such the DT graph is a subgraph of the connectivity graph. In other approaches, the restricted DT graph [11] and the  $k$ -localized DT [20] 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 2-D 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 2-D based upon greedy forwarding. Two of the earliest protocols, GFG [4] and GSPR [13], use face routing to move packets out of local minima. Bose *et al.* [2] proposed other memoryless routing algorithms for special planar graphs. These

protocols provide guaranteed delivery for a planar graph. If the connectivity graph is not planar, a planarization algorithm (such as GG [10] or RNG [26]) 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.* [14] 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, GSPR routing uses the subgraph produced by CLDP instead of by GG or RNG. CLDP was designed to provide guaranteed delivery for nodes in 2-D under the assumption that there are no degenerate link crossings caused by exactly colinear links [14].

Leong *et al.* proposed GDSTR [18], 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 [21] and GRG [8] are geographic protocols designed for 3-D. 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. For Poisson distributed nodes and ball graphs in 3-D [28] (disk graphs in 2-D [27]), the use of a “sufficiently” large transmission radius can provide guaranteed delivery.

Aside from MDT, there is one other geographic routing protocol that provides guaranteed delivery for general connectivity graphs in 3-D, namely, GDSTR-3D [31]. 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.

Embedding nodes into a virtual space and assigning them virtual coordinates can provide guaranteed delivery for geographic routing. The VPoD protocol [23] can be used by nodes to assign themselves coordinates in a Euclidean virtual space such that the distance between any two nodes in the space is a good estimate of the routing cost (in any additive metric) between them. For quasi-unit disk graph models, Sarkar *et al.* [25] proposed to use conformal mapping computed with *Ricci flow* to obtain an embedding such that greedy forwarding guarantees delivery. Xia *et al.* [29] proposed a unit tetrahedron cell mesh structure to obtain virtual coordinates to achieve deterministic greedy routing in 3-D. PIE [12] produces a greedy embedding by isometrically embedding trees in non-Euclidean spaces of dimension  $O(\log^2(n))$ . Its design targets Internet-like graphs. Each node in the graph belongs to  $O(\log(n))$  trees. PIE achieves a success ratio of 100% for any connected graph.

### B. Outline

The balance of this paper is organized as follows. In Section II, we present concepts, definitions, and model assumptions. In Section III, we present the MDT forwarding protocol. In Section IV, we present join, maintenance, and initialization

TABLE I  
DEFINITIONS OF TERMS

distributed DT	defined in Section II.A
multi-hop DT	defined in Section II.C
DT edge	an edge in a DT graph
DT neighbors or neighbors	two nodes that share a DT edge
physical neighbors	two nodes connected by a physical link
multi-hop neighbors	DT neighbors that are not physical neighbors
virtual link	a path provided by soft-state forwarding tables in nodes along the path
DT node	node that has finished executing the join protocol
node attached to the DT	node that has sent a join request and received a join reply from a DT node

protocols. In Section V, we present an experimental performance evaluation of MDT in 3-D and 4-D. We also present experimental results to demonstrate MDT’s resilience to node churn and link churn. In Section VI, we present performance comparisons of MDT to geographic routing protocols designed for 2-D and 3-D. We conclude in Section VII.

## II. CONCEPTS AND DEFINITIONS

A triangulation of a set  $S$  of nodes (points) in 2-D is a maximal planar subdivision<sup>2</sup> of the convex hull of nodes in  $S$  into nonoverlapping triangles such that the vertices of each triangle are nodes in  $S$ . A DT in 2-D is a triangulation such that the circumcircle of each triangle does not contain any other node inside [9]. 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 II-A, we present the definition of a *distributed DT* and a key result from Lee and Lam [16], [17] that we need later. (See Table I for definitions and terms.)

### A. Distributed DT

A distributed DT of a set  $S$  of nodes is specified by  $\{\langle u, N_u \rangle \mid u \in S\}$ , where  $N_u$  represents the set of  $u$ ’s neighbor nodes, which is locally determined by  $u$ .

*Definition 1:* 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,  $u \in S$ , discovers 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 information discovered by  $u$ , while  $S$  is global knowledge. For the extreme case of  $C_u = S$ ,  $u$  is guaranteed

<sup>2</sup>A maximal planar subdivision is a subdivision such that no more edges connecting two nodes can be added to the subdivision without destroying its planarity.

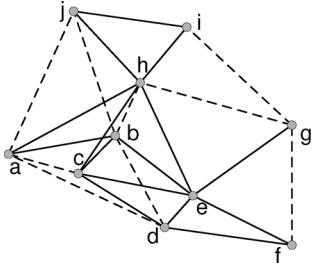


Fig. 2. MDT graph of 10 nodes.

to know its neighbors in  $DT(S)$ . However, the communication cost for each node to discover  $S$  (using, for example, a broadcast protocol) would be very high and not scalable. A *necessary and sufficient condition* [16], [17] for a distributed DT to be *correct* is that for all  $u \in S$ ,  $C_u$  includes all neighbor nodes of  $u$  in  $DT(S)$ . The condition's necessity is obvious. Its sufficiency requires a nontrivial proof (see [17]). This result enabled the design of efficient protocols for distributed DT construction.

### B. 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.<sup>3</sup>

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.<sup>4</sup> Furthermore, new physical links may be added, and existing physical links that have become error-prone are deleted.

Each node runs the same protocols. After a node boots up, it knows all of its physical neighbors. Subsequently, it discovers other nodes, including its multihop DT neighbors, from sending and receiving protocol messages.

### C. Multihop DT

A multihop DT is specified by  $\{\langle u, N_u, F_u \rangle \mid u \in S\}$ , where  $F_u$  is a soft-state forwarding table, and  $N_u$  is  $u$ 's neighbor set that is derived from information in  $F_u$ . The multihop 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 use the term “neighbor” to refer to a DT neighbor.) In a multihop DT, the neighbor of a node may not be a physical neighbor; see, for example, nodes  $i$  and  $g$  in Fig. 2.

