Randomized 3D Geographic Routing

Roland Flury
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
rflury@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

***Abstract*—We reconsider the problem of geographic routing in wireless ad hoc networks. We are interested in local, memoryless routing algorithms, i.e. each network node bases its routing decision solely on its local view of the network, nodes do not store any message state, and the message itself can only carry information about $O(1)$ nodes. In geographic routing schemes, each network node is assumed to know the coordinates of itself and all adjacent nodes, and each message carries the coordinates of its target. Whereas many of the aspects of geographic routing have already been solved for 2D networks, little is known about higher-dimensional networks. It has been shown only recently that there is in fact no local memoryless routing algorithm for 3D networks that delivers messages *deterministically*.**

In this paper, we show that a cubic routing stretch constitutes a lower bound for any local memoryless routing algorithm, and propose and analyze several *randomized* geographic routing algorithms which work well for 3D network topologies. For unit ball graphs, we present a technique to locally capture the surface of holes in the network, which leads to 3D routing algorithms similar to the greedy-face-greedy approach for 2D networks.

I. Introduction

Sensor networks and wireless mesh networks in general have received a lot of attention lately, last but not least because of their countless applications. In this paper, we consider one of the most fundamental building blocks of such networks: the exchange of information between network nodes. In general, the action of sending a message from a sender node to a target node is driven by a routing algorithm which guides the message through the network. Being such an integral part of any network, there already exists a large diversity of routing algorithms, including the IP routing of today's Internet, communication protocols that connect robots exploring our solar system, and algorithms that ensure message delivery in ad hoc networks. The different needs and characteristics of the various networks impose nearly countless challenges, requiring appropriate routing techniques. This work examines routing algorithms for large ad hoc networks (such as sensor and mobile ad hoc networks). In particular, we are interested in routing algorithms for three-dimensional networks.

In contrast to the IP based Internet routing, which is based on large forwarding tables, a routing algorithm for an ad hoc network faces not only the problem of an unstable network, but also rather limited network participants. The instability of the network may be caused due to mobility of the network nodes, or just by fluctuations of the wireless communication medium, which is far more vulnerable than a wired network. The limitations on the network nodes are various, including hardware constraints such as small memory and low processing power, as well as the limited power supply if the device runs on battery.

Exploiting the geometry of the network to perform routing is a prominent approach to overcome the challenges posed by such limited ad hoc networks. Geographic routing protocols forward the packet to a neighbor geographically closer to the target, until the message reaches its destination. Thus, a requirement for geographic routing is that each node knows its own, as well as its neighbors' Euclidean coordinates. A node can learn its position through hardware support such as GPS. Alternatively, the position can be obtained through localization algorithms, of which a variety has been proposed in recent years, e.g. [9], [20], [26]. Furthermore, the position of the target node needs to be known, as each routing step is based on this information. Because learning the position of the target node may come at a certain cost, the sender node includes this information in the message for reuse in further routing decisions. The request for the position of a given network node is handled by a *location service*, which has been broadly studied for static ad hoc networks [1], [11], and mobile networks [17], [19].

We define a *geographic routing algorithm* to base its decision solely on the position of the current node, the neighbors, and the destination, and we require the network nodes to be *memoryless*, i.e. not store any state for messages they see. This not only binds the routing state uniquely to the messages, but also removes an additional storage overhead from the nodes, which could limit the number of messages forwarded by a node if its memory is too small. As a matter of fact, the size of the memory is not the largest challenge. The problem of storing message state is that this data arrives dynamically, and it is hard to predict how much of this data needs to be stored at any given time. Dynamic memory allocation would solve the problem, but introduces an overhead that many devices cannot afford. For instance TinyOS, an operating system running on many types of sensor nodes, does not provide dynamic memory allocation. Consequently, the number of messages for which a node may store the state needs to be determined at compile time, jeopardizing routing success if more messages than anticipated need to be handled.

Another important property of geographic routing algorithms is that their decisions are only based on *local* information, which can easily be refreshed upon changes in the network. This stands in sharp contrast to routing algorithms that

IEEE Infocom 2008, Phoenix, USA.

rely in some way on a global view of the network. Whereas these global routing schemes provide excellent routing paths, the construction of their routing information is rather expensive, and any change of the network may require a complete, network wide reconfiguration of the routing information. As a result, these routing algorithms are an excellent choice for *static* networks, but not for (wireless) ad hoc networks, where a continuous change of the network topology is unavoidable.

A key concept of geographic routing is *greedy forwarding*, where each node along a routing path forwards the message to the neighbor that is geographically closest to the target. This constitutes a very simple, yet efficient way of routing messages. Greedy routing, however, is not always successful in delivering the packet. When a packet reaches a node, whose neighbors are all further away from the target than the node itself, greedy forwarding fails, and we say that the message has reached a local minimum. Such local minima are especially common in sparse networks and in networks with holes, i.e. regions in the network where no network nodes are placed, and around which a message needs to be led. For 2-dimensional networks, *face routing* and several variants thereof are the most prominent solutions to escape local minima. In the greedy-face-greedy approach [8], [12], a message routes greedily until it gets stuck in a local minimum. It then routes along the face of the network hole until it finds a node closer to the target than the local minimum, from where it continues greedily. A worst case optimal, but still average case efficient routing algorithm was obtained by constraining the range of the face routing in [23], [24], and techniques to proactively avoid routing voids are presented in [15]. In all of these protocols, the detection of the face requires a *planarized* network graph, e.g. a Gabriel Graph, which can be constructed locally in 2D.

We are interested in the question of whether we can adapt this geographic routing approach to networks in 3D. Clearly, we can still greedily forward messages towards their destination, which is likely to work for dense networks. But the recovery from local minima becomes more challenging, as the faces surrounding the network hole now expand in two dimensions, and are much harder to capture. We show in Section VI-A that it is indeed possible to describe the surface of network holes, using only local information. Our approach is quite different from 2D, as there is no equivalent to the planarization of a graph in 3D. Furthermore, we used the right/left hand rule in 2D to route along the 1-dimensional face, but there is no simple analogon in 3D to visit all nodes delimiting the surface of a hole. In fact, Durocher et al. have proven that there is no deterministic local routing algorithm for 3D networks that guarantees the delivery of messages [14]. As an immediate consequence, there is neither a local memoryless algorithm to traverse all nodes on a given surface in a deterministic manner.

The proof in [14] consists of two parts. First, the authors show that the existence of a *k-local*[1] geographic routing algorithm for UBG implies the existence of a 1-local geographic routing algorithm for any connected graph. In the second part, they show that any deterministic 1-local routing algorithm can be defeated. In fact, even very simple graph structures do not allow for deterministic routing algorithms.

In this paper, we present the first memoryless and local geographic routing protocols for three-dimensional networks and compare them to other routing techniques. Unlike the deterministic greedy-face-greedy solutions in 2D, our approach applies a randomized recovery to lead messages out of local minima.

