Locating and Bypassing Routing Holes in Sensor Networks

Qing Fang
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
E-mail: jqfang@stanford.edu

Jie Gao
Department of Computer Science
Stanford University
Stanford, CA 94305
E-mail: jgao@cs.stanford.edu

Leonidas J. Guibas
Department of Computer Science
Stanford University
Stanford, CA 94305
E-mail: guibas@cs.stanford.edu

***Abstract*— Many algorithms for routing in sensor networks exploit greedy forwarding strategies to get packets to their destinations. In this paper we study a fundamental difficulty such strategies face: the "local minimum phenomena" that can cause packets to get stuck. We give a definition of stuck nodes where packets may get stuck in greedy multi-hop forwarding, and develop a local rule, the TENT rule, for each node in the network to test whether a packet can get stuck at that node. To help the packets get out of stuck nodes, we describe a distributed algorithm, BOUNDHOLE, to build routes around *holes*, which are connected regions of the network with boundaries consisting of all the stuck nodes. We show that these hole-surrounding routes can be used in many applications such as geographic routing, path migration, information storage mechanisms and identification of regions of interest.**

Keywords: Graph theory, System design, Combinatorics

I. INTRODUCTION

A sensor network is a collection of a large number of small devices each with sensing, computation and wireless communication capability. The energy constraint of a sensor node and frequently changed topology impose extra difficulties in the design of efficient routing protocols. The routing table approach used in wired networks requires a large amount of memory at each node and does not adapt easily to topology changes. Protocols for ad hoc mobile networks, are optimized for quick response to dynamic link conditions in mobile networks, hence, not suitable for wireless sensor networks, where sensors are static in most cases. Flooding the whole network consumes more resource than necessary, and should be avoided whenever possible. Greedy forwarding, as a simple, efficient and scalable strategy, is the most promising routing scheme for large sensor networks. In a geographical greedy forwarding scheme, a source node knows the location of the destination node, either by acquiring it from a location service [1], or by computing it using a hash function in a data-centric storage scheme [2]. A packet is forwarded to a 1-hop neighbor which is closer to the destination than the current node. This process is repeated until the packet reaches the destination. However, geographic forwarding suffers from the so called *local minimum phenomenon*. Specifically, a packet gets stuck at a node whose 1-hop neighbors are all further away from the destination.

To help packets get out of the local minimum, Karp and Kung [3], and independently Bose *et al.* [4], proposed the idea of combining the greedy forwarding and the perimeter routing on a planar graph that represents the same connectivity as the original network. When a packet gets stuck at a node, it is routed by the "right-hand rule" counter-clockwise along a face of the graph until it reaches a node that is closer to the destination than the one where the packet entered the perimeter routing phase. At this point, the packet returns to greedy forwarding phase. They showed that greedy forwarding along with perimeter routing guarantees delivery of the packet if a path exists. Perimeter routing requires the maintenance of the underlying planar graph, which introduces extra cost. More importantly, perimeter routing and the planar graph are not used most of the time. From the simulations that Karp and Kung did by using the greedy perimeter stateless routing (GPSR) protocol [3], most of the packets reach their destinations by greedy forwarding only. Therefore keeping a planar graph at every node all the time, which is used only occasionally seems unnecessary. We wonder what underlying geometric properties beget the local minimum phenomenon. Are there any structures behind the seemingly unorganized nodes? This paper is to answer those questions.

II. OVERVIEW

In this paper, we study a fundamental problem behind the "local minimum phenomenon" in geographic forwarding. We define the *stuck nodes* where packets can possibly get stuck in greedy multi-hop forwarding, and developed a local rule, the TENT rule, for each node in the network to test if it is a stuck node. To help packets get out of the stuck nodes, we developed a distributed algorithm, BOUNDHOLE, to find the so-called *holes*, the regions of the network with boundaries consisting of all the stuck nodes. Holes are usually associated with regions where nodes are depleted or regions that do not have enough nodes due to irregular terrain. Holes have sometimes been referred to as "communication voids" as well. Both our analysis and simulations show that the holes identified using this method indeed capture the underlying structure of the network and correctly identify regions with communication voids.