For a node  $u$ , each entry in its forwarding table  $F_u$  is a 4-tuple  $\langle source, pred, succ, dest \rangle$ , 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

<sup>3</sup>Only links that are reliable and have an acceptable error rate are included in the connectivity graph.

<sup>4</sup>When a node fails, it becomes silent.

$t.source$ ,  $t.pred$ ,  $t.succ$ , 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,  $\langle a, b, c, d \rangle$  is in  $F_u$  if and only if  $\langle d, c, b, a \rangle$  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  $\langle -, -, succ, dest \rangle$ , which does not have a reverse in  $F_u$ .

For an example of a forwarding path, consider the MDT graph in Fig. 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:  $\langle -, -, e, i \rangle$  in node  $g$ ,  $\langle g, g, h, i \rangle$  in node  $e$ ,  $\langle g, e, i, i \rangle$  in node  $h$ , and  $\langle -, -, h, g \rangle$  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 2:* A multihop DT of  $S$ ,  $\{\langle u, N_u, F_u \rangle \mid u \in S\}$ , is *correct* if and only if the following conditions hold: 1) the distributed DT of  $S$ ,  $\{\langle u, N_u \rangle \mid u \in S\}$ , is correct; and 2) 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 multihop 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 multihop 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 multihop DT of  $S$ . The *accuracy* of  $MDT(S)$  is defined to be

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

It is straightforward to prove that the accuracy of  $MDT(S)$  is 1 (or 100%) if and only if the multihop 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  $\langle -, -, -, v \rangle$ . 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  $\langle -, -, v, v \rangle$ . 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.

### III. 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 multihop DT neighbor closest to  $d$ .

TABLE II  
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)

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:  $m = \langle m.dest, m.source, m.relay, m.data \rangle$  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 II. To forward message  $m$  to a node closest to location  $m.dest$ , the conditions in Table II 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 multihop DT neighbors.

The following theorem, which states that MDT forwarding in a correct multihop DT provides guaranteed delivery, is proved in the Appendix.

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

#### IV. 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 multihop DT, after a new node has finished joining the system, the resulting multihop DT is correct. This property ensures that a correct multihop 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 multihop DT by definition, and letting the other nodes join the existing multihop 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 multihop DT concurrently and independently using the MDT join protocol, the resulting multihop 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 multihop DT to converge to 100% accuracy.

MDT includes leave and failure protocols designed for a single leave and failure, respectively, for two 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 before the node leaves or fails. (For failure recovery, each node  $u$  prestores 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. 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 protocol to run. The leave and failure protocols are presented in the Appendix.

For a multihop 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 multihop neighbors. Leave, failure, and maintenance protocols construct a new path between two multihop neighbors whenever the previous path between them has been broken due to a node leave/failure or a link deletion.

##### A. 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.<sup>5</sup> In the MDT join protocol, a node uses the basic search technique of Lee and Lam [16] 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 the new neighbor's *local* view. If more new neighbors are found in the replies,  $w$  sends a neighbor-set request to each of them. This search process is iteratively repeated until  $w$  finds no more new neighbor in the replies. The MDT join protocol also constructs a forwarding path between  $w$  and every one of its multihop DT neighbors. A more detailed protocol description follows.

<sup>5</sup>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.

*Finding the Closest Node and Path Construction:* Node  $w$  joins by sending a join request to node  $v$  with  $w$ 's 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  $\langle -, -, v, v \rangle$  in its forwarding table. Subsequently, suppose an intermediate node (say  $u$ ) receives the join request from a physical neighbor (say  $v$ ) and forwards it to a physical neighbor (say  $e$ ), the tuple  $\langle w, v, e, e \rangle$  is stored in  $F_u$ .

When node  $z$  finally receives the join request of  $w$  from a physical neighbor (say  $d$ ), it stores the tuple  $\langle -, -, d, w \rangle$  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 physical neighbor  $e$ . Node  $x$  changes the existing tuple  $\langle e, e, *, w \rangle$  in  $F_x$  to  $\langle z, e, *, w \rangle$ , 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  $\langle -, -, -, w \rangle$  to  $\langle -, -, w, w \rangle$ .

*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  $t.dest = m.dest$ ). 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: There is a tuple  $t$  in the forwarding table such that  $t.succ = t.dest = m.dest$ , 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.

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  $w$  iteratively repeats the above search process 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 [16], [17]. 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,  $C_u$  consists of nodes in  $\{u\} \cup \{v \mid v = t.dest, t \in F_u\}$  as well as new nodes that may become DT neighbors. 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. Furthermore, whenever a tuple  $t$  is deleted from  $F_u$  upon timeout, each endpoint of  $t$  is deleted from  $C_u$  unless it is also an endpoint of another tuple.

*Path Construction to Multihop DT Neighbors:* The MDT join protocol also constructs a forwarding path between the joining node  $w$  and each of its multihop 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 multihop 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  $w$ .

Note that  $m.relay$  serves two different functions in different types of MDT protocol messages [15]. In a data message (also a join request message),  $m.relay$  is used to indicate a multihop DT neighbor that can route the message out of a local minimum. In a neighbor-set request message sent by a joining node (say  $u$ ),  $m.relay$  is the node that previously informed  $u$  of the new node to which the neighbor-set request is destined.

*Example:* Let node  $a$  in Fig. 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 multihop DT (of nine 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 Fig. 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.

*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 multihop 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 multihop DT of  $S \cup \{w\}$  is correct.

A proof of Theorem 2 is presented in the Appendix. A pseudocode specification of the MDT join protocol can be found in our technical report [15].

### B. 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. Toward 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.<sup>6</sup>

*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 to correct 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. We found that if each node runs the maintenance protocol repeatedly, the node states converge to a correct multihop DT very quickly. (See results from our system initialization experiments in Section V-C and churn experiments in Section V-F.)