A. Related Work

Routing algorithms for ad hoc networks can be roughly classified as proactive or reactive. Reactive routing schemes determine the route only on demand using flooding [28] to find a path to the destination. Whereas this approach does not generate a static overhead due to changes in the network topology, it introduces an excessive cost for route discoveries.

In proactive routing schemes, on the other hand, routes are determined ahead of time and stored in routing tables on the nodes. They are efficient only if the network is *stable* for a long time, as topology changes may require a network-wide reconfiguration. The probably most prominent members in this class constitute the compact routing schemes, which guarantee routes of nearly optimal length with moderate sized routing tables of polylogarithmic size in the number of network nodes [2], [31]. Compact routing schemes nearly always go along with a node labeling, i.e. each node is assigned a label. Just as with our geographic routing scheme, where the sender needs to determine the position of the destination node, compact routing requires the sender to learn the label of the target node, which is an integral part of the routing algorithm. In fact, any routing algorithm that does not label the nodes and desires a routing stretch below 3 needs routing tables of $\Omega(n)$ bits per node [18].

Whereas compact routing schemes try to minimize the size of routing tables, geographic routing does not need them at all, as messages are forwarded based only on local position information. Kleinberg showed in [21] that it is possible to assign to each network node a virtual coordinate in the hyperbolic plane (the label), and perform *greedy routing* with respect to these virtual coordinates, not needing any routing tables at all. However, the construction of the virtual coordinates is based on a (non-local) spanning tree, introducing a worst case stretch of $\Omega(n)$. In addition, the virtual coordinates need to be reevaluated upon any change in the network, which makes the scheme impractical.

The non-existence of local, memoryless routing algorithms that deliver messages deterministically [14] has many direct and indirect consequences. Whereas it is possible to deterministically traverse a planar subdivision and report all nodes and faces [7], there is no corresponding algorithm in 3D. However, it has been shown in [13] that for *any* undirected graph, it is possible to assign each node a local ordering of its edges such that a routing algorithm can visit all nodes in $\mathrm{O}(n)$

[1] A *k-local* routing algorithm can base its routing decision on a k-neighborhood of the current node.

time (deterministically!) by leaving a node through the edge succeeding the edge through which it entered. Unfortunately, the construction of the local edge-orderings requires a global view of the graph and has construction time cubic in the number of nodes.

B. Random Walks

The greedy geographic routing we would like to apply to our 3D graphs is actually close to optimal - as long as the message does not fall into a local minimum. But because there is no deterministic local memoryless routing algorithm [14], there is also no deterministic recovery algorithm that could lead our messages out of local minima. In this section, we take a short excursion to random walks, which we propose to use to escape from such local minima.

Whereas a message moving around randomly in our network may seem very inefficient and too simplistic, there is quite some work in this area indicating that random walks need not be as bad as it looks at a first glance. The two prominent models to capture a random walk on a graph $G = (V, E)$ are (1) the Markov chain, and (2) the flow in an electrical network obtained from G by replacing every edge by a resistance of 1Ω. See [25] for a survey of the topic. In the following, we use $n := |V|$ and $m := |E|$.

For our purpose, the hitting time H_{uv}, and the cover time C_G turn out to be most interesting. H_{uv} is the expected time until the random walk first visits vertex v when starting its walk at u, and C_G is the expected time needed to visit all nodes of G at least once. For arbitrary graphs, we have $C_G = \mathrm{O}(n \cdot m) = \mathrm{O}(n^3)$ [5], which poses also an upper bound on H_{uv}. The complete graph has optimal cover time $\Theta(n \log n)$, and the worst cover time of $\Theta(n^3)$ is obtained from the lollipop graph [16]. The 2D mesh has a non-optimal cover-time $\mathrm{O}(n \log^2 n)$, whereas the 3D mesh has optimal cover-time $\mathrm{O}(n \log n)$ [10]. Using the electrical resistance approach, the cover time of an arbitrary graph can be bounded to $C_G \leq 2mR_{span}$ [10], where R_{span} is the minimum resistance of a spanning tree in G. Since $R_{span} \leq n - 1$ for any graph, we obtain that $C_G \leq 2m(n-1)$. The hitting time H_{uv} seems to be intrinsically difficult to capture, but the somewhat related commute time κ_{uv}, the expected time to travel from u to v and back again, can be expressed by $H_{uv} < \kappa_{uv} = 2mR_{uv}$, where R_{uv} is the effective resistance between u and v [10].

A *random geometric graph* is obtained by placing n nodes uniformly at random in the unit square and connect two points if their distance is at most r. The minimal value of r such as to obtain a connected graph is subject of the percolation theory. It has been shown that if $r \geq r_{con} = \Theta\left(\sqrt{\log n / n}\right)$, the graph is connected w.h.p. [27]. Random geometric graphs with $r \geq \sqrt{8} r_{con}$ have optimal cover time of $\mathrm{O}(n \log n)$ [3]. Whereas this model seems appealing for wireless networks at the first moment, we need to keep in mind that the analysis only holds for $n \to \infty$. Furthermore, a connected random geometric graph requires so many network nodes that the graph tends to have no routing voids at all, causing no local minima to the greedy routing algorithm. In addition, wireless networks are obviously *not* random geometric graphs: There tend to be many holes in the network, where no nodes are deployed, which is ignored completely in this model.

II. Notation and Model

This section summarizes the notation used throughout the paper, and in the following section, we show a lower bound on the routing stretch of *any* local routing algorithm in 3D. The *stretch* of a routing algorithm $\mathcal{A}$ compares the length of routes found by $\mathcal{A}$ to the corresponding optimal routes. It is defined to be the maximal factor by which $\mathcal{A}$'s routes are longer than the corresponding optimal routes.

Whereas the lower bound of Section III applies for arbitrary graphs, we restrict our attention to unit ball graphs in Section V. Unit ball graphs (short UBG) are the 3D equivalent to the unit disk graphs in 2D, and constitute a basic model for wireless networks by assuming that any two network nodes are connected if their distance is below a certain threshold r_{max}, the maximal transmission radius[2]. W.l.o.g. we will assume that $r_{max} = 1$ unit. As usual, we describe the network as a graph $G = (V, E)$, where V is the set of network nodes, and E the set of connections between nodes. The number of nodes is denoted by $n := |V|$, and the number of edges by $m := |E|$. Furthermore, we use the notation $\mathcal{B}_r(v)$ to denote the ball of radius r around a given node v, and the set of neighbors of node v is abbreviated by $\mathcal{N}(v)$.

In Section VI, we present and discuss the first geographic routing algorithms for 3D. *Geographic routing* algorithms are *memoryless* in the sense that nodes store no message state, and they base their routing decision only on *local* knowledge. In previous work, geographic routing has been given various names, such as $\mathrm{O}(1)$-memory routing algorithm in [6], local routing algorithm in [22], or geometric ad hoc routing in [24].