This paper focuses on defining and discovering holes in a sensor network, as well as building routes around them. In the process of discovering a hole, we also find the "boundary" of the hole, i.e., a closed cycle with no self-intersections that bounds a closed region. In other words, a hole is defined by its boundary. The boundary of a hole can be cached locally in that region, which provides a "conduit" for stuck packets being routed using the greedy forwarding scheme. Depending on application requirements, these routes can be found on demand, i.e., only when a packet gets stuck at a node, is the exploration of the boundary of the hole started. Boundary information is then saved at the nodes on the boundary to help follow-on packets. On the other hand, the routes around the holes can also be discovered in a preprocessing phase and stored locally along the boundaries of holes. Topological changes of a hole can be discovered and updated locally through careful design of protocols that handle node failures and additions.

Routing in a network with all the holes identified before hand can be very efficient. When a packet gets stuck under greedy forwarding, it must be at a stuck node. The locally stored information about the routes around the hole will help the packet get out of the local minimum in a way similar to that of perimeter routing on a planar graph. Unlike the planar graph approach, computing and storing the information of holes are only necessary at the problematic parts of the network where there are indeed communication voids. Similarly, holes can help geographic multi-cast [5], where data needs to be routed to a geographic region instead of a destination node specified by an address.

Information about holes can be used to help path migration, where a communication path is to be kept between two continuously moving objects of interest. Local and greedy path migration has the same problem as that in the greedy forwarding, where there is no local improvement when the routing path is along the boundary of a hole although "bypassing" the hole will give a much shorter path. The information about the holes tells when and how to shift the path from one side of the hole to the other side.

Holes can also be used in information storage mechanisms in sensor networks, such as the geographic hash table (GHT) [2] and DIMENSIONS [6]. A geographic hash table hashes data to points in the plane, which are in most cases not co-located with sensor nodes. A perimeter that encloses the destination is found by the GPSR protocol to replicate data. We show that in this case the planar graph used in GPSR to find the perimeter can be replaced by the holes identified using our algorithm. In fact, almost all the places where a planar graph is needed can be replaced by the holes, whose computation is more efficient and problem-oriented.

There are many applications where information about holes is especially useful. Some applications actually aim at finding the holes. In disaster detection, for example, a forest fire, destroyed the sensor nodes in the fire region and left a hole in the network. Figure 1 shows such a scenario, where a large hole is formed in the network. Detection of the boundary of this hole indicates the troublesome region. Massive drop-off of sensor nodes on irregular terrains leaves holes that correspond to mountains or shadows. Hence the holes can tell the geographic properties of the underlying terrain. Holes can also be used to detect regions with low sensor density due to depletion of node power. Therefore holes are the places where adding new nodes will dramatically improve the connectivity of the network.

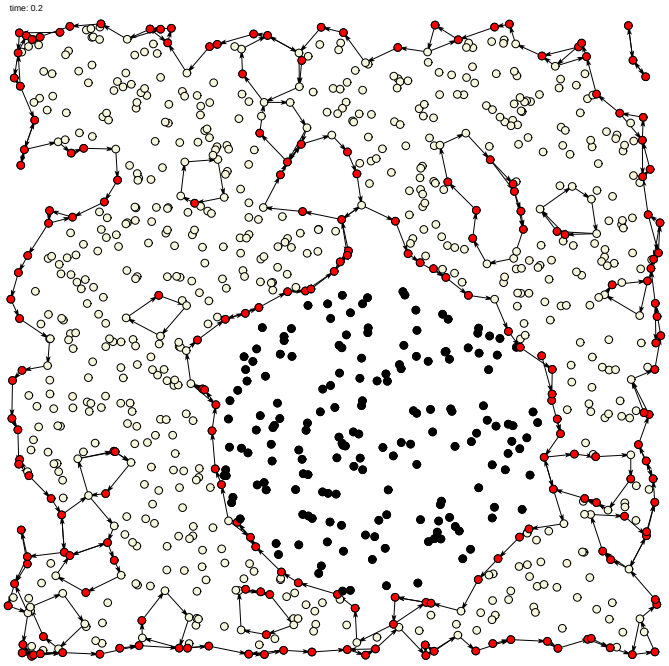

Fig. 1. The sensors in black failed, leaving a large "hole" in a network with sensors randomly placed. There are also small "holes" due to low sensor density. Small circles represent sensor nodes. Black nodes represent failed sensors. Red nodes represent stuck nodes that will be explained later.