### C. 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

<sup>6</sup>The maintenance protocol can use the same iterative search technique used in the join protocol. However, experimental results show that condition *C1* can be used to reduce the number of neighbor-set messages sent by the maintenance protocol without any impact on its effectiveness to find DT neighbors.

a token to each physical neighbor that is not known to have joined the multihop DT. Each token is sent after a random delay uniformly distributed over time interval  $[1, \tau]$ , where  $\tau$  is in seconds. If a node receives more than one token, any duplicate token is discarded.

## V. PERFORMANCE EVALUATION

### A. 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 3-D (2-D) 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. Alternatively, 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 [19] 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 multihop 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 multihop DT is defined to be the average of the routing stretch values of all source–destination pairs in  $S$ . The *distance stretch* of the multihop DT is defined similarly with distance replacing number of physical links as metric.

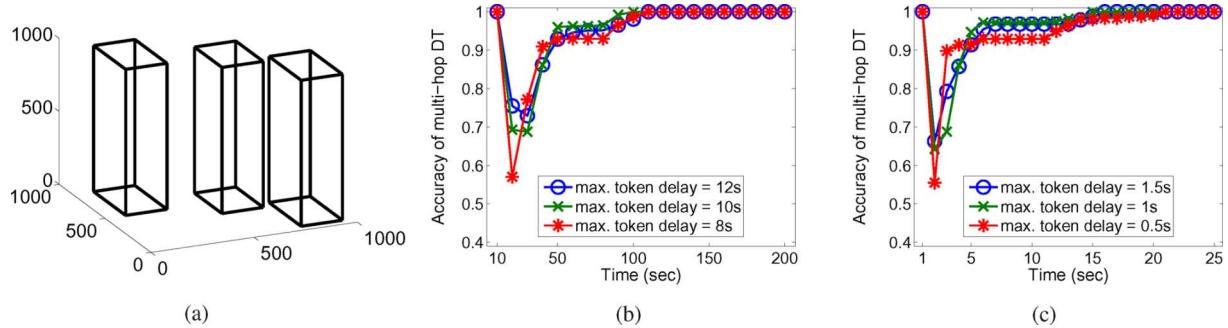


Fig. 3. Accuracy versus time for concurrent joins in 3-D. (a) Obstacles in a 3-D space. (b) Ave. message delay = 150 ms. (c) Ave. message delay = 15 ms.

### B. Design of Experiments

Our simulation experiments were designed to evaluate geographic routing in the most challenging environments. In general, everything else being equal, 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.<sup>7</sup> We experimented with obstacles of different shapes and sizes and nodes with large location errors or arbitrary coordinates in 2-D, 3-D, and 4-D. In this paper, we present experimental results for large obstacles, such as those shown in Fig. 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 (average node degree) is an important parameter that impacts geographic routing performance. We present experiments for node densities of 13.5 for 3-D and 9.7 for 2-D. These node densities are relatively low compared to node densities used in prior work on geographic routing. We found that node densities lower than 13.5 for 3-D and 9.7 for 2-D would result in many disconnected graphs for spaces with large obstacles and a high missing link probability. We also conducted experiments for higher node densities that resulted in better MDT performance, thus allowing us to conclude that MDT works well for a wide range of node densities. 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

<sup>7</sup>In a recent paper on 3-D routing, unit-ball graphs were still used for simulation experiments [31].

radio transmission range to keep node density approximately the same.

### C. System Initialization Experiments

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

Fig. 3(b) and (c) shows results from two sets of experiments using concurrent-join initialization. In each experiment, the physical space is a  $1000 \times 1000 \times 1000$  3-D space, with three large obstacles, placed as shown in Fig. 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 V-A 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. 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 to 200 ms (average = 150 ms) and a maintenance protocol timeout duration of 60 s. The second set of experiments is for faster networks with one-hop message delays sampled from 10 to 20 ms (average = 15 ms) and a maintenance protocol timeout duration of 10 s.

In the legend of Fig. 3(b) and (c), “max. token delay” is maximum token delay  $\tau$ . In each experiment, note that accuracy of the multihop DT is low initially when many nodes are joining at the same time. With a smaller  $\tau$ , more nodes initiate their join process earlier at about the same time, resulting in a lower MDT accuracy at the beginning. However, accuracy improves and converges to 100% quickly for all  $\tau$  values. In every experiment, after each node's initial join, the node had run the maintenance protocol only once or twice by the time 100% accuracy was achieved.

### D. MDT Performance in 3-D

We evaluated the performance of MDT routing for 100–1300 nodes in 3-D. We present results from four different sets of experiments using connectivity graphs created in a 3-D space

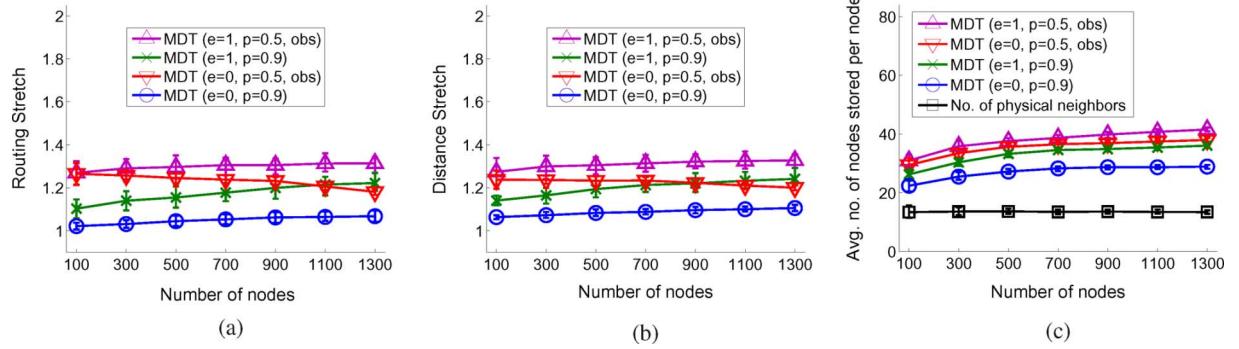


Fig. 4. MDT performance in 3-D (average node degree = 13.5). (a) Routing stretch versus  $N$ . (b) Distance stretch versus  $N$ . (c) Storage cost versus  $N$ .

