Geographic Routing without Location Information

Mobicom Submission # 475

Abstract— **For many years, scalable routing for wireless communication systems was a compelling but elusive goal. Recently, several routing algorithms that exploit geographic information (*e.g.,* GPSR) have been proposed to achieve this goal. These algorithms refer to nodes by their location, not address, and use those coordinates to route greedily, when possible, towards the destination. However, there are many situations where location information is not available at the nodes, and so geographic methods cannot be used. In this paper we define a scalable coordinate-based routing algorithm that does not rely on location information, and thus can be used in a wide variety of ad hoc and sensornet environments.**

I. Introduction

The increasing size and use of wireless communication systems strengthens the need for scalable wireless routing algorithms. Standard Internet routing achieves scalability through address aggregation, in which each route announcement describes route information for many nodes simultaneously. This approach to scalability is not applicable to many wireless environments, such as ad hoc networks or sensornets, where the node identifiers of topologically and/or geographically close nodes may not be similar (*e.g.,* by sharing high-order bits).For these cases, two main scalable routing techniques have been proposed: on-demand routing and geographic routing.[1]

In its simplest incarnation, on-demand routing involves no periodic exchanges of route information but instead establishes routes when needed by flooding a route request to the network. This approach was first presented in [14], and a series of refinements and variations have been proposed [12], [13], [23]. These techniques work well for small and moderate sized systems, and for large systems with relatively stable routes and limited communication patterns with significant destination locality.[2] However, for large systems with bursty any-to-any communication patterns, the overhead (and latency) of route discovery can be significant [16].

Geographic routing uses nodes' locations as their addresses, and forwards packets (when possible) in a greedy manner towards the destination. The most widely known proposal is GPSR [17], but several other geographic routing schemes have been proposed [1], [2], [3], [6], [7], [9], [19], [20].[3] One of the key challenges in geographic routing is how to deal with dead-ends, where greedy routing fails because a node has no neighbor closer to the destination; a variety of methods (such as perimeter routing in GPSR) have been proposed for this. Geographic routing is scalable, as nodes only keep state on their neighbors, and supports a fully general any-to-any communication pattern without explicit route establishment. However, geographic routing requires that nodes know their location. While this is a natural assumption in some settings (*e.g.,* sensornet nodes with GPS devices), there are many settings where such location information isn't available.

In this paper we address how to retain the benefits of geographic routing in the absence of location information. This problem has been partially addressed in [5], but there they consider the case where some nodes don't have geographic information; here we focus mainly on the case where no (or only the perimeter) nodes have location information. Our approach involves assigning *virtual* coordinates to each node and then applying standard geographic routing over these coordinates. These virtual coordinates need not be accurate representations of the underlying geography but, in order to serve as a basis of routing, they must reflect the underlying connectivity. Thus, we construct these virtual coordinates

[1] An additional technique, distance vector routing, has been applied to ad hoc networks (see [15]), but each node must maintain routing state proportional to the number of nodes in the system. Since our focus is on scalable techniques (*i.e.,* those that can apply to extremely large systems), we do not consider distance vector techniques here.

[2] By destination locality we mean that nearby nodes are often sending to the same destination. This allows the overhead of route establishment to be shared among the many sources seeking to send to a particular destination.

[3] Some of these algorithms don't strictly use geographic coordinates, but instead use hop-count information [2]; there are other algorithms not listed here, such as [18], that use location information but not in the packet-forwarding process.

using only local connectivity information. Since local connectivity information is always available (nodes always know their neighbors), this technique can be applied in most settings. Moreover, as our simulation results show, there are scenarios, such as in the presence of obstacles, where greedy routing performs better using virtual coordinates than using true geographic coordinates.

The paper is organized as follows. We deal with some technical preliminaries in Section II. The key contribution of our paper, the construction of virtual coordinates, is presented in Section III. The performance of the algorithm is evaluated in Section IV and we conclude with a few remarks in Section V.

Before turning to our technical content, we first put our work in context. The vast literature on ad hoc routing contains many valuable proposals, each with their own niche of applicability. It is not our intent to compare them here, or to study study under which conditions each is most appropriate.[4] Instead, we narrow our focus to environments with the following characteristics: a very large number of nodes with a relatively high density, a very general communication pattern with many host pairs communicating, and a need for low-latency first-packet delivery. These characteristics require a routing algorithm that only keeps per-neighbor state (not per-node or per-route) and does not involve a route establishment phase. To our knowledge, only geographic routing algorithms meet these requirements. We don't claim to know exactly how large or how dense the system, or how general the communication pattern, or how tight the latency requirements, must be in order for geographic routing to be the most appropriate choice; that analysis will have to await further study. Moreover, geographic routing is known to have serious limitations, particularly with how to route around dead-ends and obstacles and how to function at very low densities. Fixing those problems is not our focus here. Instead, our goal here is simply to explore whether one can apply the geographic routing paradigm, with both its strengths and its weaknesses, to situations where no, or only a very few, nodes have location information.

[4]See [4], [11] for performance comparisons on relatively small systems.

II. Preliminaries

The core of this paper describes an algorithm for assigning virtual coordinates to nodes. For these coordinates to be useful, we must address two other issues: (1) routing in this virtual coordinate space, and (2) how routing in this space can be used in both traditional ad-hoc routing environment and data centric networks such as sensornets. While these pieces are not part of our research contribution, we present their designs here for completeness.

A. Routing Algorithm

There have been many geographic routing algorithms, such as [3], [6], [7], [9], [17], [19], [20] and others. Our purpose here is not to improve these algorithms, merely to provide a set of virtual coordinates over which they can operate. For the purposes of evaluation, we use a very simple routing algorithm. We assume that all nodes know their own coordinates, those of their neighbors, and those of their neighbor's neighbors; we call this set the *routing table*. This information is easily obtained by having neighboring nodes periodically exchange their coordinates, and their neighbor's coordinates.

Destinations in packets are virtual coordinates. Packets are routed according to three rules:

- *Greedy:* The packet is forwarded to the node in the routing table closest (in virtual coordinates) to the destination, if (and only if) that node is closer to the destination than the current node.
- *Stop:* If the node is closer to the destination than any other node in its routing table, and higher layers determine that the packet is indeed bound for this node, then the packet is considered to have arrived at its destination. For example, if the payload is an IP packet, the receiving node can check if its IP address matches that of the packet. In many environments, one can determine if the packet has arrived at the appropriate destination.[5]
- *Dead-end:* When a packet is not able to make greedy progress, nor has reached a stopping point, we say it has reached a *dead-end*. In this case, the node where the dead-end was reached performs an

[5]What we've presented here is a very primitive stopping condition; one can design much more sophisticated stopping conditions, but we do not do so here.

expanding ring search until a closer node is found (or a maximum TTL has been exceeded).[6]

B. Distributed Hash Table

Above we described how packets are routed over (real or virtual) coordinates. However, routing by itself is not sufficient to use the system. To make this point clear, we consider our two target environments, ad hoc networks and sensornets, separately.