In the following sections, we will introduce the algorithms of finding the stuck nodes and the holes, followed by the protocol issues, applications and discussions.

III. Algorithms

We discuss two types of stuck nodes: *weak stuck nodes* and *strong stuck nodes*. Assume there are n wireless nodes S in the plane, each with a communication range as a disk of radius 1. Each node can get the location information of itself and its 1-hop neighbors, either by GPS or other location services [7], [8], [9], [10].

In the remaining part of this section, we give the formal definitions of holes. We discuss the limitations of using the weak stuck node definition. We then propose a distributed greedy algorithm, BoundHole, to find holes using the strong stuck node definition, followed with proofs of correctness and guaranteed termination of the algorithm.

A. Weak stuck nodes and holes

1) Weak stuck nodes: A node $p \in S$ is called a *weak stuck node* if there exists a node $b \in S$ outside p's transmission range so that none of the 1-hop neighbors of p is closer to b than p itself. The destination b is called a *black node* of p, as shown in Figure 3.

This definition of stuck node suits applications where routing destinations are always some nodes in the network. In greedy forwarding, a packet can only get stuck at a stuck node. We show that holes under this definition are formally the faces of at least 4 vertices in the Delaunay triangulation with all the edges longer than 1 removed.

2) Finding the holes: For a set of nodes S in the plane, the *Voronoi diagram* partitions the plane into convex regions, called *Voronoi cells*, such that all the points inside one cell are closest to only one node. The *Delaunay triangulation* is the dual graph of the Voronoi diagram, by connecting the nodes whose corresponding cells are adjacent in the Voronoi diagram. The delaunay triangulation enjoys a "empty-circle" property: the circumcircle of a Delaunay triangle contains no nodes of S inside [11].

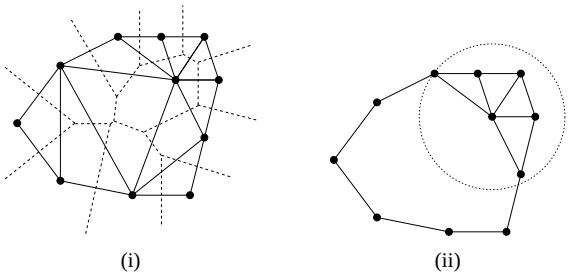

Fig. 2. (i) Voronoi diagram and Delaunay triangulation; (ii) Restricted Delaunay Graph.

Lemma 3.1. *In the Delaunay triangulation $DT(S)$, if all the edges adjacent to a node p are no longer than 1. Then p is not a stuck node.*

Proof: We prove by contradiction. Assume a node p is a stuck node and all the edges adjacent to p are less than or equal to 1. Assume that there is a node q so that all the 1-hop neighbors of p are farther away from q than p itself. We take one triangle $\triangle upv$ adjacent to p so that q is inside the cone defined by the two lines pu and pv, as shown in Figure 3. Since $\triangle upv$ is a Delaunay triangle, the circle centered at O defined by $\triangle upv$ has no other nodes inside. We draw the bisector ℓ_1 of edge up and the bisector ℓ_2 of edge pv. ℓ_1 and ℓ_2 intersect at point O and divide the plane into 4 quadrants. Only the points in the quadrant containing p are closer to p than u and v. Then node q must be inside the region bounded by ℓ_1, ℓ_2 and edges pu, pv. This region is fully inside the circumcircle of $\triangle upv$. This contradicts to the empty-circle property of the Delaunay triangulation.