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 Fig. 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 multihop DT was first constructed. *Routing success rate was 100% in every experiment* and is not plotted.

Fig. 4(a) and (b) shows 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.<sup>8</sup>

**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 B per dimension for storing each node's coordinates (e.g., 12 B for a node in 3-D); 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-B local identifier in our current implementation. The location table stores pairs of global and local identifiers (e.g., 13 B per node for nodes in 3-D). 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  $p = 0.5$

<sup>8</sup>Distance stretch is almost the same as routing stretch (except in 4-D experiments for which physical distance is not meaningful) and will not be shown again.

with obstacles. The average location table size is 540.2 B. The average forwarding table size is 88.8 B. 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 that may cause bias when different routing protocols are compared. Fig. 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 multihop DT neighbors. For comparison, the bottom curve is the average number of physical neighbors per node.

**Varying Obstacle Locations:** Each data point plotted in Fig. 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 Figs. 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 Fig. 4 for six obstacles and also for nine 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 Fig. 4.<sup>9</sup> 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%).

<sup>9</sup>Performance measures from experiments for nine obstacles are smaller than those from experiments for three obstacles by less than 0.5%.

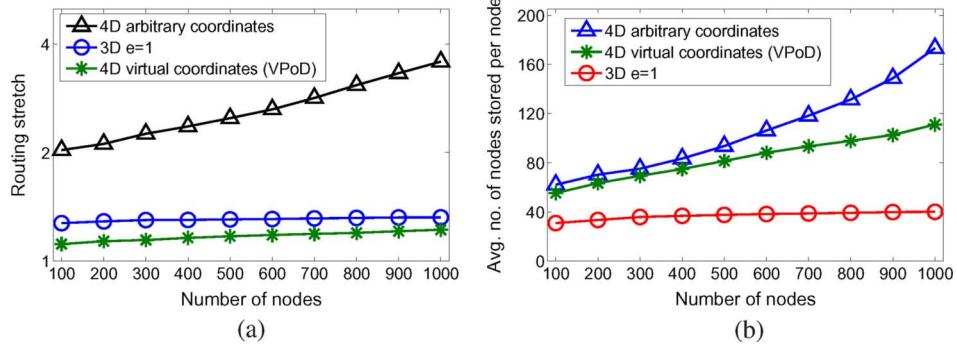


Fig. 5. MDT performance in 3-D and 4-D (average node degree = 13.5,  $p = 0.5$ , obstacles). (a) Routing stretch versus  $N$ . (b) Storage cost versus  $N$ .

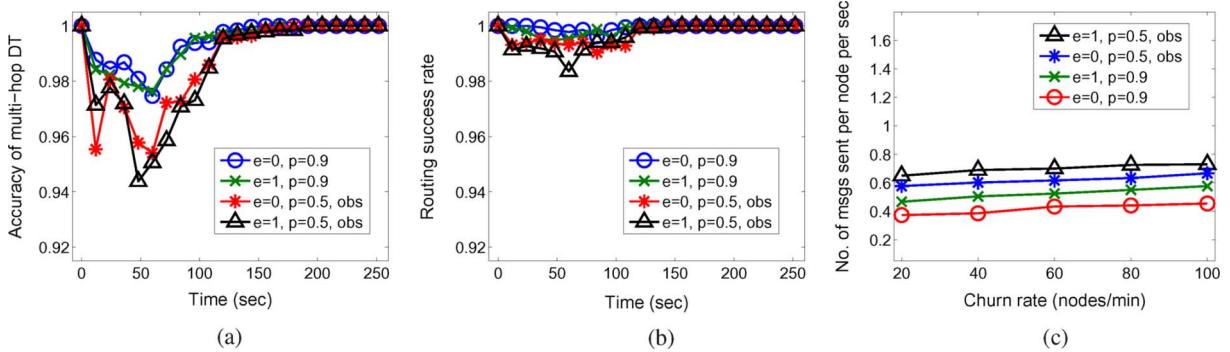


Fig. 6. MDT performance under node churn (ave. message delay = 150 ms, timeout = 60 s). (a) Churn rate = 100 nodes/min. (b) Churn rate = 100 nodes/min. (c) Communication cost versus churn rate.

#### E. MDT Performance in 4-D

To illustrate how MDT can be used in 4-D, consider the connectivity graphs created for the set of experiments in Fig. 4 with many missing links ( $p = 0.5$ ) and large obstacles. Suppose the nodes have *no location information*. We experimented with two cases. 1) Each node assigns itself an arbitrary location in a 4-D space and sends its (arbitrary) coordinates to its physical neighbors. These coordinates are used by MDT protocols to construct and maintain a multihop DT as well as for routing. 2) After a multihop DT has been constructed by the nodes using the initial (arbitrary) coordinates, each node then runs VPoD [23], which is a virtual positioning protocol that does not require any node location information, any special nodes (such as, landmarks), nor any use of flooding. Nodes use VPoD to change their coordinates by comparing distances with routing costs to their physical and DT neighbors. A new multihop DT is then constructed using the new coordinates. After several iterations, the node coordinates will converge to achieve the following property [23]: The distance between any two nodes in the virtual space is a good estimate of the routing cost (*in any additive metric*) between them.

For the results presented in Fig. 5, we used 1 (hop) as the routing metric between two physical neighbors. Each data point plotted in Fig. 5 is the average value from 50 experiments.