Ad Hoc Networks The goal here is to reach an specific host (as specified by an address or other identifier). However, because geographic routing is based on the coordinates, not the identifier, one can't directly reach the intended target without knowing where that intended target is. Thus, geographic routing must be augmented with a service that can translate identifiers into locations. The GLS system [19] provides a scalable and elegant solution to this problem. Our approach is very similar in spirit, though differing in details. As described below, e implement a distributed hash table (DHT) on top of the routing system. In a DHT, each node uses the `put` operation to store its location using its identifier as the key. Communicating with a particular node requires using the `get` operation, with the identifier as key, to retrieve its current location.

Sensornets In sensornets, there is little reason to attempt to communicate with specific node based on their identifiers. Instead, one wants to describe the desired data directly. Such data-centric approaches are now seen as fundamental to sensornets [10], [8], [22]. Many of these approaches don't require any underlying routing functionality besides flooding. However, a recently developed extension of these ideas, data-centric storage, does require substantial routing support. Data-centric storage involves implementing a DHT in the sensornet and then storing notable events and data by name; for example, at a simplistic level, sightings of animals would be stored in the DHT using the animal name as the key. The initial incarnations of data-centric storage, such as [25] and [24], used geographic routing as the underlying routing algorithm. However, one could implement the hash table functionality directly on top of our routing algorithm, even though the coordinates are not related to the real geographic locations. Thus, both environments require a hash-table-like functionality. This can be easily implemented on top of our routing algorithm as follows. Whenever a `put` or `get` command is issued with a given *key*, that key is hashed into a coordinate and the packet is routed towards that coordinate. When the packet is *stopped* (*i.e.,* it has arrived at the node closest to its destination), it then either puts or gets the data, as required. There are many algorithmic extensions to achieve higher reliability (using several different hash functions, replicating data locally, etc.); some of these are described in [25].

We described these two pieces, routing and distributed hash-table, because they are necessary for the overall system. Our key contribution is the definition of virtual coordinates; we borrow from the literature for the routing and DHT components. Therefore, in our evaluations we want to evaluate the quality of these coordinates, not the quality of the various algorithmic extensions (such as our DHT construction) implemented on top of them. Consequently, we have chosen to focus mainly on how well one can route greedily over these coordinates, and to compare that with how well one could route greedily over the true geographic coordinates. There are many algorithms for dealing with dead-ends (*i.e.,* cases where greedy routing fails) for both the delivery of packets and for the construction of DHTs (see, *e.g.,* [17], [25]). We don't want the defects of our coordinate construction algorithm to be masked by these techniques, so we report on how often purely greedy routing reaches its intended destination. If our results are roughly comparable to the case where location information is known, then we have extended geographic routing to the realm where location information is not available.

III. Coordinate Construction

This section describes the main contribution of our paper: a method for constructing virtual coordinates without location information. To more clearly present the various pieces of our construction, we consider three scenarios with decreasing degrees of location information. These scenarios differ in how much information *perimeter* nodes know about their location; all other nodes are assumed to have no location information.[7] In what follows, we always consider the nodes to be lying in the plane; however, our techniques general-

[6] Expanding ring searches are also used in [6], [26] for similar reasons.

[7] Perimeter nodes are those nodes lying on the outer boundary of the system.

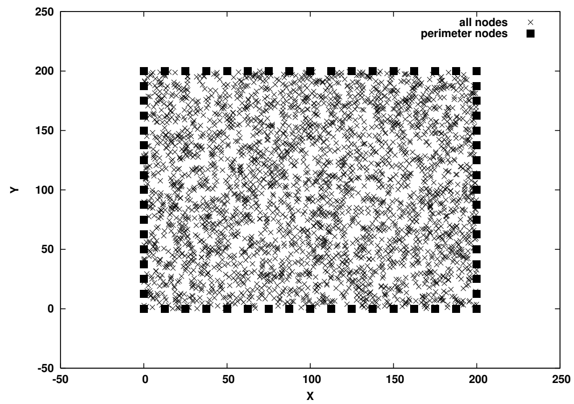


Fig. 1. A network with 3200 nodes. Perimeter nodes are represented by black squares. The radio range of each node is 8 units.

ize straightforwardly to higher dimensions (though we don't discuss that generalization here).

- Perimeter nodes know their location
- Perimeter nodes know that they are perimeter nodes, but don't know their location
- Nodes know neither their location, nor whether they are on perimeter

We present construction techniques for each of these scenarios; as the location information decreases the construction algorithm becomes increasingly complex.

To illustrate these algorithms we use a network consisting of 3200 nodes uniformly spread throughout a square area of 200×200 units (see Figure 1). There are 64 perimeter nodes (15 on each side and 4 in the corners). Each node has a radio range of 8 units, so nodes have on average around 16 neighbors. In all simulation results presented in this section we ignore packet losses and signal attenuation (aside from assigning a fixed radio range; *i.e.,* inside the range packets are received, and outside they aren't, and there is no probabilistic degradation in between). More details about our simulator, as well as more complex and realistic simulation scenarios are presented in Section IV.

A. Perimeter Nodes Know Location

We first consider the scenario where all perimeter nodes know their exact geographic coordinates. We describe a way in which all non-perimeter nodes can determine their coordinates through an iterative relaxation procedure. The analogy, borrowed from theory of graph embeddings, is that each *link* – each neighbor relation – is represented by a force that pulls the neighbors together [21]. We assume that the force in the x-direction is proportional to the difference in the x-coordinates (and the same for the y-coordinates). If we hold the neighbors fixed, then a node's equilibrium position (the one where the sum of the forces is zero) is where the its x-coordinate is the average of its neighbors' x-coordinates (and, again, the same for y-coordinate). We use these facts to motivate an iterative procedure where each non-perimeter node periodically updates its virtual coordinates as follows:

$$x_i = \frac{\sum_{k \in neighbor\_set(i)} x_k}{size\_of(neighbor\_set(i))} \quad (1)$$

$$y_i = \frac{\sum_{k \in neighbor\_set(i)} y_k}{size\_of(neighbor\_set(i))} \quad (2)$$

Consider the network in Figure 1. Figure 2 illustrates how the relaxation algorithm works when all non-perimeter nodes start with the same initial coordinates, in this case $(100, 100)$. The relaxation equations imply that non-perimeter nodes that have a perimeter node among its neighbors will "move" towards that perimeter node. As the iterations progress, nodes tend to move towards the perimeter nodes that are closest to them in terms of the number of hops. As shown in Figure 2(c), the algorithm eventually converges to a state in which the nodes are spread throughout the region, although the distribution has more "holes" than the set of true positions (compare Figure 1 with Figure 2(c).

We now evaluate how well one can route over these virtual coordinates by measuring two key metrics:

- *Success Rate:* This is the fraction of times packets reach their intended destination using purely greedy routing. This measurement, as we explained in Section I, is designed to evaluate the performance of the coordinate construction algorithm without interference from the various dead-end avoidance techniques that can be applied to both real and virtual coordinate routing algorithms.
- *Average Path Length:* This is the average number of hops taken along the path.