□

We define the *Restricted Delaunay Graph (RDG)* to be the subgraph of the Delaunay triangulation so that only edges no longer than 1 are kept. (Notice the restricted delaunay graph defined in [12] may contain edges that are not in the Delaunay triangulation, as long as the graph is still planar.) We identify all the nodes with at least one adjacent delaunay edges longer than 1 to be the possible stuck nodes. Define a *hole* to be a face in the RDG with at least 4 vertices. Then from the previous lemma we know that,

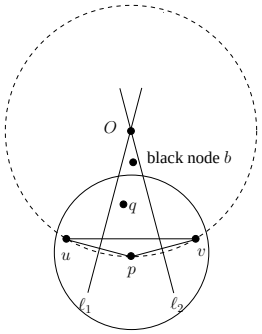

Fig. 3. Solid circle represents node p's transmission range 1. Dotted circle represents the circumcircle of $\triangle upv$.

Theorem 3.2. *All the weak stuck nodes must be on the boundaries of holes.*

Therefore, we can identify the weak stuck nodes and the corresponding holes by computing the Delaunay triangulation and removing the edges longer than 1. Figure 4 shows an example outcome of the holes identified by the Restricted Delaunay Graph in a 300m by 300m area with 1000 sensors placed uniformly at random. The algorithm for computing the Delaunay triangulation is a centralized algorithm [11]. Leibeherr *et al.* [13] proposed a distributed implementation, which is still a heavy algorithm for sensor networks. Moreover, the set of nodes identified using this method, is a super set of the set of real weak stuck nodes. Furthermore, the definition of weak stuck nodes is not inclusive enough for applications where the destination may not be a node in the network, as the scenarios discussed in the geographic hash table [2]. This motivates us to find better definitions and algorithms.

B. Strong stuck nodes and holes

1) Strong stuck node: We say a node $p \in S$ is a *strong stuck node* if there exists a location q outside p's transmission range in $\mathbb{R}^2$ so that none of the 1-hop neighbors of p is closer to q than p itself. The collection of q's is called the *black region*, shown as the shaded area in Figure 5. All the weak stuck nodes must be strong stuck nodes.

2) The TENT *rule:* We use a simple rule to detect the stuck nodes. For each node p, we first order all the 1-hop neighbors of p counterclockwise. For each pair of adjacent nodes u, v, we draw the perpendicular bisector of up and vp, which intersect at a center O. If O is inside the communication range of p, the black region (recall the black region is the collection of nodes inside the cone upv which are closer to p than u or v) must also be inside the communication range of p. Since u and v are adjacent in counterclockwise order, there are no nodes inside the black region. Therefore p can not get stuck

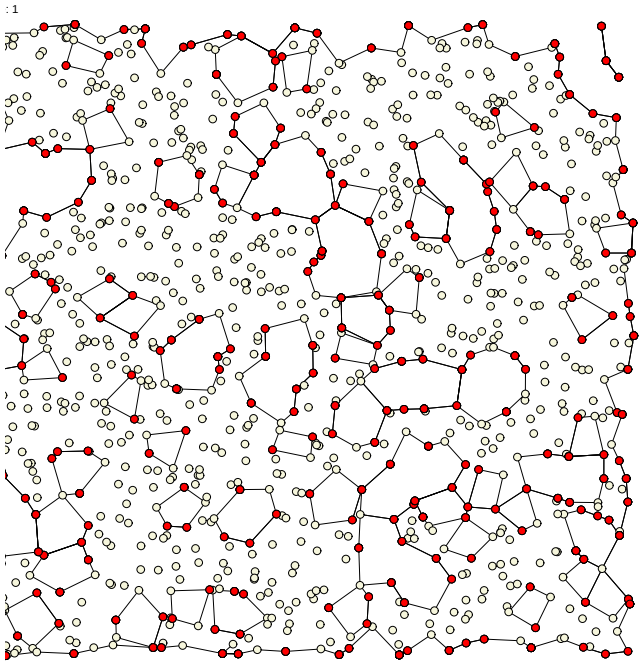

Fig. 4. A super set of weak stuck nodes (in red) and the holes identified by the Restricted Delaunay Graph.