III. Lower Bound

We start by deriving a lower bound for the performance of geographic routing algorithms in three dimensions. The following theorem states that any randomized geographic routing algorithm has at least a cubic stretch. (This lower bound would also hold for deterministic algorithms, which we know to not exist at all.)

Theorem 3.1: Let d be the length of the optimal path between a given source and destination in a 3-dimensional network. There are networks for which the route found by any randomized geographic routing algorithm has expected length $\Omega(d^3)$.

Proof: The proof idea is similar to the lower bound for 2-dimensions presented in [24]. We consider the following family of networks: For a positive integer r, construct a 3-dimensional graph as shown in Figure 1: First place nodes on the surface of a sphere with radius r such that the mutual distance between any two nodes is at least 2. Obtain a first set

[2]We are aware that a UBG is a very simplistic model for wireless networks, where the transmission range is far from circular. Our main routing techniques presented in this paper, however, are valid for real wireless networks.

of surface-nodes from $S_1 := \{(r \cdot \sin(2i \cdot \arcsin(1/r)), 0, r \cdot \cos(2i \cdot \arcsin(1/r))) \mid i \in [0, \lfloor 0.5\pi / \arcsin(1/r) \rfloor]\}$. In Figure 1, these nodes are drawn as solid squares on the left boundary of the sphere. The remaining surface points, also drawn as solid squares, are obtained from this initial set: For each $(x, y, z) \in S_1$, add $\{(x \cdot \sin(2i \cdot \arcsin(1/x)), x \cdot \cos(2i \cdot \arcsin(1/x)), z) \mid i \in [1, \lfloor \pi / \arcsin(1/x) \rfloor]\}$ to an initially empty set S_2. As a second step, add intermediate nodes on the surface (drawn as small diamonds) that connect nearby surface nodes. (The math is nearly the same as for the surface nodes and is omitted.) Furthermore, append to each surface-node a line of $\lfloor (r-1)/2 \rfloor$ nodes. The distance between nodes on the line is 1, and the line is directed towards the center of the sphere. These line-nodes are represented with round (red colored) dots in Figure 1. Finally, select an arbitrary surface node w, and append to its line further nodes until the center of the sphere, node t, is reached.

Note that there is no edge between nodes on different lines, as the lines are mounted on surface-points at least 2 units apart, and the length of the lines is less than $r/2$. Furthermore, the number of points per line is $\Theta(r)$. To determine the number of surface nodes, we use (*a*) $\alpha/2 < \arcsin(\alpha) < 2\alpha$, and (*b*) $sin(\alpha) > \alpha/2$ for $\alpha \in [0, 1]$. For a surface-node in S_1 with a given x-coordinate, the number of surface-nodes added to S_2 is $\lfloor \pi / \arcsin(1/x) \rfloor > x$ for $x \geq 1.4$, using (*a*). Thus, we can bound $|S_2|$ by summing up the values of the x-coordinates of the nodes in S_1 using (*a*) and (*b*):

$$|S_2| \geq 2 \cdot \sum_{i=1}^{\frac{1}{2}\left\lfloor \frac{\pi}{2 \arcsin(1/r)} \right\rfloor} r \cdot \sin(2i \cdot \arcsin \frac{1}{r}) \geq \sum_{i=1}^{r/2} i = \Theta(r^2) .$$

The total number of nodes in the graph is $(|S_1|+|S_2|) \cdot \Theta(r) = \Theta(r^3)$.

We now route from an arbitrary node s on the surface to node t in the center of the sphere. An optimal routing algorithm routes along the surface until it hits w and then follows the line until it reaches t. The path on the surface consists of at most 2.5 surroundings of the sphere, requiring $O(r)$ hops. The line contains at most r nodes to traverse, which results in a total cost of $O(r)$ hops for the optimal algorithm.

A geographic routing algorithm, on the other hand, needs to find the surface-node w on whose line-end the destination node t is located. Since only local routing information is available, this can only be achieved by exploring the lines by descending from the surface-nodes until t is found. For any randomized routing algorithms, the adversary can attach the line leading to t to a random surface-node, requiring the algorithm to explore $\Omega(r^2)$ lines until it finds t, which requires $\Omega(r^3)$ hops, which shows that the expected routing stretch is at least cubic. □

IV. Towards 3D Routing Algorithms

For our geographic routing algorithm, we use a *greedy-random-greedy* approach, short GRG, where the message is forwarded greedily until a local minimum is encountered. To resolve local minima, a randomized recovery algorithm kicks in. Unlike the deterministic face-routing in 2D, there is *no* deterministic recovery algorithm for 3D networks [14]. Therefore, our recovery technique is bound to be randomized.

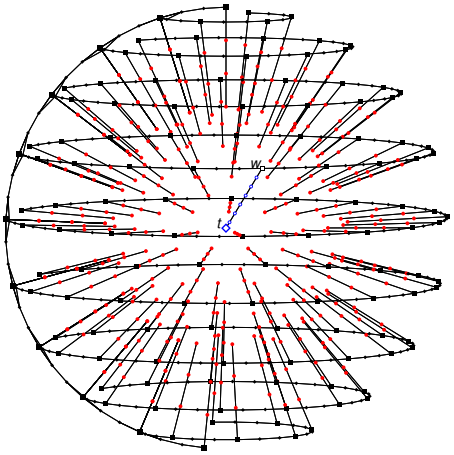

Fig. 1. Lower bound graph for geographic routing algorithms. Nodes represented by solid squares lie on the surface of a sphere with mutual distance at least 2. Nodes printed as diamonds lie also on the surface and connect these points. The round (red colored) nodes lie on lines leading from the surface-nodes towards the center. A single dedicated surface-node w has an extended line leading to node t in the center of the sphere.

Of course, the recovery part also needs to be memoryless and local, which immediately rules out flooding techniques[3] which could quickly find a node closer to the destination than the local minimum. We propose to use random walks (short RW), which constitute a very simple, yet surprisingly efficient recovery technique. We apply the following four techniques to ensure the performance of random walks:

(*1*) *Region Limited Random Walks:* When applying a RW to escape a local minimum at node u, the message is likely to explore large parts of the entire network until hitting a node v which is closer to the target than u, requiring $O(n \cdot m)$ hops. In most cases, however, such an extensive exploration is not needed: Let v be the node closet to u such that v is closer to the target than u, and let k be the length of the shortest path connecting u and v. Then, exploring $\mathcal{B}_k(u)$ with a RW would have been sufficient in order to find v. As the value of k is not known, the recovery algorithm performs an exponential search by limiting the RW in sequence to $\mathcal{B}_{2^i}(u)$ with $i \in \{2, 3, \dots\}$ until a node closer to the target is found. For each ball of radius r, the recovery algorithm performs $O(r^6)$ RW hops, which corresponds to the cover time for the sparse subgraph contained in the ball, bounding the recovery cost to $O(k^6)$ hops (see Theorem 6.1).