We pick node pairs at random and send a packet from to the other; in the simulations discussed here we chose 32000 such random pairs. We compare the results of routing over virtual coordinates to routing over the true geographic coordinates. For greedy routing with true geographic coordinates, for the scenario described above, the success rate is 0.989 and the average path length is 16.8. In contrast, with virtual coordinates, the routing success rate is 0.993 (better than when using true coordinates!) and the average path length is 17.1 (only slightly worse than using true coordinates).

While this performance is quite satisfactory, one concern is that it might take too many iterations to converge (*e.g.,* 1000 iteration in this example). In Section III-B we address this problem, by presenting a simple method of picking the the initial coordinates of the non-parameter nodes which reduces the convergence time to a few (just one in this test case) iterations.

The relaxation algorithm does not require all perimeter nodes to know their location. For example, Figure 3(a) shows the virtual coordinates of the nodes when only 8 perimeter nodes (out of 64) know their true coordinates. While the distribution of the virtual coordinates looks skewed in this case, the greedy routing performance is still very good: the success rate is 0.981, and the average path length is 17.3.

In addition, so long as the relative ordering of perimeter nodes is preserved, their locations need not be exact. For example, Figure 3(b) shows the virtual coordinates of nodes when the perimeter nodes are evenly spaced along a circle (preserving the order in which they appear on the real perimeter). In this cases, the routing success rate is 0.99, while the average path length is 17.1. As we will see in the next sections, the limited requirements imposed by this simple relaxation algorithm and its robustness to imperfect positioning information allow us to altogether eliminate the need for knowledge of the geographic coordinates of perimeter nodes.

B. Perimeter Nodes are Known

We now retain our assumption that perimeter nodes know that they are indeed on the perimeter but eliminate the assumption that these perimeter nodes know their exact geographic location. To do so, we preface the previous relaxation method with a phase where perimeter nodes compute their own virtual coordinates. Briefly, in this phase, perimeter nodes flood the network to discover the distances (in hops) between all perimeter nodes and then use a triangulation algorithm that computes perimeter positions from this inter-perimeter distance matrix.

Our perimeter coordinate algorithm consists of three stages.

Stage 1: Each perimeter node broadcasts a HELLO message to the entire network. This allows perimeter nodes to discover their distance (in hops) to all other perimeter nodes.[8] We call this vector of distances a node's *perimeter vector*.

Stage 2: Each perimeter node broadcasts its perimeter vector to the entire network. At the end of this stage, every perimeter node knows the distances between *every* pair of perimeter nodes.

Stage 3: Every perimeter node uses a triangulation algorithm to compute the coordinates of all other perimeter nodes (including its own coordinates). The coordinates are chosen such as to minimize

$$\sum_{i,j \in perimeter\_set} (measured\_dist(i,j) - dist(i,j))^2, \tag{3}$$

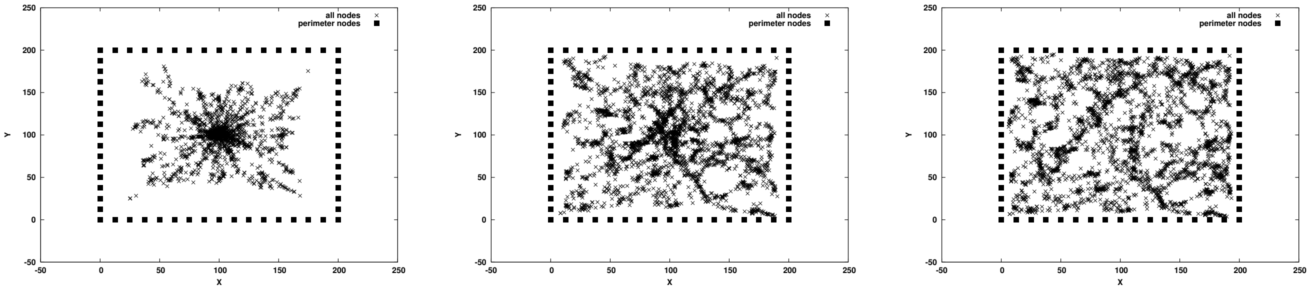
where *measured_dist(i, j)* denotes the distance between nodes i and j measured in Stage 1, and *dist(i, j)* represents the geometric distance between the virtual coordinates of nodes i and j. This can be seen as minimizing the potential energy when each perimeter node is a ball and they are attached to every other perimeter node by a spring whose length is proportional to the hop count distance.

At the end of Stage 3, each perimeter node knows its own (virtual) coordinates, and non-perimeter nodes can now use the relaxation algorithm described in Section III-A to compute their own virtual coordinates. This completes the preparatory phase.

Note that because perimeter beacons are flooded over the entire network, non-perimeter nodes also learn of the inter-perimeter distances and can hence run the same triangulation algorithm to compute reasonable initial coordinates rather than starting at the center of the virtual coordinate space.

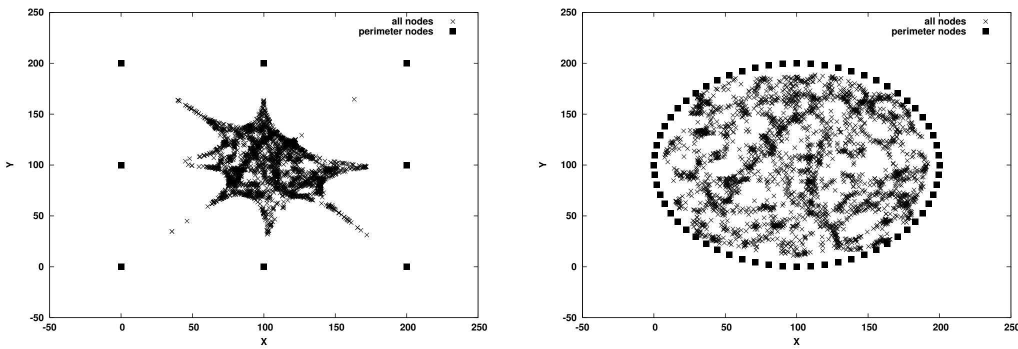
Figure 4 shows the virtual coordinates of the network

[8]This distance is computed by maintaining a distance counter in the HELLO message that is incremented at every hop. A node's distance to a flooding node is then merely the minimum counter value over all the messages it receives from it.



(a) (b) (c)

Fig. 2. The virtual coordinates of the non-perimeter nodes after (a) 10, (b) 100, and (c) 1000 iterations, respectively. The initial virtual coordinates of non-perimeter nodes are set to (100, 100).



(a) (b)

Fig. 3. The virtual coordinates of the non-perimeter nodes when (a) only 8 nodes on the perimeter know their geographic coordinates, and (b) when the perimeter nodes use inaccurate coordinates, but which preserve the order of the nodes on the perimeter.

studied in Section III-A, where nodes (both perimeter and non-perimeter) use triangulation to compute their coordinates. We see that the triangulation algorithm performs very well: the routing success rate and the average number of hops suffer virtually no degradation when compared to the case when perimeter nodes know their coordinates. Further, the initial conditions drastically reduce the convergence time for the relaxation algorithm. After only *one* iteration the routing success rate is as high as 0.992, while the average path length is 17.2. After ten iterations the success rate increases slightly to 0.994 while the average path length remains unchanged. For comparison, when all non-perimeter nodes start with the same initial coordinates, it takes 1000 iterations to obtain a success rate of 0.995.