For comparison, we have also plotted the results for MDT routing using inaccurate 3-D coordinates, that is, the case of ( $e = 1, p = 0.5, \text{obs}$ ) in Fig. 4. Fig. 5(a) on routing stretch, plotted in logarithmic scale, shows that MDT routing using 4-D virtual coordinates is better than using inaccurate coordinates

in 3-D. Fig. 5(b) on storage cost shows that MDT routing using inaccurate coordinates in 3-D is better than using 4-D 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.

#### F. 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  $1000 \times 1000 \times 1000$  3-D physical space. Like the experiments used to evaluate MDT routing stretch in Fig. 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 multihop DT. Churn begins at time = 0 and ends at time = 60 s.

Fig. 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 s. The churn rate is 100 nodes/min in Fig. 6(a) and (b) and varies in Fig. 6(c). Fig. 6(a) shows the accuracy of the multihop DT versus time. The accuracy returns to 100% quickly after

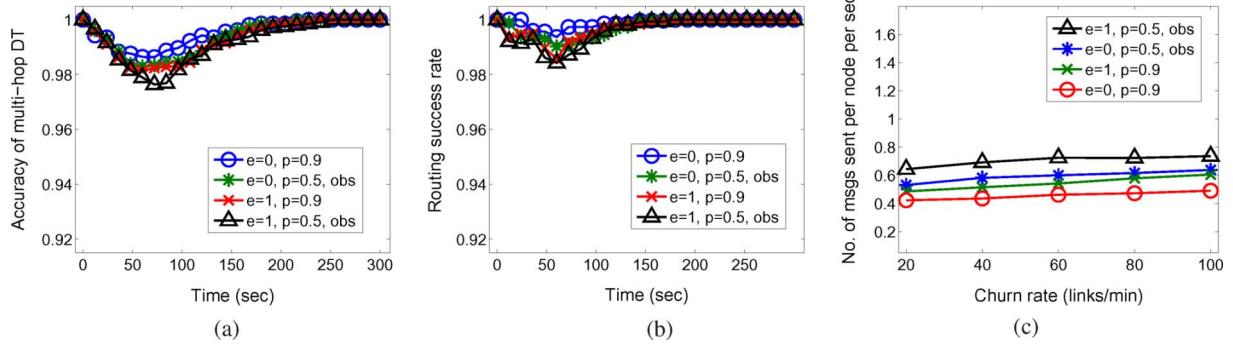


Fig. 7. MDT performance under link churn (ave. message delay = 150 ms, timeout = 60 s). (a) Churn rate = 100 links/min. (b) Churn rate = 100 links/min. (c) Communication cost versus churn rate.

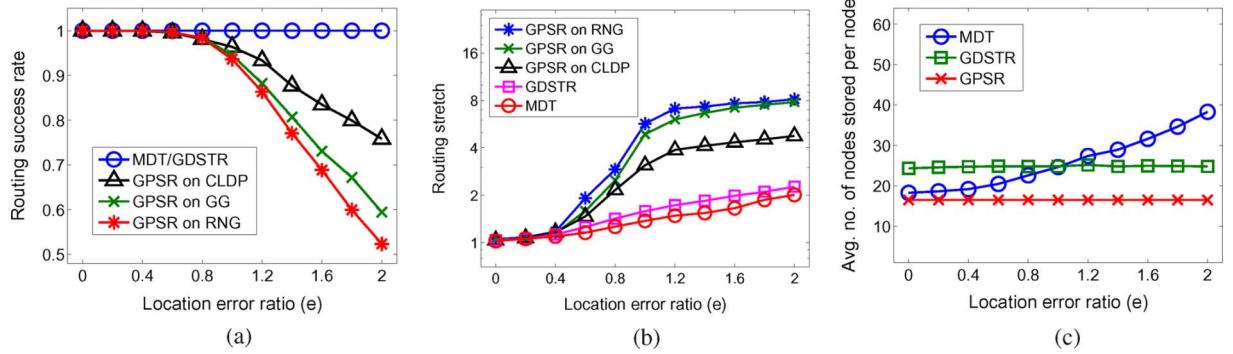


Fig. 8. Performance comparison of 2-D protocols (average node degree = 16.5). (a) Routing success rate versus  $e$ . (b) Routing stretch versus  $e$ . (c) Storage cost versus  $e$ .

churn. Fig. 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. Fig. 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  $300/100 = 3$  min, which represents a very high churn rate for most practical systems.

Fig. 7 presents results from link churn experiments for low-speed networks with a maintenance timeout value of 60 s. Fig. 7(a) shows the accuracy of the multihop DT versus time. The accuracy returns to 100% quickly after churn. Fig. 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. Fig. 7(c) shows the communication cost (per node per second) versus churn rate.

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

#### G. Discussion on Overheads

Nodes that implement MDT protocols incur extra storage and communication costs when compared to a simple greedy routing protocol. The extra storage cost of MDT is the difference between the MDT storage cost and the number of physical neighbors; see Fig. 4(c). Observe that the extra storage cost converges to an asymptotic constant as  $N$  becomes large. There are two

types of extra communication costs: 1) communication costs to construct a multihop DT initially (see Fig. 10 in the next section), and 2) communication costs incurred by the maintenance protocol during churn [see Figs. 6(c) and 7(c)]. The per-node churn cost is less than 0.8 messages/s for very high churn rates and the most difficult case. Note that when the network topology is static, MDT incurs (essentially) no extra overhead.

## VI. PERFORMANCE COMPARISON

### A. Comparison of 2-D Protocols

The geographic routing protocols, GPSR running on GG, RNG, and CLDP graphs [13], [14], and GDSTR [18] were designed for routing in 2-D. We implemented these protocols in our simulator.<sup>10</sup> The experiments in Fig. 8 were carried out for 300 nodes in a  $1000 \times 1000$  2-D 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 Fig. 9 were carried out for 300 nodes in a  $1000 \times 1000$  2-D space with three randomly placed obstacles (a  $200 \times 300$  rectangle and two  $200 \times 350$  rectangles) and many missing links ( $p = 0.5$ ). The radio transmission range is  $R =$

<sup>10</sup>Using, as our references, [14] for CLDP; GDSTR code from <http://www.comp.nus.edu.sg/~bleong/geographic/>; and GPSR, GG, and RNG code from <http://www.cs.ucl.ac.uk/staff/B.Karp/gpsr/>. GDSTR uses two hull trees [18].