(*2*) *RW on the surface:* Similar to the face routing in 2D graphs, we can further restrict the RW to nodes delimiting the hole which causes the local minimum and which needs to be surrounded. Section VI-A describes how the nodes can *locally* determine the surface in a UBG, using the dual graph of Section V.

[3]Needless to say that we could implement a memoryless flooding algorithm where each node rebroadcasts a message whenever its TTL permits to do so, allowing multiple transmissions from the same node. However, such a scheme comes with an impractical overhead growing exponentially with the TTL.

(*3*) *Sparse Subgraph:* We have seen that for arbitrary graphs, $H_{uv} = 2mR_{uv} < 2mn$, and $C_G \leq 2mR_{span} < 2mn$. Both H_{uv} and C_G grow linearly in m and n, and we can improve H_{uv} and C_G by removing nodes in dense regions, and any edge that is not critical for the connectivity of the graph. We can achieve both points relatively easy by performing the RW on a *connected dominating set*, e.g. see [32], which implicitly also reduces the number of edges. Similarly, *topology control* algorithms build sparse subgraphs considering the network characteristics, and tend to be more stable [29], [33]. A truly sparse subgraph could be obtained by limiting the RW to an arbitrary *spanning tree*, reducing C_G and H_{uv} to $2n^2$. But as a spanning tree cannot be constructed locally, this approach is rather impractical for unstable networks. The *dual graph* presented in V is not only sparse, but also also fulfills the property that any $\mathcal{B}_r(u)$ contains at most $\mathrm{O}(r^3)$ dual nodes, limiting the cover time of a RW restricted to $\mathcal{B}_r(u)$ to $\mathrm{O}(r^6)$. (*4*) *Power of choice for RW:* The cover time of a RW can be improved by not returning to the previous node, if applicable. I.e. when the message is sent from n to m, and if n is not the only neighbor of m, then m forwards the message to a random neighbor, but not n. This improvement derives from the power of choice for RW [4].

V. Dual Graph

We now describe the construction of a dual graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ of $G = (V, E)$, on which our routing schemes are based. The position of the dual vertices (short *DV*) is bound to the intersection points of a regular cubic (imaginary) grid, covering the entire space. The *DV* are only placed nearby network nodes in order to populate the grid in regions where G is present. Each *DV* is *owned* by exactly one nearby network node, relating $\widetilde{G}$ to G. The relation from G to $\widetilde{G}$ is a bit more involved, as the relation is not bijective. In order to switch from a node $v \in V$ to $\widetilde{G}$, v chooses the *DV* d closest to v. We will see that d is owned by either v or a neighbor of v.

Edges in $\widetilde{G}$ are only present between direct neighbors in the underlying grid. The dual graph is defined such that the connectivity of G is preserved in $\widetilde{G}$. I.e. a path between $\widetilde{u}, \widetilde{v} \in \widetilde{V}$ in $\widetilde{G}$ implies a path in the original graph between *own*$(\widetilde{u})$ and *own*$(\widetilde{v})$, where *own*(x) denotes the owner of x. Similarly, a path between $u, v \in V$ in G implies a path between the corresponding *DV* in $\widetilde{G}$. As a result, we can perform a virtual routing on $\widetilde{G}$, and execute the corresponding routing steps on G. Finally, $\widetilde{G}$ is *sparse* in the sense that each $d \in \widetilde{V}$ has constant out-degree, and $\|\widetilde{V}\|$ is linear in the volume of $\bigcup_{v \in V} \mathcal{B}_1(v)$, the volume G occupies. In the remainder of this section, we show the following theorem:

Theorem 5.1: $\widetilde{G}$ is a sparse, connectivity preserving virtual graph of G, which can be constructed locally.

The construction of $\widetilde{G}$ consists of the following two steps. First, each node determines the set of *DV* it owns. Then, the edges $\widetilde{E}$ are added to $\widetilde{G}$ such that two *DV* are connected iff they are direct neighbors in the virtual grid: $(x_1, x_2) \in \widetilde{E} \Leftrightarrow \|\overline{x_1 x_2}\| = \eta$, where η denotes the cell-side-length of the grid, whose value we determine in section V-B. This step basically requires each node to determine all neighboring *DV* owned by other nodes. The construction of $\widetilde{G}$ is completely local, and each of the network nodes only knows a very limited local view of $\widetilde{G}$ at any time.

Algorithm 1: Construction of $\tilde{G}$ (Code for node v)

1 **Ownership Selection**
2 foreach(*DV* $d \in \mathcal{B}_\rho(v)$)
3 if($\nexists u \in \mathcal{N}(v)$ s.t. u.ID $< v$.ID$\wedge \|ud\| \leq \rho$)
4 v selects d as *RDV*
5 foreach($u \in \mathcal{N}(v)$)
6 $S := \{$*DV* $d \mid$ dist$(d, \overline{vu}) \leq h \ \wedge \ \|\overline{vd}\| < \|\overline{ud}\|\}$
7 remove from S all *DV* already known to be *RDV*
8 *TDV*$=$ *TDV* $\cup S$
9 send *TDV* to $\mathcal{N}(v)$
10 drop multi-owned *DV* if v.ID $>$ ID of other owner
11 **Connect**
12 send(*RDV* $\cup$ *TDV*) to 3-hop neighborhood of v
13 Determine the edges adjacent to any owned *DV*

A. Ownership Selection

The *DV* are positioned only on specific positions in space, defined by the intersection points of a regular cubic grid. The set of possible positions for a *DV* is given by $(i\eta, j\eta, k\eta) \mid i, j, k \in \mathbb{N}$. Our algorithm will ensure (locally!) that *at most one DV* is added to $\widetilde{V}$ for any of these positions.

The ownership selection algorithm executed by each node $v \in V$ determines for each node the set of *DV* v owns, see Algorithm 1. It consists of two substeps. First, v determines its *regular dual vertices* (short *RDV*), for which v can determine statically whether it owns them (lines 2–4). Node v chooses as *RDV* all *DV* which are most ρ away from v. In addition, for every selected *RDV* d, v may not have a neighbor u with a lower ID, whose distance to d is bounded by ρ. Formally, the set of *RDV* of node v is $\{d \mid d$ is a *DV* $\in \mathcal{B}_\rho(v) \ \wedge \ \nexists u \in \mathcal{N}(v)(u.\text{ID} < v.\text{ID} \wedge d \in \mathcal{B}_\rho(u))\}$. The exclusion of some nodes based on their ID is to ensure that each *DV* is owned by exactly one network node. In Figure 2, the sphere with radius ρ around v denotes the region where v selects its *RDV*. The value of ρ at least as large as to ensure that $\mathcal{B}_\rho(v)$ contains at least one *DV*, further information about ρ is given in section V-B. Please note that if a node $v \in V$ does not own any *RDV*, then the *DV* $\in \mathcal{B}_\rho(v)$ are owned by (direct) neighbors of v.