In our description so far, in the above 3-stage algorithm each perimeter node uses complete knowledge of the inter-perimeter distances to independently compute its coordinates using triangulation. In practice, message loss and node failure can cause perimeter nodes to have incomplete (and inconsistent) knowledge of the inter-perimeter distances in which case the above triangulation algorithm would cause different perimeter nodes to compute inconsistent coordinates. This is because any set of coordinates the satisfies the minimization condition can be rotated, translated, and flipped while still satisfying the minimization condition. Thus, we need to *canonicalize* the computation so that all nodes performing it independently will arrive at the same solution. To address this problem we use the following technique.

Bootstrapping beacons Two designated bootstrap beacons[9] flood the network with HELLO messages. Perimeter nodes then include these bootstrap beacons in their regular triangulation computation and compute the coordinates of all the perimeter and bootstrap nodes. Every node the computes the center of gravity (CG)

[9]These two beacons nodes could be picked in a number of ways: for example, simply pre-configuring two specific node identifiers to act as beacons or relying on more complex probabilistic beacon election algorithms.

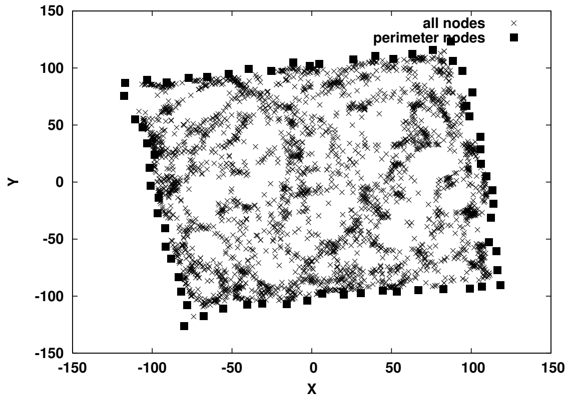


Fig. 4. The virtual coordinates of a network with 3200 nodes and with 64 perimeter nodes. The perimeter nodes use triangulation to compute their coordinates, while non-perimeter nodes use the relaxation algorithm to compute their coordinates.

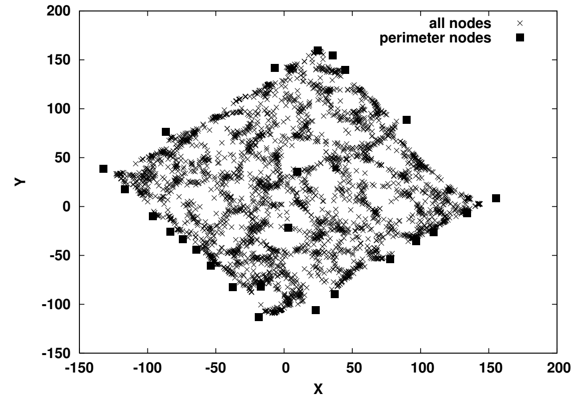


Fig. 5. The virtual coordinates of a network with 3200 nodes where no node knows its coordinates or whether it is on the perimeter.

of all these positions. The CG together with the positions of the two bootstrap nodes define a new coordinate axes – the CG forms the origin, the first bootstrap node defines the positive x axis and the second bootstrap node defines the positive y. All computed coordinates are then normalized with respect to these new axes. Note that CG is resilient to incomplete information. If a node lacks information about one of the bootstrap nodes it can afford not to act as a perimeter node since the relaxation does not require all perimeter nodes to be detected.

C. No Location Information

In the final scenarios, we relax the assumption that perimeter nodes know they are on the perimeter. We add to the algorithms described before another preparatory stage where a subset of nodes identify themselves as perimeter nodes. To achieve this we leverage one of the bootstrap beacon nodes described in the previous section. Recall that these bootstrap nodes broadcast HELLO messages to the entire network; hence, every node discovers its distance to these bootstrap nodes. Nodes use the following criteria, called *perimeter node criterion*, to decide if they are perimeter nodes: *if a node is the farthest away, among all its two-hop neighbors from the first bootstrap node, then the node decides that it is on the perimeter*.

To correctly maintain perimeter nodes under mobility, the first bootstrap node periodically rebroadcasts and nodes determine afresh whether they are perimeter nodes or not. Having determined perimeter nodes, the system embarks on the triangulation phase described in Section III-B, in which all perimeter nodes measure the distances between themselves, and use triangulation to compute their fixed coordinates.

Figure 5 shows the virtual coordinates of all nodes when nodes use their distance to the designated beacon to determine whether they are on the perimeter or not. We make two observations. First, there are two nodes that wrongly decide that they are on the perimeter, when they are not. However, these decisions have little effect on the virtual coordinates of the other nodes, since the triangulation algorithm correctly positions these nodes inside the network perimeter. Second, the routing success rate and the average path length remain excellent at 0.996 and 17.3, respectively, after only 10 iterations.

Finally, we add one additional mechanism that allows us to maintain a consistent well-known virtual coordinate space even in the face of node mobility and failures.

Coordinate projection on circle Once perimeter nodes compute their coordinates using triangulation they project these coordinates on a circle with origin at the center of gravity of the perimeter nodes and radius equal to the average distance of perimeter nodes from the center of gravity. There are two reasons for doing this. First, this gives us a well-defined area for implementing a distributed hash table (DHT).[10] Second, this

[10] Recall that in defining the DHT we needed a hash function to map keys onto the set of coordinates; knowing that the

virtual circle makes it easier to support mobility. When a new node concludes that it is on the perimeter that node will simply compute the projection of its current coordinates on the circle and assume these coordinates to be fixed. In turn, a node that is no longer on the perimeter "un-fixes" itself and starts to update its coordinates using the relaxation algorithm.

This completes our description of the algorithm. We now briefly recap our algorithm before moving on to its evaluation in Section IV.

D. Summary

Our coordinate assignment algorithm consists of the following key steps:

1) Two designated bootstrap beacon nodes broadcast to the entire network. Nodes use their distances to one of these bootstrap nodes to determine whether they are perimeter nodes.
2) Every perimeter node sends a broadcast message to the entire network to enable every other node to compute its perimeter vector, *i.e.,* the distances from that node to all perimeter nodes.
3) Perimeter and bootstrap nodes broadcast their perimeter vectors to the entire network
4) Each node uses these inter-perimeter distances to compute normalized coordinates for both itself and the perimeter nodes. Perimeter nodes project their coordinates onto the outer circle.
5) Perimeter nodes stay fixed while all other nodes run a relaxation algorithm.
6) A designated bootstrap node periodically broadcasts by which nodes periodically re-asses whether they lie on the perimeter or not.