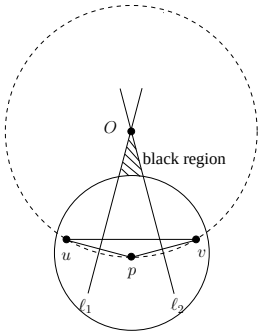

Fig. 5. Solid circle represents node p's transmission range 1. Dotted circle represents the circumcircle of $\triangle upv$.

for any destinations inside the cone defined by pu and pv. We say that p is not stuck in the direction of upv. Conversely, if the center O is outside the communication range, there must a destination in the plane where the packet can get stuck at p. So the TENT rule is both sufficient and necessary. To formalize, we call a *stuck angle* at a node p to be the angle spanned by a pair of angularly adjacent neighbors of node p in which direction p becomes a local minimum. A node is a strong stuck node if it has at least one stuck angle.

We do this local check for all the pairs of adjacent neighbors of p and identify all the possible stuck directions. The computation can be preformed with only information on 1-hop neighbors. The definition of a strong stuck node also implies that if the angle upv is smaller than 120 degrees, the node p can not get stuck. So one node can have at most 3 stuck directions.

Comparison of strong stuck nodes with the nodes identified by the Restricted Delaunay Graph shows that both of them are supersets of the weak stuck nodes, but neither is a superset of the other, as shown in Figure 6.

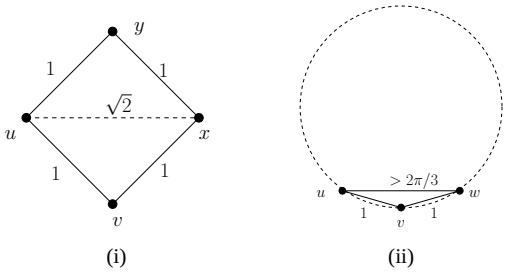

Fig. 6. (i) $uvxy$ is a unit square. ux is a delaunay edge with length $\sqrt{2}$ and therefore deleted in the restricted delaunay triangulation. u, v are identified as possible weak stuck nodes by the restricted delaunay method, but they are not strong stuck nodes; (ii) uvw is a delaunay triangle. uv and vw are edges of length 1. The angle uvw is larger than $2\pi/3$. v is a strong stuck node by the TENT rule, but is not identified by the restricted delaunay method.

C. BOUNDHOLE - *the finding hole algorithm*

Unlike the case of weak stuck nodes, where a hole can naturally be defined as a face in the Restricted Delaunay Graph, here we need to define what is a hole, how to identify them and how to route around them. This is the major contribution of this paper. The algorithm we propose is very simple. The basic intuition is shown in Figure 7. There are nodes s, p, t_1, t_2... bounding a hole. Node p is stuck at the direction spt_1 facing the hole. Our approach is to start from a stuck node p and sweep over the stuck direction and connect to a neighbor t_1 in a counterclockwise order. In the context of networking, we say "connect" meaning to have a packet hop to that node. Node t_1 further passes this packet on to its neighbor bounding the hole, in this case, node t_2. How to identify t_2 is what algorithm BOUNDHOLE addresses. Repeat this process for every node this packet hops to. We prove that the packet will mark the boundary of the hole and return to node p after touring the boundary. The hole is therefore defined as the closed region bounded by the cycle that BOUNDHOLE produces.

Having outlined the basic ideas, we can now formalize our description of the algorithm. We begin by introducing the following definitions.

Definition 3.3. *A* hole *is a closed region bounded by a non self-intersecting polygonal loop. The polygonal loop is called the boundary of the hole.*

Definition 3.4. *A hole H* belongs *to a strong stuck node v if H is identical to the loop formed by* BOUNDHOLE *starting at v.*

Definition 3.5. *We call a node u the* upstream *node of node v, if u is the previous hop neighbor sharing the same boundary of a hole with v in a clockwise order. In this case, v is called the* downstream *node of u.*

1) A greedy algorithm - BOUNDHOLE*:* Assume p is a stuck node and the angle spt_1 is the stuck direction, we use the following algorithm to find the hole that contains p on the boundary. Basically we are trying to find a closed cycle that goes back to p. The cycle is found by a local rule at each node. The cycle $pt_1t_2\cdots t_kp$ is oriented. Define the right hand side of each edge t_it_{i+1} is the positive side. The cycle divided the plane into two faces. The face with positive orientation, i.e., the face lies on the positive side of every edge t_it_{i+1}, is the hole. The algorithm works as described below.