The second substep of the ownership selection ensures connectivity in the dual graph by adding additional *DV* to $\widetilde{G}$. In contrast to the first substep, there may be several nodes $v \in V$ that decide to own the same *DV*. The resolution of these conflicts is straight forward, but requires communication with the 1-hop neighborhood in G. Therefore, each node calls the *DV* selected in this substep *tentative dual vertices* (short *TDV*). To ensure connectivity, the *TDV* depend on the neighborhood $\mathcal{N}(v)$ of node v. For each neighbor u, node v determines all *DV* at most $h = \sqrt{3}\eta/2$ away from the line $\overline{vu}$ and closer to v than to u[4]. Then, v selects as *TDV* only the *DV* which it

[4]In case of equal distance, the node with smaller ID may own the *DV*.

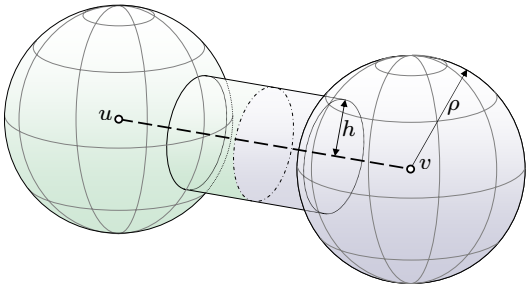

Fig. 2. Dual vertices that are at most ρ away from a network node $v \in V$ are added to the dual graph $\widetilde{G}$. The location of these *DV* is depicted by the balls of radius ρ around u and v. To ensure connectivity, the dual graph also contains all *DV* whose distance from any edge $(u, v) \in E$ is bounded by $h = \sqrt{3}\eta/2$, illustrated with the cylinder of radius h around the edge (u, v).

does not already own (as *RDV* or *TDV*), and are not already known to be regularly owned by a direct neighbor (lines 5–8). Figure 2 depicts the region where node v chooses its *TDV* with a cylinder enclosing the line $\overline{vu}$. For each *RDV* node v keeps, it also remembers the *reason*, i.e. the node u due to which the *RDV* was considered in the first place.

For the removal of multiple owners for the same *TDV* introduced in this second step, each node v sends its *TDV* list to its neighbors (line 9). Assume v selects *TDV* d due to its neighbor u, and also another node n choose d. Again, we resolve the conflict using the IDs, and let the lower ID win. Assume w.l.o.g. that $n.\text{ID} < v.\text{ID}$. Then, v needs to learn that it should drop its ownership of d. The value of η, and therefore the value of h, is chosen such that in such a situation, either (*a*) $n \in \mathcal{N}(v)$ or (*b*) $n \in \mathcal{N}(u)$, see Section V-B for more details. In case (*a*), v receives the *TDV* list from n directly and therefore learns about n's ownership of d. In the second case (*b*), u receives the *TDV* lists from both, v and n, and u can detect the conflict. u then determines whether u was v's *reason* for the conflicting *TDV* d, and if $n \notin \mathcal{N}(v)$. If (and only if) both conditions hold, u sends a *withdraw(d)* message to v, indicating that v should not own d.

After this second substep is completed, each *DV* (*RDV* and *TDV*) is sure to have exactly one owner $v \in V$. When the *DV* are connected as described in the introduction of this section, we are ready to state a first connectivity property of $\widetilde{G}$: If there is a path between two nodes $v_1, v_2 \in V$, there also exists a path between $d_1, d_2 \in \widetilde{V}$, where d_i is the closest *DV* to v_i. This property follows directly from Lemma 5.2, which is formulated for a single hop in G.

Lemma 5.2: Given two neighboring nodes $(v_1, v_2) \in E$ and two dual vertices $d_1, d_2 \in \widetilde{V}$ s.t. d_i is the closest *DV* to v_i, we can ensure that d_1 and d_2 are connected in $\widetilde{G}$ by adding to $\widetilde{V}$ all *DV* at most $h = \sqrt{3}\eta/2$ away from the line $\ell = \overline{v_1 v_2}$.

Proof: Because the side length of the grid is η, we know that the d_i themselves are at most h away from v_i. This forms the base-case of our inductive proof. As for the induction step, we consider a point P that moves along ℓ from v_1 to v_2, and a *DV* x which is at most h away from ℓ. Let Q be the position of P when P leaves $\mathcal{B}_h(x)$. We show that when P moves out of $\mathcal{B}_h(x)$, there is a *DV* x' whose distance to Q is strictly smaller than h, and that x' is connected to x in $\widetilde{G}$.

Let C be the cube of the virtual grid (which has side length η and x as one of its corners) that surrounds Q. If Q is exactly the center of C, all 8 corners of C have distance at most h from ℓ. Therefore, all of them are added to $\widetilde{G}$, and there is a path to each of them starting from x. Thus, when P moves from Q to Q', ϵ closer to v_2, at least one of the 7 remaining corners is in $\mathcal{B}_h(Q')$, which we pick as x'. If Q is not the center of C, there is a corner a of C neighboring x, whose distance to Q is strictly smaller than h. Therefore, a is added to $\widetilde{G}$ and is suitable as x'. □

B. Tuning the Dual Graph

Before we describe the construction of the edge set $\widetilde{E}$, which connects any two *DV* that are direct grid neighbors, we make a short detour and give some insight in how to determine the values of η and ρ. Remember that η is the side length of the virtual grid and therefore the minimal distance between any two *DV*, and ρ is the radius of the ball $\mathcal{B}_\rho(v)$ in which a node $v \in V$ searches for *RDV*. We select these two parameters such that for any edge $(d_1, d_2) \in \widetilde{E}$, there is a *short* path of at most 3 hops in G between $v_1 = \textit{own}(d_1)$ and $v_2 = \textit{own}(d_2)$, the corresponding owners. This will be a crucial property for routing algorithms simulated on $\widetilde{G}$, as they need to perform the actual routing hops on G.

We need to distinguish the following three cases: (***i***) d_1 and d_2 are both *RDV*, (***ii***) d_1 xor d_2 is *RDV*, and the other is *TDV*, and (***iii***) d_1 and d_2 are both *TDV*. From the ownership selection algorithm, we deduce that the maximal distance of a *RDV* from its owner is ρ.