Note that steps 1-4 are only required to bootstrap the coordinate assignment when the network is first initialized and only steps 5 and 6 constitute normal operation.

IV. Performance Evaluation

In the previous section we defined a complete algorithm for constructing virtual coordinates with no location information. We now evaluate this algorithm through a series of simulations; the algorithm we simulate includes all of the techniques we described, including bootstrapping beacons and projecting the coordinates onto a circle. As before, we consider two metrics: success rate of greedy routing and path length (in hops). These metrics allow us to evaluate the effectiveness of our virtual coordinate construction algorithm; in real usage, the success rate will be higher because various techniques for dealing with dead-ends could be employed.

perimeter nodes are arranged in a circle makes the choice of hash function simpler.

A. Experiment Design

To study the behavior of our solution for large scale networks, we have implemented a packet level simulator that is able to scale to tens of thousands of nodes. However, this scalability doesn't come for free: our simulator does not model all the details of a realistic MAC protocol. In particular, we consider (unless specified otherwise) a simplified MAC layer that models neither packet loss nor signal propagation. Radios have a precise (circular) radio range, and nodes can send packets only to nodes within this range. This simple model allows us to abstract away the impact of message loss and signal attenuation on routing performance, which would apply regardless of whether we have location information, and let us concentrate on how well our algorithm constructs virtual coordinates. Also, we try to only use metrics that do not depend on a using a specific MAC or physical layer technology to evaluate our algorithm.

In addition, to get a better idea of how our algorithm would work in a real environment, we present simulations in which we model:

- *losses* - nodes drop incoming packets with a given probability p. Since we do not model a specific MAC layer, radio technology or data-traffic pattern, we resort to a uniform loss model. While this may not be a realistic loss model, it does provide some insight into the robustness of the algorithm in the presence of losses.
- *mobility* - to simulate mobility, we use the random waypoint model [4].
- *obstacles* - we model obstacles by using straight *walls* that are parallel to the x or the y-axis. Nodes on one side of a wall cannot communicate with the nodes on the other side. To stress our algorithm, we consider scenarios with up to 50 walls.
- *irregular shapes and voids* - we create networks with voids, *i.e.,* regions inside the network that do not contain any nodes, and we simulate networks

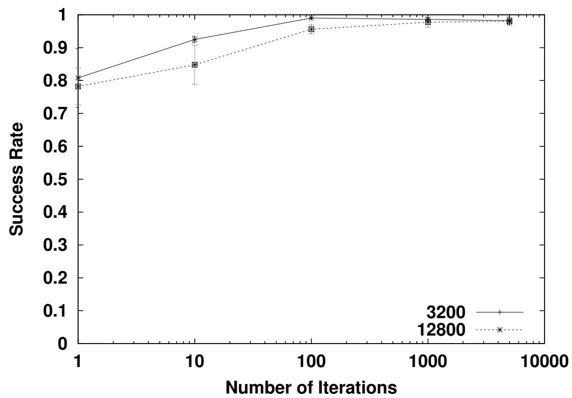


Fig. 6. The success rate of the greedy routing with virtual positions for two network sizes versus the number of iterations.

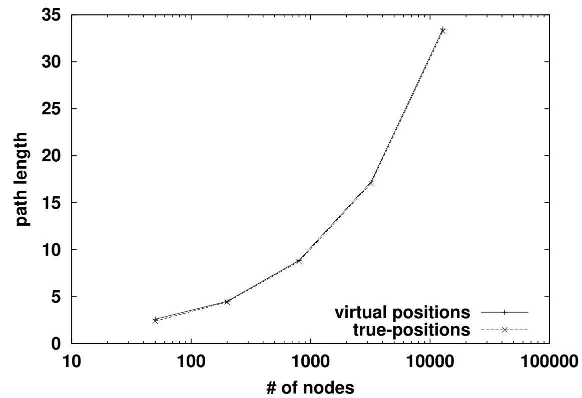


Fig. 7. The path length in hops using greedy routing with virtual positions. Note that x axis uses logarithmic scale

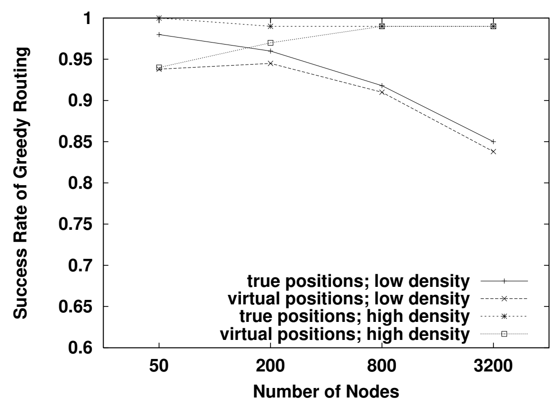


Fig. 8. The success rate of greedy routing with virtual and true coordinates for increasing network sizes at two different densities.

of various shapes, including concave shapes.

Most of our simulations involve the same basic scenario described in Section III: we use a 3200 node network, where nodes are uniformly distributed in an area of 200×200 square units (see Figure 1). The radio range is 8 units, and on average nodes have 16 neighbors.

Each node broadcasts heartbeats within its radio range every t sec, where t is uniformly distributed between 1 and 3 sec. Every heartbeat message contains a list of the sender's one-hop neighbors. This information is used by the receiver to learn its two-hop neighborhood. If a node does not hear for three heartbeat intervals from a neighbor n, it assumes that n is no longer its neighbor. In addition, there is a beacon node that periodically floods the network. Nodes use these floods to determine whether they are on the perimeter (see Section III-C). The beacon floods the network every 50 sec. While the choice of values for these timers is somewhat arbitrary, we found that these timers are able to accommodate mobility up to speeds of 0.64 units per second while keeping the traffic overhead reasonably low for static networks. Ideally, we would like to have algorithms that automatically tune these timers based on how dynamic the network is, but this is beyond the scope of this paper.

B. Scalability

In this section we study the scaling of our algorithm. We measure the success rate and the path length for greedy routing, and the overhead of the bootstrapping phase, as a function of the network size and node density.

Network Size Figure 6 shows the success rate of the greedy routing with virtual coordinates for both our baseline network with 3200 nodes, and for a network with 12800 nodes. Each point is the result of ten independent runs. In both cases non-perimeter nodes compute their initial coordinates based on the distances to the perimeter nodes as discussed in Section III-B.

We make several observations. First, the algorithm converges very fast. After only 10 iterations the success rate is already greater than 0.9. An iteration corresponds to heartbeat period, which in our case is 2 sec on average. After 100 iterations, the success rate reaches 0.97 for the 3200 node network, and 0.95 for the 12800 node network.[11] Second, these success rates

[11] The main reason the success rate for the 3200 node network is slightly lower (by 0.02) than in Section III-C is be-

Number of nodes	Number of flooding nodes
50	5.22
200	6.5
800	9.25
3200	14.75
12800	30.8

TABLE I
THE NUMBER OF FLOODING NODES DURING THE BOOTSTRAP PHASE AS THE NETWORK SIZE VARIES.

are very close to the success rates for greedy routing with true positions, *i.e.,* 0.99 for 3200 nodes, and 0.98 for 12800 nodes. Third, the success rates are very consistent across the runs, and the variation decreases with the number of iterations: the difference between the maximum and the minimum success rates for 1000 iterations is no larger than 0.02. Fourth, the success rate decreases very little (*i.e.,* from 0.97 to 0.95) as the network quadruples in size. This suggests that our algorithm is quite robust as the network size increases.