1) We use the "right-hand rule" starting from t_1: take the first node t_2 by sweeping from p counterclockwise, such that t_2 is the first one hit by the sweeping line which is also not inside the shaded region in Figure 7. The shaded region is called the *forbidden region* of node t_1 where the next hop of the route can not fall inside. The forbidden region is essential in the algorithm since we want to make substantial progress at each step. If the angle spt_1 is greater than π, then the forbidden region is empty. t_2 is then notified by t_1 to continue this process.

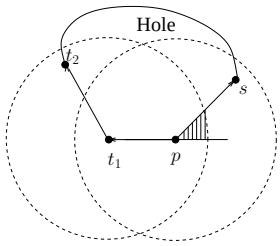

Fig. 7. Greedy sweeping at t_1.

2) This procedure is continued until the path $pt_1t_2\cdots t_k$ goes back to p and encloses a closed region.
3) **Edge intersection** For the case where edge t_jt_{j+1} intersects edge t_it_{i+1}, $j > i$, there could be two cases. For the edge intersection of the first kind, node t_j is not visible to node t_i and t_{i+1}. For the edge intersection of the second kind, node t_i is not visible to node t_j and t_{j+1}. These two cases are the only possible cases. The detailed proof appears in the appendix.
For the first kind, we simply delete the segments $t_{i+1}t_{i+2}\cdots t_jt_{j+1}$ and continue on $t_0t_1\cdots t_it_{j+1}t_j$, $p = t_0$, as shown in Figure 8.

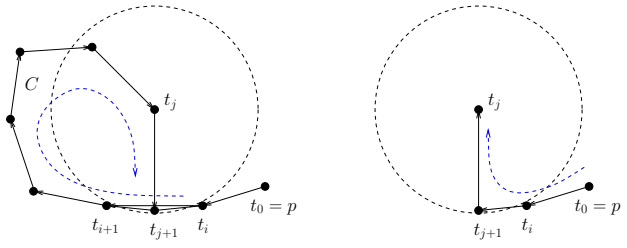

Fig. 8. The edge intersection of the first kind, before and after.

For the second kind, we take t_{i+1} as the next hop for t_j and continue on $t_0\cdots t_jt_{i+1}t_i$, $p = t_0$, as shown in Figure 9.

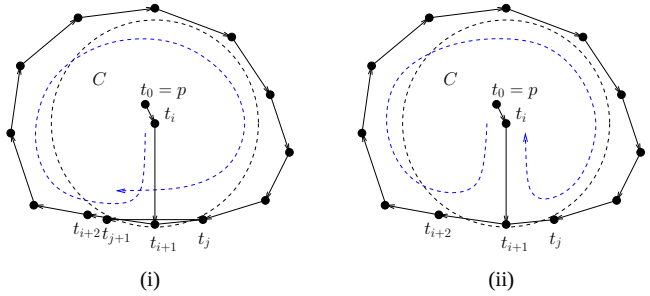

Fig. 9. The edge intersection of the second kind, before and after.

The algorithm BOUNDHOLE to find a hole for each strong stuck node and one stuck direction is simple and localized. The computation at each node depends only on the 1-hop neighbors. Thus the algorithm is distributed and scales well to large networks. Figure 10 shows the stuck nodes and holes identified using BOUNDHOLE for the same network configuration shown in Figure 4. If we compare the two figures, the result by BOUNDHOLE captures the topology of the holes more preceisely. The result by the Restricted Delaunay method has a lot of false positives. Furthermore, BOUNDHOLE produces "tighter" holes, i.e., the number of nodes on the boundary of a hole is less. This is due to the observation in the following subsection that we are making substantial progress at each step of BOUNDHOLE. We will refer to strong stuck nodes as stuck nodes from this point on.