In the first case (***i***), there should be an edge $(d_1, d_2) \in \widetilde{E}$ only if $(v_1, v_2) \in E$, requiring (***a***) $2\rho + \eta < 1$. Otherwise, it could happen that the path between v_1 and v_2 is of length $\mathrm{O}(|V|)$, which we wanted to avoid. For (***ii***), assume w.l.o.g. that d_1 is a *TDV* selected by v_1 due to its connection with r_1 (the reason for d_1), and that d_2 is a *RDV*. Then, the distance from v_2 to d_1 is at most $\rho+\eta$, and the distance from v_2 to $\overline{v_1 r_1}$ is bounded by $\rho+\eta+h$. By requiring (***b***) $\rho+\eta+h \leq \sqrt{3}/2$, we ensure that $v_2 \in \mathcal{B}_1(v_1) \cup \mathcal{B}_1(r_1)$, and that v_2 is a neighbor of either v_1 or r_1. As a result, we can route from v_1 to v_2 either directly, or via r_1. For (***iii***), where v_i choose d_i as *TDV* due to a node r_i, there should be an edge $(d_1, d_2) \in \widetilde{E}$ only if v_1 or r_1 is neighbor of at least one of v_2 or r_2: $(\mathcal{N}(v_1) \cup \mathcal{N}(r_1)) \cap (v_2 \cup r_2) \neq \emptyset$. Otherwise, it could again happen that the path between v_1 and v_2 requires $\mathrm{O}(|V|)$ hops. Assume the situation where $(\mathcal{N}(v_1) \cup \mathcal{N}(r_1)) \cap (v_2 \cup r_2) = \emptyset$. Then, d_2 can be placed closest to $\overline{v_1 r_1}$ if $\|\overline{v_1 r_1}\| = \|\overline{v_2 r_2}\| = 1$ and if v_2 and r_2 are placed on $\mathcal{B}_1(v_1) \cap \mathcal{B}_1(r_1)$. Then, the minimal distance between the two lines $\overline{v_1 r_1}$ and $\overline{v_2 r_2}$ is $1/\sqrt{2}$. As the d_i may be placed h away from their corresponding line, we need that (***c***) $1/\sqrt{2} - 2h > \eta$ s.t. $(d_1, d_2) \notin \widetilde{E}$. Thus, if $(d_1, d_2) \in \widetilde{E}$, this constraint ensures that there is a route from v_1 to v_2, either directly or via r_1 and r_2, requiring at most 3 hops.

Our goal is to maximize η such that $\|\widetilde{E}\|$ is as small as possible. Considering condition (c), we need to choose h as small as possible. In Section V-A, we have seen that $h = \sqrt{3}\eta/2$ satisfied our requirements. In fact, choosing h any smaller could break the connectivity of $\widetilde{G}$. Using (c), we deduce that $\eta < 1/(\sqrt{2}+\sqrt{6})$. As for ρ, we would like to choose its value as large as possible to maximize the number of statically determinable *RDV*. Constraint (a) yields $\boldsymbol{\rho}$**=0.37** for $\boldsymbol{\eta}$**=0.258**, which also satisfies condition (b). We have just shown the following lemma:

Lemma 5.3: Given two *DV* d_1, d_2 from the dual graph, and $v_i = own(d_i)$, the corresponding owners of the d_i. Then, if $(d_1, d_2) \in \widetilde{E}$, i.e. the distance between d_1 and d_2 is η, there is a path from v_1 to v_2 in G of at most 3 hops.

Proof: The lemma is satisfied by setting the values of η, ρ and h as described above. □

C. Connecting the Dual Graph

In the remainder of this section, we describe the construction of the edge set $\widetilde{E}$. The dual edges connect any two *DV* which are direct grid neighbors, which limits the outdegree of any *DV* to 6. For our local construction, this means that the owner $own(d) = v$ of a *DV* d needs to determine which of the 6 potential neighbors of d exist. For each existing neighbor n, v determines the owner $own(n)$, and the path over which $own(n)$ can be reached in G.

We know from Lemma 5.3 that the owner $own(n)$ is at most 3 hops away from v. Therefore, it is sufficient if each node broadcasts[5] the set of *DV* it owns to its 3-hop neighborhood (line 11 of Algorithm 1). Along the broadcasting path, every node that forwards the message can add its own ID, such that the receiver can easily determine the path (in G) to reach the sender. Please note that any node owns only $\mathrm{O}(1)$ *DV*, and therefore the storage overhead to remember the local view of the dual graph does not exceed the desired size.

Finally, we have seen all the pieces to state the proof of Theorem 5.1:

Proof of Theorem 5.1: From the ownership selection algorithm we deduce that the maximal distance between any *DV* and its owner is below 1. With other words, a network node $v \in V$ only owns *DV* that lie in $\mathcal{B}_1(v)$. Thus, any $u \in \widetilde{V}$ lies in $Q = \bigcup_{v \in V} \mathcal{B}_1(v)$. As the volume Q is composed of balls of radius 1, there are no arbitrarily thin areas in Q, and we can conclude that Q contains $\mathrm{O}(Q)$ *DV*. In combination with the fact that the outdegree of any *DV* is at most 6, this shows that $\widetilde{G}$ is sparse.

The connectivity of the dual graph $\widetilde{G}$ is an immediate consequence of Lemma 5.3 and Lemma 5.2, and an algorithm to construct $\widetilde{G}$ locally has been presented in this section. □

[5]Broadcasting from a node v to its 3-hop neighborhood can be implemented quite efficiently without retransmissions in the following way: The message contains a TTL counter initially set to 3 and a variable s storing the sender v. Upon reception of a message m at node u, u first decrements the TTL. Then, u rebroadcasts the message only if the following three conditions hold. $TTL > 0$, $m.s \neq u$, and $m.s \notin \mathcal{N}(u)$ if $TTL = 1$.

VI. Routing on the Dual Graph

Our simplest routing scheme, called `pureRW`, performs region limited random walks (see Section IV) until it finds the target. The balls delimiting the regions to explore are centered at the sender node, and the random walk applies the "power of choice" technique. Note that `pureRW` does *not* perform any greedy routing steps at all.

The RW selects its next hop based on the sparse dual graph $\widetilde{G}$, the selection algorithm is described in Algorithm 2. If the sender node s does not own any *DV* itself, it first sends the message to its neighbor which owns the *DV* closest to s. (The existence of such a neighbor is given, see Section V-A.) We use the following additional notation: For a node v, let $v.DV$ denote the set of *DV* owned by v, and $\widetilde{\mathcal{N}}(v)$ the set of *DV* which are neighboring a *DV* in $v.DV$, excluding the ones owned by v itself.

Algorithm 2: RW step (On node v, previous node p, $\mathcal{B}_r(s)$)

```
1 if(target ∈ (v ∪ N(v)))          v or one of its neighbors is the target
2   deliver the message and return
3 if(number of hops for this ball-size has been performed)
4   r = 2r                         Increase the ball delimiting the region of the RW
5 N := set of owners of Ñ(v) s.t. ∀n ∈ N : ||ns|| ≤ r
                                   Only select the owners which are in B_r(s)
6 if(N = {p})                      RW can only return to p, no choice
7   send message to p
8 else                             Apply "power of choice"
9   send message to random node in (N \ p)
```

The number of random hops for a ball of radius r is $\mathrm{O}(r^6)$, the cover time of the nodes contained in the ball. The routing stretch of `pureRW` can be bounded as following:

Theorem 6.1: Let S_{opt} be the stretch of the optimal geographic routing algorithm for 3D graphs. The expected stretch of `pureRW` is $\mathrm{O}({S_{opt}}^2)$.