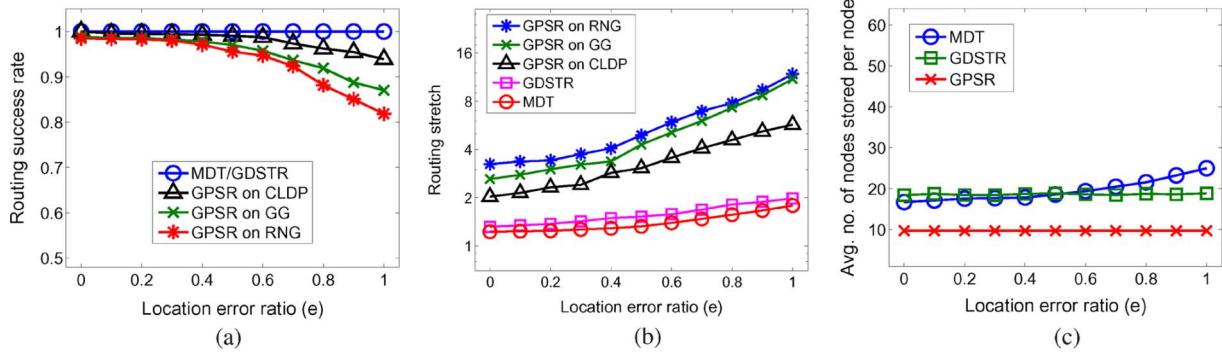


Fig. 9. Performance comparison of 2-D protocols (three large obstacles, average node degree = 9.7). (a) Routing success rate versus  $e$ . (b) Routing stretch versus  $e$ . (c) Storage cost versus  $e$ .

150. The average node degree is 9.7. The performance results are plotted versus location error ratio, from  $e = 0$  to  $e = 1$ .

In Figs. 8(a) and 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 Fig. 8(a) and  $e > 0.3$  in Fig. 9(a), their routing success rates drop significantly.

Figs. 8(b) and 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.

Figs. 8(c) and 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 multihop DT with message costs of CLDP graph construction using serial probes [14] and GDSTR hull tree construction [18]. The physical space is a 2-D 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.

In Fig. 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.

Fig. 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) 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.

### B. Comparison of 3-D Protocols

We compare the routing performance of MDT with GRG [8] and GDSTR-3D [31]. We implemented the basic version of

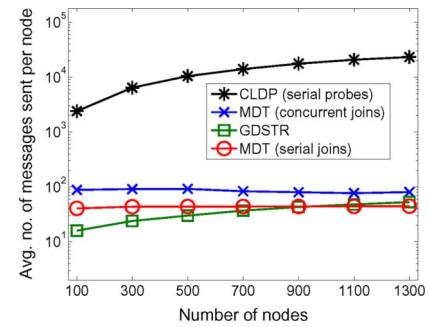


Fig. 10. Initialization message cost versus  $N$  (average node degree = 12).

GRG in our simulator. Several techniques to improve the performance of GRG are presented for *unit ball graphs* [8]. 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 2-D convex hulls to aggregate the locations of all descendants in the subtree rooted at the node; the two 2-D convex hulls approximate a 3-D 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 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 nongeographic routing protocol, VRR [5], is included in the comparison. We implemented VRR for static networks without joins and failures.<sup>11</sup> 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 Fig. 11(b) and (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 forwarding).

In our experiments, the number  $N$  of nodes is varied from 100 to 1300. The physical space and large obstacles are the same as

<sup>11</sup>With reference from <http://www.cs.berkeley.edu/~mccaezar/vrccode>.

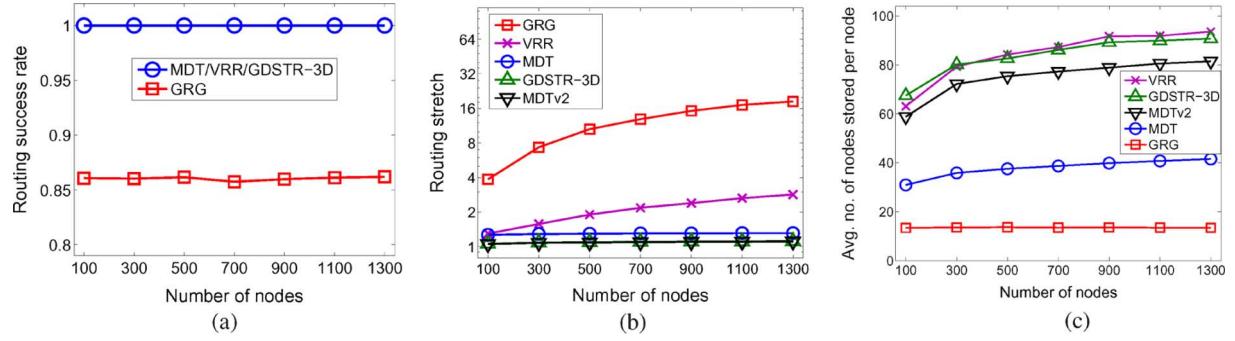


Fig. 11. Performance comparison of 3-D protocols (average node degree = 13.5). (a) Routing success rate versus  $N$ . (b) Routing stretch versus  $N$ . (c) Storage cost versus  $N$ .

the ones used in Fig. 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.

Fig. 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%. Fig. 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 Fig. 11(b)]. MDT, which uses 1-hop greedy forwarding, ranks a close third.

In Fig. 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 that 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 3-D 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 ( $d > 3$ ).