Figure 7 shows the path length (in hops) using greedy routing as the network size increases from 50 to 12800 nodes. The results show that using virtual coordinates for routing has virtually no impact on the path length: the path length is almost identical to the case when using true coordinates.

Density Figure 8 plots the success rate of greedy routing for two different node densities: one node per 12.5 square units (this is the same density as in our baseline network, *i.e.,* 3200 nodes uniformly distributed in a 200×200 grid), and a lower density of approximately one node per 19.5 square distance units (*i.e.,* 3200 nodes in a 250×250 grid). The success rate of greedy routing with virtual positions closely tracks the success rate with true positions. As expected, the success rate decreases as the density decreases. Finally, the path length when using virtual coordinates is within 0.2 hops from the path length when using true positions.

The density of nodes also affects the number of perimeter nodes that flood during the bootstrap phase. This is because small voids in the layout of nodes are more likely to occur at a low density thus causing more nodes to conclude that they are on the perimeter.[12] For example, for the 3200 node case above, the number of floods increased from approximately 14 at high density to about 30 at low density.

Overhead The most expensive operation of our algorithm is the bootstrap phase which requires perimeter nodes to compute and exchange the distances to each other. This essentially requires a certain fraction of such nodes to flood the network. In practice, a perimeter nodes suppresses its flood on hearing a flood from another perimeter node less than 5 hops away. Table I shows the increase in the number of such bootstrap floods as the network size increases. As expected, the number of bootstrap floods grows proportionally to the square root of the network size. We believe this growth is reasonable particularly since the cost of flooding is incurred only when the network is initialized.

After bootstrapping is completed, the overhead consists of (1) one beacon periodically flooding the entire network, and (2) each node periodically broadcasting within its radio range its neighborhood information. As a result, the overhead (in terms of number of messages and packet sizes) per node remains *constant*.

C. Mobility

In this section, we model node mobility by using the random waypoint model [4]. Each node picks a destination at random within the 200×200 square grid and moves towards the destination with a speed uniformly distributed in the range $[0, 0.64]$. The average speed is thus 0.32 which is equivalent to the average speed used in Broch et al. [4]. When a node reaches the destination point, the node remains stationary for a time interval called *pause time*, then selects another destination, and repeats. /horizontal

Let T be the time interval between two consecutive broadcasts initiated by the beacon node (see Section III-B). Recall that these broadcast messages are used by nodes to detect whether they are on the perimeter or not. Once a node decides that it is on the perimeter it projects its current virtual position on the circle. After this the node stops updating its coordinates until it decides again that it is no longer on the perimeter.

Figure 9 shows the success rate of greedy routing function of maximum pause time for $T = 10$, and $T = 50$

cause here the perimeter nodes are projected on the outer circle.

[12] Recall that a node decides that it is on perimeter if it is the farthest node to a designated beacon among all its two-hop neighborhood (see Section III-C).

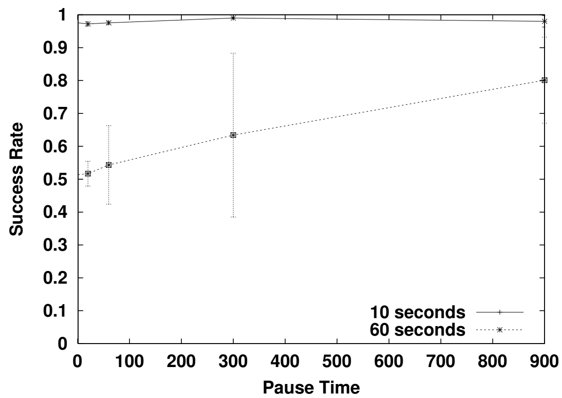


Fig. 9. The success rate of greedy routing versus the maximum pause time. The error bars represent the minimum and maximum values.

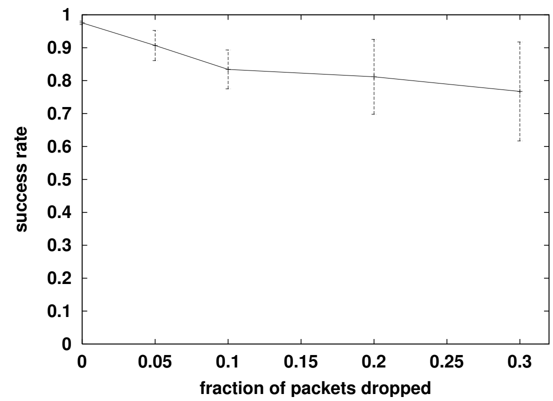


Fig. 10. The success rate of greedy routing as a function of loss rate.

sec, respectively. For comparison a node needs less than $8/0.32 = 25$ sec on average to move out of the radio range of a fixed node. As expected, the success rate of greedy routing decreases as T increases. When $T = 50$ sec, the success rate is as low as 0.5. This is because a node can cross several radio ranges before it can detect whether it is on the perimeter. In contrast, the success rate for $T = 10$ sec is at least 0.97. Finally, note that as the maximum pause time increases the success rate increases. This is to be expected since a larger pause time decreases the level of mobility.

D. Losses and Collisions

In this section, we study the robustness of our algorithm in the presence of losses. We model losses by randomly dropping *control* packets with a given probability p. To factor out the routing failures due to data packet losses we do not drop any data packets. While arguably this is not a realistic loss model, it allows us to study how robust is our algorithm when using incomplete information.

Figure 10 shows the success rate of greedy routing when the loss rate p increases from 0 to 30%. As expected the success rate drops as the loss rate increases. However, this drop is not severe. For 30% loss rate, the average success rate of greedy routing is still greater than 0.77. These results suggest that our algorithm is robust in the presence of packet losses. Intuitively, this is because the operations used to exchange control information, *i.e.,* broadcasts and periodic heart-beats, are highly robust. The success rate is greater than the probability of hop-by-hop packet delivery because we ignore losses on the data path.

E. Obstacles

In this section we study how our algorithm works in the presence of obstacles. We model the obstacles as *walls* with lengths of up to 50 units. For comparison, recall that the radio range of a node is 8 units, and a node knows only its two-hop neighborhood. Thus, for large obstacles it won't be always possible for nodes to get around the obstacles by just using greedy routing.

Figure 11 plots the success rate for greedy routing in our 3200 node network for different obstacle lengths, and for different number of obstacles. As expected, the success rate decreases as the number of obstacles and/or their length increases. Arguably, the most surprising result is that the success rate of greedy routing performs better with virtual coordinates than with geographic coordinates. For example, for 50 obstacles with length of 20 units, the success rate is as low as 0.64%. This compares with a success rate of 0.81% for virtual positions. Intuitively, this is because virtual positions better reflect the network connectivity than the real positions. Two nodes that are on each side of a wall can be very close in the geographic space although they can't communicate. In contrast, in the virtual space the same two nodes will be quite far apart.