Proof: For any source-target pair (s, t), let k be the length of the optimal route between s and t, and j the smallest integer s.t. $2^j \geq k$. `pureRW` performs a RW limited to $\mathcal{B}_{2^i}(s)$ for $i \in \{2, 3, \ldots, j\}$. For any ball of radius r, it performs $\mathrm{O}(r^6)$ random hops, which results in a total of $\mathrm{O}(k^6)$ random hops. As any optimal algorithm has cubic stretch requiring $\mathrm{O}(k^3)$ hops in the worst case (Theorem 3.1), the stretch of `pureRW` is $\mathrm{O}({S_{opt}}^2)$. □

Clearly, `pureRW` is not a practical routing scheme as its expected delivery time is just as bad as its stretch. Therefore, we try to route much more optimistically using the GRG approach, at the cost of that we are unable to analytically express its performance.

The greedy routing step of the GRG routing scheme selects the *DV* in $\widetilde{\mathcal{N}}(v)$ closest to the target and forwards the message to its owner (line 3 of Algorithm 3). Initially, if the sender s of the message does not own any *DV*, it sends the message to its neighbor which owns the *DV* closest to s. The message is greedily forwarded until the target is found (line 2), or a local minimum is reached (line 5):

Algorithm 3: Dual Greedy Step (On node v)

```
1 if(target ∈ (v ∪ N(v)))          v or one of its neighbors is the target
2   deliver the message and return
3 select d ∈ (Ñ(v) ∪ v.DV) closest to the target
4 if(d is owned by v)              v is in a local minimum for the target
5   start recovery
6 else
7   send message to own(d)
```

The recovery algorithm chosen by the GRG is either the region limited RW described in Algorithm 2, or a region limited RW that is also bound to the surface of the hole which needs to be surrounded.

A. Routing on the Surface

We describe the surface of a hole in the network with the aid of the virtual 3D grid introduced in Section V. The surface $\mathcal{S}$ is described by a list of grid-cubes which delimit the boundary between the hole and the network. Each node has only its local view of $\mathcal{S}$, which it determines using the presence, respectively absence of *DV*.

When the greedy routing described in Algorithm 3 reaches a local minimum at node v (line 5), the *DV* d closest to the target is owned by v itself. Therefore, at least one of the 6 grid-neighbors of d is not present, otherwise, d would not be closest to the target. Let i be the non-present grid-neighbor which is closest to the target, i.e. i is an intersection point of the grid neighboring d, where no *DV* was placed. Then, the four grid-cubes delimited by the cube-edge $\overline{di}$ are part of the surface $\mathcal{S}$. The remaining grid-cubes describing v's view of $\mathcal{S}$ are obtained iteratively: Until $\mathcal{S}$ does not change anymore, v does the following for every grid-cube $q \in \mathcal{S}$: For every corner c of q for which v owns the corresponding *DV*, v determines $\{c_1, c_2, c_3\}$, the three corners adjacent to c on q. For each c_i, v adds to $\mathcal{S}$ the 4 grid-cubes delimited by the cube-edge $\overline{cc_i}$ *iff* no *DV* was placed on c_i. When the iteration stops, $\mathcal{S}$ contains the grid-cubes delimiting the surface around the local minimum, as seen by v.

Let $S\mathcal{N}(v)$ be the set of owners which own a *DV* lying on the corner of a grid-cube $q \in \mathcal{S}$, and exclude v from $S\mathcal{N}(v)$. This set describes the neighbors of v which also lie on the surface $\mathcal{S}$, and from which the RW picks an arbitrary node to forward the message to. If node v decides to forward the message to $u \in S\mathcal{N}(v)$, it needs to describe the surface to u. It does so by including each $q \in \mathcal{S}$ which has a corner owned by u. Upon receiving the message, u sets its initial view $\mathcal{S}$ of the surface to this subset and determines the remaining grid-cubes describing u's view of the surface by applying the iterative algorithm described above.

Thus, the description of the surface changes constantly, but remains strictly local. In fact, when v sends the surface description to u, it is possible that u determines yet more grid-cubes which touch *DV* of v as well. This can happen when the surface $\mathcal{S}$ touches v in two or more independent places, such that v cannot determine locally the relationship. In situations where u is the only node that knows that the surface bends back to v, we need to ensure that u sends its view of the surface to v from time to time. But this requires that we drop the 'power of choice" optimization presented in Section IV. Otherwise, we risk falling into an infinite loop, as the RW will not explore the entire surface.

VII. Simulation

In order to validate our geographic routing algorithms for 3D networks, we performed a series of simulations in Sinalgo [30], a Java-based simulation framework for testing and validating network algorithms. We chose a fairly large simulation area of $20 \times 20 \times 10$ units and deployed between 2000 and 40000 nodes to cover the range between very sparse and dense networks. In order to obtain more realistic networks, we first placed 100 randomly rotated and randomly positioned cuboids of $2 \times 2 \times 1$ units in the simulation area. The cuboids were areas where no node could be placed, and they enforced holes in the network, such that, especially for dense networks, the messages could not be forwarded greedily without surrounding any holes.

Sparse graphs tend to be heavily twisted, which challenges our GRG routing algorithms with many local minima. To account for this fact, we performed more simulations for sparse networks, which can also be seen by the accumulation of samples for small n in Figure 3. For each initial deployment of n nodes, we first connected the nodes to a UBG, and kept only the *giant component*, the largest connected part of the network.

For each network, we selected 5000 random sender/target node pairs (s, t) and sent a message from s to t using the following five recovery algorithms when the message got stuck in a local minimum: RW on the dual, RW on the surface, RW on the Graph, bounded DFS on a spanning tree, and a bounded flooding. All RW were limited to exponentially growing regions (see Section IV), and all but the RW on the surface implemented the power of choice technique. The bounded DFS on an arbitrary spanning tree is *not* a local algorithm and was chosen for comparison. In that algorithm, we first built a spanning tree, and then perform a DFS on the tree, where the maximal depth to explore the graph increases exponentially. Finally, we implemented a recovery algorithm that uses flooding to escape a local minimum. The flooding relies on a mark-bits to avoid repetitions of the message, and is thus *not* memoryless. The TTL of the flooding message was incremented exponentially to obtain an optimal search time.

Figure 3 compares the overhead (measured in routing hops) of the five recovery algorithms. For ease of interpretation, we plotted against the overhead of the flooding algorithm, such that the y-axis shows how much more overhead the other routing algorithms induced.