## VII. CONCLUSION

MDT is the only geographic routing protocol that provides guaranteed delivery in 2-D, 3-D, 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 multihop 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 multihop DT are fairly low and independent of  $N$ .

## APPENDIX

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

*Proof:*

- 1) By definition, a correct multihop 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 multihop 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 II is true, then a node closest to  $\ell$  is found. If the conditions in lines 1–3 are all false, node  $u$  performs greedy forwarding in lines 4 and 5. If it succeeds to find in  $P_u$  a physical neighbor  $v$  that is closer to  $\ell$  than node  $u$ , message  $m$  is transmitted directly to  $v$  (line 4 in Table II); else, greedy forwarding is performed over the set of DT neighbors (line 5 in Table II). The proof of [16, Theorem 1] for a distributed DT guarantees that either node  $u$  is closest to  $\ell$  or there exists in  $N_u$  a node  $v$  that is closer to  $\ell$  than  $u$ . Therefore, if node  $u$  is not a closest node to  $\ell$ , executing the *greedy forwarding code* (lines 4 and 5 in Table II) finds a node  $v$  that is closer to  $\ell$  than node  $u$ .
- 4) Any other node in  $S$  that is closer to  $\ell$  than  $u$  will not use the actions in lines 4 and 5 in Table II 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  $\ell$  than  $u$ . In this case, the condition of line 3 in Table II must be true for  $m$  at node  $u$ . Thus, node  $u$  executes the greedy forwarding code (lines 4 and 5 in Table II) 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  $\ell$  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 multihop 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 multihop 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 multihop 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 [16, Lemma 9], 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: 1) the path from the joining node to its closest DT node ( $z$ ) has a finite number of hops (by Theorem 1); and 2) 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 multihop DT is correct. ■

In the above proof, we make use of [16, Lemma 9] for a distributed DT in which every node can directly communicate with every other node. Let  $S'$  denote  $S \cup \{w\}$ . The main ideas used in the proof of this lemma are the following. 1) The existing node  $z \in S$  closest to the joining node  $w$  is a neighbor of  $w$  in  $DT(S')$ . 2) For any two neighbors of  $w$  in  $DT(S')$ , say  $u$  and  $v$ , if the facet shared by the Voronoi cells of  $u$  and  $w$  is adjacent to the facet shared by the Voronoi cells of  $v$  and  $w$  in  $DT(S')$ , then  $u$  and  $v$  are neighbors in  $DT(S)$ . Therefore, having found at least one neighbor in  $DT(S')$ ,  $w$  can find any other neighbor in  $DT(S')$  by following a sequence of existing edges in  $DT(S)$ . A detailed proof is presented in [16].

We next describe the MDT leave and failure protocols designed for handling a single leave and failure, respectively. Concurrent join, leave, and failure occurrences in different parts of a large network are often independent of each other. In this case, after a leave or failure, node states can be quickly and effectively repaired by these leave and failure protocols without waiting for the maintenance protocol to run upon timeout.

**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 ([16, Lemma 10]). 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  $DT(N_u)$ <sup>12</sup>;
- 2) a graph  $G = \langle V, E \rangle$ , where the set of vertices  $V = N_u$ , and the set of edges,  $E = \{(x, y) | x, y \text{ are neighbors in } DT(N_u) \text{ and } F_u \text{ does not contain a tuple with } x \text{ and } y \text{ as endpoints}\}$ .

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 multihop 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 multihop 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 our technical report [15].

**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. (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  $u$ .

## ACKNOWLEDGMENT

The authors thank B. Leong, their ACM SIGMETRICS shepherd, Q. Lv, and the IEEE/ACM TRANSACTIONS ON NETWORKING reviewers for their constructive comments and suggestions. They also thank J. Kim for programming help.

## REFERENCES

- [1] S. M. N. Alam and Z. J. Haas, "Coverage and connectivity in three-dimensional networks," in *Proc. ACM MobiCom*, 2006, pp. 346–357.

<sup>12</sup>Note that  $u$  is not in  $N_u$ .