We do not plot results with 50 obstacles when the length of the obstacles is higher than 20 units because the network become disconnected.

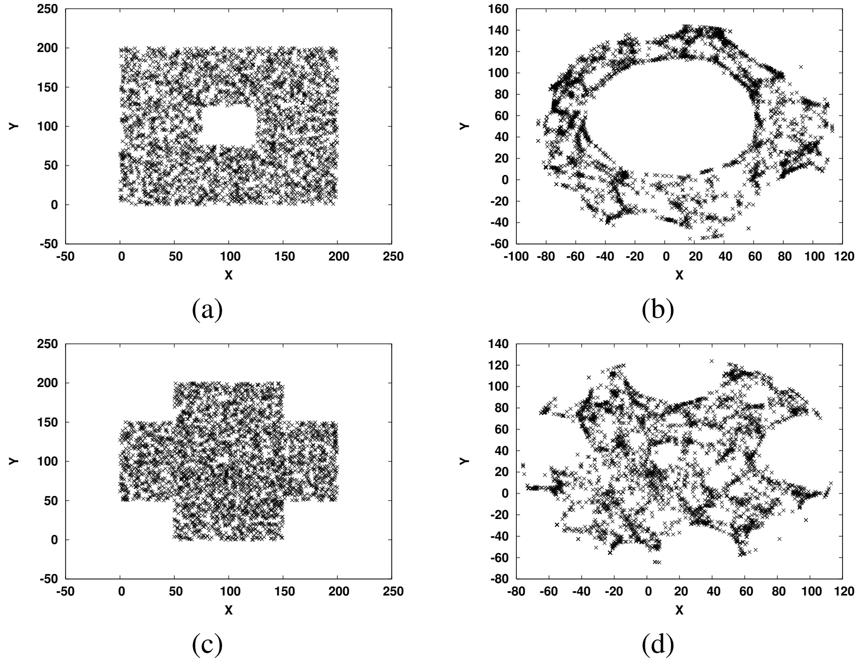


Fig. 12. (a) True positions and (b) virtual positions of a network with a large void in the center. (a) True positions and (b) virtual positions of a concave network.

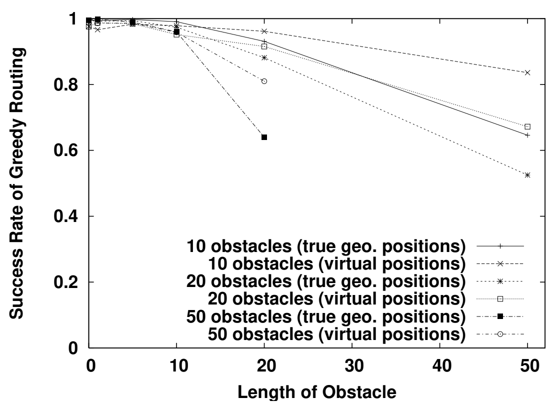


Fig. 11. The success rate of the greedy routing with true geographic positions and virtual positions for different number of obstacles and different obstacle sizes.

F. Irregular Shapes

Figure 12(a) presents the true positions of a network with 3200 nodes which contains a large void in the center. Figure 12(b) shows the virtual positions of the same network nodes. The virtual shape is rounded because our algorithm projects the perimeter nodes on a circle (see Section III-C). The center of the circle coincides to the center of gravity of the perimeter nodes, and the radius is equal to the average distance of the perimeter nodes to the center of gravity, as computed by the triangulation algorithm. As expected the virtual shape preserves the void. The success rate of the greedy routing with virtual coordinates is 0.97, which is higher than the success rate with true positions, 0.93. This is consistent to our results in the presence of obstacles, which showed that greedy routing with virtual positions outperforms greedy routing with true positions. The average path length is 18.48 for virtual positions versus 17.8 for true positions.

Similarly, Figure 12(c) shows the true positions of the nodes in a network with a concave shape, while Figure 12(d) shows the virtual positions of the nodes in the same network. The success rate of greedy routing with true positions is 1.0, while the success in the case of the virtual positions is 0.99. The average path length is 13.9 for true positions versus 14.3 for virtual positions.

In summary, our algorithms works well in the case of network with irregular shapes including networks with large voids. In addition, our algorithm preserves the general shape of the network in the geographic space.

G. 3-D Space

So far we have assumed that nodes lie in a 2-dimensional space. In this section we consider a 3-dimensional network with 3200 nodes uniformly dis-

Number of Iterations	Success Rate	Path Length
0	0.72	9.2
1	0.88	9.2
10	0.97	9.1
100	0.982	9.3
1000	0.978	9.3

TABLE II

SUCCESS RATE AND PATH LENGTH WITH INCREASING NUMBERS OF ITERATIONS IN A 3-DIMENSIONAL SPACE.

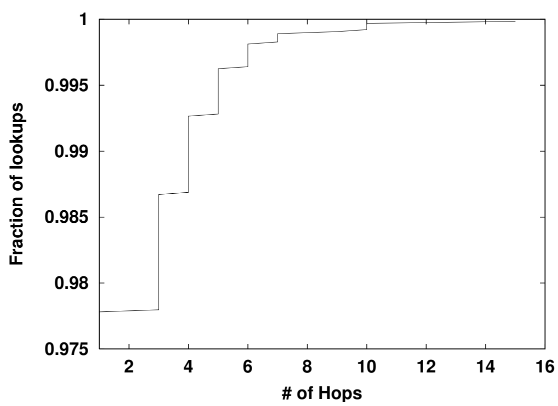


Fig. 13. Cumulative distribution of the number of hops required to reach the node closest to the destination from the node at which greedy routing stops.

tributed in a cube of side 75. Each node has 14 neighbors on average. Table II shows the success rate and the path length as the number of iteration increases. After 10 iterations the success rate reaches 0.97, and it exceeds 0.98 after 100 iterations. These results suggests that our algorithm works well in higher dimensional spaces too.

Max. TTL	messages (6400 queries)	success rate
no expanding ring search	1.049×10^5	0.985
3	1.111×10^5	0.995
4	1.140×10^5	0.998
unlimited	1.152×10^5	1.0

TABLE III

THE EFFECT OF EXPANDING RING SEARCH ON THE SUCCESS RATE AND OVERHEAD OF DHT LOOKUPS

H. Distributed Hash Tables

To implement a Distributed Hash Table (DHT), we associate with each item in the DHT a two-dimensional key (x, y) that maps within the perimeter circle. An item with key (x, y) is stored at the node closest to (x, y) in the virtual space.

To route to the node responsible for a key (x, y) we augment the greedy routing protocol with a simple expanding ring search scheme. When a packet is not able to make greedy progress, and the current node doesn't store the requested data item, the node performs an expanding ring search until a closer node is found or a TTL has been exceeded.