A first important observation is that limiting the RW to the surface of the hole does in fact not help at all. The reason is two-fold: First, unless the network is very dense, it tends to have a single huge face covering nearly the entire network. I.e. the holes in the network are nearly never completely closed

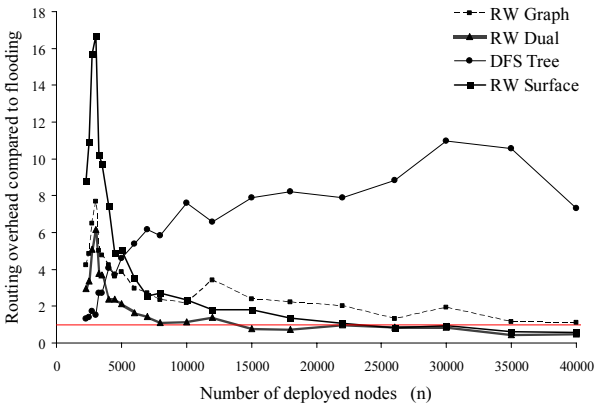

Fig. 3. The overhead of our routing techniques compared against a *non-memoryless* flooding algorithm. The x-axis indicates how many nodes were deployed, the y-axis how much more routing-messages our routing algorithms induced.

and most of them are interconnected over the surface. As a result, the restriction to the face does not reduce the number of nodes to visit. Secondly, we needed to drop the "power of choice" for the RW on the surface, which boosts the RW considerably, especially for sparse, tree-like networks. Thus, limiting the RW to the surface is not worth its price.

We can further observe that the overhead of the RW on the dual is below the overhead of the RW on the graph, which shows that we achieved our goals of obtaining a sparse network graph via the dual graph. The astonishing good performance of the DFS on the spanning tree for the sparse networks can be traced back to the fact that the nodes of these networks have very low degree, resulting in a tree-like network. For denser networks, however, the RW approaches perform much better. In particular, the RW on the dual and the RW on the surface perform even better than the flooding for very dense networks, as they operate on a sparser network.

Conclusions

Geographic routing schemes are both memoryless and local, which makes them highly suitable for mobile ad hoc networks and sensor networks. The geographic routing schemes for 2D networks, however, cannot be translated to 3D networks directly. For instance, limiting the recovery algorithm to visit only nodes on the surface of the network hole which caused the local minimum makes little sense in 3D, as most networks tend to have a single huge surface. Whereas the analysis of the surface detection was limited to UBG, the greedy-random-greedy routing scheme is applicable for real networks.

References

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: a Locality Aware Location Service for Mobile Ad Hoc Networks. In *DIALM-POMC*, pages 75–84, 2004.

[2] I. Abraham, C. Gavoille, A. Goldberg, and D. Malkhi. Routing in Networks with Low Doubling Dimension. In *ICDCS*, 2006.

[3] C. Avin and G. Ercal. On the Cover Time and Mixing Time of Random Geometric Graphs. *Theor. Comput. Sci.*, 380(1-2):2–22, 2007.

[4] C. Avin and B. Krishnamachari. The Power of Choice in Random Walks: An Empirical Study. In *MSWiM*, pages 219–228, 2006.

[5] G. Barnes and U. Feige. Short Random Walks on Graphs. In *STOC*, pages 728–737, 1993.

[6] P. Bose and P. Morin. Online Routing in Triangulations. In *ISAAC*, pages 113–122, 1999.

[7] P. Bose and P. Morin. An Improved Algorithm for Subdivision Traversal without Extra Storage. In *ISAAC*, pages 444–455, 2000.

[8] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing With Guaranteed Delivery in Ad Hoc Wireless Networks. In *DIAL-M*, pages 48–55, 1999.

[9] J. Bruck, J. Gao, and A. A. Jiang. Localization and Routing in Sensor Networks by Local Angle Information. In *MobiHoc*, pages 181–192, 2005.

[10] A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky. The Electrical Resistance of a Graph Captures its Commute and Cover Times. In *STOC*, pages 574–586, 1989.

[11] S. M. Das, H. Pucha, and Y. C. Hu. Performance Comparison of Scalable Location Services for Geographic Ad Hoc Routing. In *Infocom*, 2005.

[12] S. Datta, I. Stojmenovic, and J. Wu. Internal Node and Shortcut Based Routing with Guaranteed Delivery in Wireless Networks. *Cluster Computing*, 5(2):169–178, 2002.

[13] S. Dobrev, J. Jansson, K. Sadakane, and W.-K. Sung. Finding Short Right-Hand-on-the-Wall Walks in Graphs. In *SIROCCO*, pages 127–139, 2005.

[14] S. Durocher, D. Kirkpatrick, and L. Narayanan. On Routing with Guaranteed Delivery in Three-Dimensional Ad Hoc Wireless Networks. In *ICDCN*, pages 546–557, 2008.

[15] Q. Fang, J. Gao, and L. J. Guibas. Locating and Bypassing Routing Holes in Sensor Networks. In *INFOCOM*, 2004.

[16] U. Feige. A Tight Upper Bound on the Cover Time for Random Walks on Graphs. *Random Struct. Algorithms*, 6(1):51–54, 1995.

[17] R. Flury and R. Wattenhofer. MLS: An Efficient Location Service for Mobile Ad Hoc Networks. In *MobiHoc*, 2006.

[18] C. Gavoille and M. Gengler. Space-Efficiency for Routing Schemes of Stretch Factor Three. *J. Parallel Distrib. Comput.*, 61(5):679–687, 2001.

[19] M. Grossglauser and M. Vetterli. Locating Mobile Nodes With EASE: Learning Efficient Routes From Encounter Histories Alone. *IEEE/ACM Trans. Netw.*, 14(3):457–469, 2006.

[20] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-Free Localization Schemes for Large Scale Sensor Networks. In *MobiCom*, pages 81–95, 2003.

[21] R. Kleinberg. Geographic Routing Using Hyperbolic Space. In *Infocom*, 2007.

[22] E. Kranakis, H. Singh, and J. Urrutia. Compass Routing on Geometric Networks. In *Proc. 11 th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[23] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: of Theory and Practice. In *PODC*, pages 63–72, 2003.

[24] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically Optimal Geometric Mobile Ad Hoc Routing. In *DIAL-M*, 2002.

[25] L. Lovász. Random Walks on Graphs: A Survey. *Combinatorics*, 2:353–398, 1996.

[26] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust Distributed Network Localization With Noisy Range Measurements. In *SenSys*, pages 50–61, 2004.

[27] M. Penrose. The Longest Edge of the Random Minimal Spanning Tree. *Ann. Appl. Probab*, 7(2):340–361, 1997.

[28] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. *RFC3561*, 2003.

[29] P. Santi. *Topology Control in Wireless Ad Hoc and Sensor Networks*. Wiley, 2005.

[30] Sinalgo. *Simulator for Network Algorithms*. http://dcg.ethz.ch/projects/sinalgo, 2007.

[31] A. Slivkins. Distance Estimation and Object Location via Rings of Neighbors. In *PODC*, pages 41–50, 2005.

[32] P.-J. Wan, K. M. Alzoubi, and O. Frieder. Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Hetworks. *Mob. Netw. Appl.*, 9(2):141–149, 2004.

[33] R. Wattenhofer and A. Zollinger. XTC: A Practical Topology Control Algorithm for Ad-Hoc Networks. In *WMAN*, April 2004.