- [2] P. Bose, A. Brodnik, S. Carlsson, E. Demaine, R. Fleischer, A. Lopez-Ortiz, P. Morin, and J. Munro, "Online routing in convex subdivisions," *Int. J. Comput. Geom. Appl.*, vol. 12, no. 4, pp. 283–295, 2002.
- [3] P. Bose and P. Morin, "Online routing in triangulations," *SIAM J. Comput.*, vol. 33, no. 4, pp. 937–951, 2004.
- [4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with Guaranteed delivery in Ad Hoc wireless networks," in *Proc. DIALM*, 1999, pp. 48–55.
- [5] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual ring routing: Networking routing inspired by DHTs," in *Proc. ACM SIGCOMM*, 2006, pp. 351–362.
- [6] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. ACM MobiCom*, 2004, pp. 114–128.
- [7] S. Durocher, D. Kirkpatrick, and L. Narayanan, "On routing with guaranteed delivery in three-dimensional ad hoc wireless networks," in *Proc. ICDCN*, 2008, pp. 546–557.
- [8] R. Flury and R. Wattenhofer, "Randomized 3D geographic routing," in *Proc. IEEE INFOCOM*, 2008, pp. 834–842.
- [9] S. Fortune, J. E. Goodman, and J. O'Rourke, Eds., "Voronoi diagrams and Delaunay triangulations," in *Handbook of Discrete and Computational Geometry*, 2nd ed. Boca Raton, FL: CRC Press, 2004.
- [10] K. R. Gabriel and R. R. Sokal, "A new statistical approach to geographic variation analysis," *Syst. Zool.*, vol. 18, no. 3, pp. 259–278, 1969.
- [11] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Geometric spanner for routing in mobile networks," in *Proc. MobiHoc*, 2001, pp. 45–55.
- [12] J. Herzen, C. Westphal, and P. Thiran, "Scalable routing easy as PIE: A practical isometric embedding protocol," in *Proc. IEEE ICNP*, 2011, pp. 49–58.
- [13] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. ACM MobiCom*, 2000, pp. 243–254.
- [14] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, "Geographic routing made practical," in *Proc. USENIX NSDI*, 2005, vol. 2, pp. 217–230.
- [15] S. S. Lam and C. Qian, "Geographic routing in  $d$ -dimensional spaces with guaranteed delivery and low stretch," Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, Tech. Rep. TR-10-03, Jan. 2010, (revised, March 18, 2011).
- [16] D.-Y. Lee and S. S. Lam, "Protocol design for dynamic Delaunay triangulation," Dept. Comput. Sci., Univ. Texas Austin, Tech. Rep. TR-06-48, Dec. 2006.
- [17] D.-Y. Lee and S. S. Lam, "Efficient and accurate protocols for distributed Delaunay triangulation under churn," in *Proc. IEEE ICNP*, Nov. 2008, pp. 124–136.
- [18] B. Leong, B. Liskov, and R. Morris, "Geographic routing without planarization," in *Proc. USENIX NSDI*, 2006, vol. 3, p. 25.
- [19] M. Li and Y. Liu, "Rendered path: Range-free localization in anisotropic sensor networks with holes," in *Proc. ACM MobiCom*, 2007, pp. 51–62.
- [20] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang, "Localized Delaunay triangulation with application in ad hoc wireless networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 10, pp. 1035–1047, Oct. 2003.
- [21] C. Liu and J. Wu, "Efficient geometric routing in three dimensional ad hoc networks," in *Proc. IEEE INFOCOM*, 2009, pp. 2751–2755.
- [22] D. Pompili, T. Melodia, and I. Akyildiz, "Routing algorithms for delay-insensitive and delay-sensitive applications in underwater sensor networks," in *Proc. 12th MobiCom*, 2006, pp. 298–309.
- [23] C. Qian and S. S. Lam, "Greedy distance vector routing," in *Proc. IEEE ICDCS*, Jun. 2011, pp. 857–868.
- [24] C. Qian and S. S. Lam, "ROME: Routing on metropolitan-scale Ethernet," in *Proc. IEEE ICNP*, 2012, to be published.
- [25] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu, "Greedy routing with guaranteed delivery using Ricci flows," in *Proc. IPSN*, 2009, pp. 121–132.
- [26] G. Toussaint, "The relative neighborhood graph of a finite planar set," *Pattern Recogn.*, vol. 12, no. 4, pp. 261–268, 1980.
- [27] P.-J. Wan, C.-W. Yi, L. Wang, F. Yao, and X. Jia, "Asymptotic critical transmission radius for greedy forward routing in wireless ad hoc networks," *IEEE Trans. Commun.*, vol. 57, no. 5, pp. 1433–1443, May 2009.
- [28] Y. Wang, C.-W. Yi, and F. Li, "Delivery guarantee of greedy routing in three dimensional wireless networks," in *Proc. WASA*, 2008, pp. 4–16.
- [29] S. Xia, X. Yin, H. Wu, M. Jin, and X. D. Gu, "Deterministic greedy routing with guaranteed delivery in 3D wireless sensor networks," in *Proc. MobiHoc*, 2011, Article no. 1.
- [30] G. Xing, C. Lu, R. Pless, and Q. Huang, "On greedy geographic routing algorithms in sensing-covered networks," in *Proc. ACM MobiHoc*, 2004, pp. 31–42.
- [31] J. Zhou, Y. Chen, B. Leong, and P. Sundaramoorthy, "Practical 3D geographic routing for wireless sensor networks," in *Proc. SenSys*, Nov. 2010, pp. 337–350.



**Simon S. Lam** (M'69–SM'80–F'85) received the B.S.E.E. degree with Distinction from Washington State University, Pullman, in 1969, and the M.S. and Ph.D. degrees in engineering from the University of California, Los Angeles (UCLA), in 1970 and 1974, respectively.

From 1971 to 1974, he was a Postgraduate Research Engineer with the ARPA Network Measurement Center, UCLA, where he worked on satellite and radio packet switching networks. From 1974 to 1977, he was a Research Staff Member with the IBM

T. J. Watson Research Center, Yorktown Heights, NY. Since 1977, he has been with the faculty of the University of Texas at Austin, where he is Professor and Regents Chair in computer science and served as Department Chair from 1992 to 1994.

Prof. Lam is a Member of the National Academy of Engineering and a Fellow of the ACM. He served as Editor-in-Chief of the IEEE/ACM TRANSACTIONS ON NETWORKING from 1995 to 1999. He served on the Editorial Boards of the IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON COMMUNICATIONS, PROCEEDINGS OF THE IEEE, Computer Networks, and Performance Evaluation. He cofounded the ACM SIGCOMM conference in 1983 and the IEEE International Conference on Network Protocols in 1993. He received the 2004 ACM SIGCOMM Award for lifetime contribution to the field of communication networks, the 2004 ACM Software System Award for inventing secure sockets and prototyping the first secure sockets layer (named Secure Network Programming), the 2004 W. Wallace McDowell Award from the IEEE Computer Society, as well as the 1975 Leonard G. Abraham Prize and the 2001 William R. Bennett Prize from the IEEE Communications Society.



**Chen Qian** (S'08) received the B.Sc. degree in computer science and technology from Nanjing University, Nanjing, China, in 2006, and the M.Phil. degree in computer science and engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2008, and is currently pursuing the Ph.D. degree in computer science at the University of Texas at Austin.

He has published research papers in a number of conferences and journals including ACM SIGMETRICS, IEEE ICNP, IEEE ICDCS, IEEE PerCom, IEEE/ACM TRANSACTIONS ON NETWORKING, and IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. His research interests include computer networking, data-center networks and cloud computing, and scalable routing and multicast protocols.

Mr. Qian is a student member of the ACM. He is the recipient of the James C. Browne Outstanding Graduate Fellowship at the University of Texas at Austin.