Figure 13 plots the CDF of the distance in hops from the node at which greedy routing fails to the node closest to the destination in the virtual space. Greedy routing terminates at the correct node in over 97.5% of cases. Only for a very small fraction (approximately, 0.2%) of lookups, the destination node is quite far (*i.e.,* farther than 8 hops) from the node at which greedy routing fails.

Table III shows the impact of the maximum time-to leave (TTL) of the expanding ring search on the success rate and the message overhead. The results represent averages over 6400 queries. As the maximum TTL increases, both the success rate and the overhead increase. In all cases the overhead increases by less than 4% which we believe is a small price to pay to achieve an 100% success rate.

I. Summary of Results

In summary, our algorithm consistently matches the performance of greedy routing with true positions over a range of simulation scenarios. In fact, our algorithm outperforms greedy routing with true positions when the network connectivity is inconsistent with network geography. As shown in Section IV-E, in the presence of a large number of obstacles, the success rate of greedy routing with virtual coordinates exceeds by up to 20% the success rate when true positions are used.

In addition, our algorithm is scalable. For example, in the case of a 12800 node network we require a total of only 30 nodes to flood the network in the bootstrap phase. After the bootstrap phase the overhead in terms of number of messages and message sizes is independent of network size. In particular, our algorithm only requires each node to periodically send heartbeats, and a single beacon node to periodically flood the network.

V. Conclusions and Future Work

In this paper we present a geographic routing algorithm that does not require nodes to know their location. Our key contribution is a relaxation algorithm that associates virtual coordinates to each node. These virtual coordinates are then used to perform geographic routing. Simulation results show that the success rate of greedy routing with virtual coordinates is very close to the success rate of greedy routing using true coordinates. Furthermore, in some cases such as in the presence of obstacles, greedy routing with virtual coordinates significantly outperforms greedy routing with true coordinates. Intuitively, this is because virtual coordinates reflect the network connectivity instead of the nodes' true positions which are less relevant in the presence of obstacles.

We plan to extend this work in four directions. First, we intend to continue the study of our algorithm by simulation using more realistic link layer models and network topologies. Second, we plan to study better heuristics to increase the success rate of DHT operations using greedy routing. One possibility would be to use waypoint routing *i.e.,* when a source fails to reach a destination, the source can pick a waypoint and ask it to perform routing on its behalf. Such a simple optimization has the potential to avoid voids and obstacles without the need for expanding ring searching. Third, we plan to explore the behavior of DHTs in networks with large voids. In such cases all items whose keys map within the void will be stored at nodes along the perimeter of the void. This can create storage and communication imbalance. Finally, we plan to implement our algorithm on sensor motes, and study it in real world scenarios.

References

[1] S. Basagni, I. Chlamtac, V. Syrotiuk, and B. Woodward, "A distance routing effect algorithm for mobility (DREAM)," in Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '98, (Dallas, Texas), August 1998.

[2] Nicklas Beijar Networking. Zone Routing Protocol (ZRP). `citeseer.nj.nec.com/538611.html`

[3] Prosenjit Bose and Pat Morin and Ivan Stojmenovic and Jorge Urrutia. Routing with Guaranteed Delivery in Ad Hoc Wireless Networks, In *Wireless Networks*, Vol. 7, pages 609 – 616, 2001. `citeseer.nj.nec.com/bose00routing.html`

[4] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking (MobiCom'98), ACM, Dallas, TX, October 1998.

[5] Douglas S. J. De Couto and Robert Morris, Location Proxies and Intermediate Node Forwarding for Practical Geographic Forwarding, MIT Laboratory for Computer Science technical report MIT-LCS-TR-824, June 2001.

[6] Gregory G. Finn. Routing and addressing problems in large metropolitan-scale intemetworks. ISi/RR-87-180, ISI, March 1987.

[7] J. Gao, L. J. Guibas, J. Hershburger, L. Zhang, A. Zhu, "Geometric Spanner for Routing in Mobile Networks ", In Proceedings of the 2nd ACM ", In Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001), pages 45–55, October 2001.

[8] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, Building efficient wireless sensor networks with low-level naming, In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Banff, Alberta, Canada, Oct. 2001.

[9] T. Imielinski and J. Navas. GPS-Based Addressing and Routing RFC nnnn, Computer Science, Rutgers University, March 1996. `citeseer.nj.nec.com/33074.html`

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[11] Per Johansson and Tony Larsson and Nicklas Hedman and Bartosz Mielczarek and Mikael Degermark. Scenario-based performance analysis of routing protocols for mobile ad-hoc networks, In *Proceedings of the fifth annual ACM/IEEE International Conference on Mobile computing and Networking*, pages 195 – 206, Seattle, Washington, 1999. ACM Press.

[12] David B. Johnson, David A. Maltz, and Josh Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. in Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5, pages 139–172, Addison-Wesley, 2001.

[13] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Mobile Computing, edited by Tomasz Imielinski and Hank

Korth, Chapter 5, pages 153–181, Kluwer Academic Publishers, 1996.

[14] David B. Johnson. Scalable and Robust Internetwork Routing for Mobile Hosts. In Proceedings of the 14th International Conference on Distributed Computing Systems, pages 2–11, IEEE Computer Society, Poznan, Poland, June 1994.

[15] Charles Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, in *Proceedings of ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, pages 234–244.

[16] B. Karp. *Geographic Routing for Wireless Networks*. Ph.D. Dissertation, Division of Engingeering and Applied Sciences, Harvard University, 2000.

[17] B. Karp and H. Kung. Greedy Perimeter Stateless Routing. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, 2000.

[18] Young-Bae Ko and Nitin H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks In *Mobile Computing and Networking*, pages 66–75, 1998.

[19] Jinyang Li, John Jannotti, Douglas S. J. De Couto, David R. Karger, and Robert Morris, A Scalable Location Service for Geographic Ad Hoc Routing, ACM Mobicom 2000.

[20] Wen-Hwa Liao and Jang-Ping Sheu and Yu-Chee Tseng. GRID: A Fully Location-Aware Routing Protocol for Mobile Ad Hoc Networks", In *Telecommunication Systems*, Volume 18, pages 37–60, 2001.

[21] Nathan Linial, Laszlo Lovasz, Avi Wigderson. Rubber bands, convex embeddings and graph connectivity. In *Combinatorica*, 8(1): 91-102 (1988).

[22] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. *TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks.* OSDI, 2002.

[23] Charles E. Perkins and Elizabeth M. Royer. "Ad hoc On-Demand Distance Vector Routing." Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pages 90–100.

[24] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, Data-centric Storage in Sensornets, In *ACM SIGCOMM HotNets*, Jul. 2002.

[25] S. Ratnasamy, B. Karp, D. Estrin, R. Govindan, and S. Shenker. GHT: A Geographic Hash-Table for Data-Centric Storage in SensorNets. Under submission to the *First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)* (June 2002).

[26] Y. Yu, D. Estrin, and R. Govindan. Geographical and Energy-Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks. UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023, May 2001.