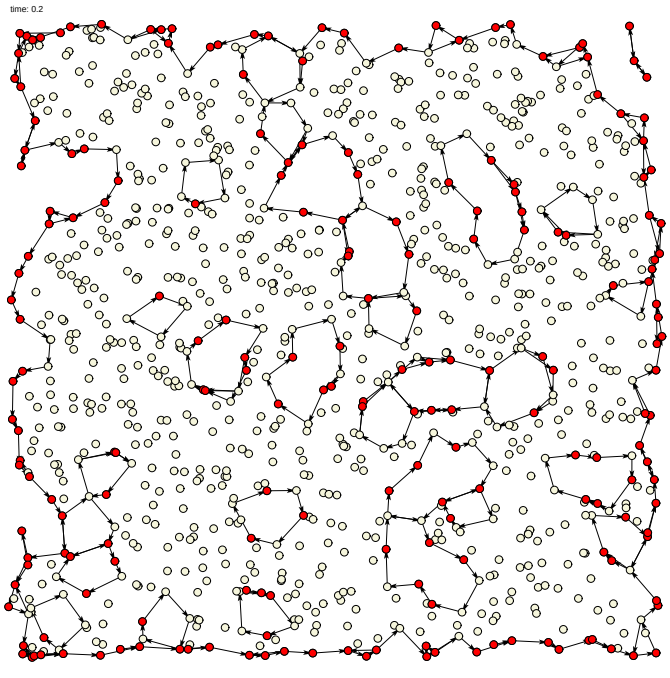

Fig. 10. Strong stuck nodes (in red) by the TENT rule and the holes identified by BOUNDHOLE.

2) Termination of the algorithm: Simple as the BOUNDHOLE algorithm, the proof that it terminates and indeed generates a finite hole is highly non-trivial. BOUNDHOLE generates a sequence of nodes $t_0t_1\cdots t_k$, where $t_0 = p$. We are going to prove that the sequence is finite, i.e., the algorithm terminates. The proof is based on an important property, which says we are indeed making progresses in generating the sequence.

Property 3.1. *If $t_{i-2}t_{i-1}t_i$ are consecutive nodes in the sequence, then t_i is not visible to t_{i-2}; or, the angle $t_{i-2}t_{i-1}t_i$ is greater than or equal to π.*

Lemma 3.6. *Property 3.1 is true for $i = 2$.*

Lemma 3.7. *For a piece of sequence $t_jt_{j+1}t_{j+2}\cdots t_k$ with no edge-intersections, if for $i = j + 2$, Property 3.1 is true, then Property 3.1 is true for all $j + 2 \le i \le k$.*

Lemma 3.8. *If the sequence $t_0t_1\cdots t_j$ satisfies Property 3.1, and there is an edge intersection t_it_{i+1} and t_jt_{j+1}, $j > i$, the sequence is modified to $t_0t_1\cdots t_{i-1}t_it_{j+1}t_j$ (or $t_{j-1}t_{i+1}t_i$), both $t_{i-1}t_it_{j+1}$ and $t_it_{j+1}t_j$ ($t_{j-1}t_jti+1$ and $t_jt_{i+1}t_i$) satisfies Property 3.1.*

With the help of the above lemmas, whose proofs are in the appendix, we have,

Theorem 3.9. *The sequence $t_0t_1\cdots t_k$, $t_0 = p$, generated by* BOUNDHOLE *satisfied Property 3.1, for all $i = 2,\cdots k$.*

Proof: We chop the sequence $t_0t_1\cdots t_k$ into a set of segments. Each segment stops at an edge intersection, i.e., when t_jt_{j+1} intersects with t_it_{i+1}, we stop the current segment at $t_i(t_j)$, and the next segment starts from $t_it_{j+1}t_j(t_jt_{i+1}t_i)$, for the edge intersection of the first(second) kind. We check the segments sequentially. For a segment $t_\ell t_{\ell+1}t_{\ell+2}\cdots t_m$, we assume all the segments before it have been proved to have Property 3.1. From Lemma 3.6 and Lemma 3.8, we know that the first three nodes $t_\ell t_{\ell+1}t_{\ell+2}$ satisfies Property 3.1. Then Lemma 3.7 says that the whole segment $t_\ell t_{\ell+1}t_{\ell+2}\cdots t_m$ satisfies Property 3.1. Therefore the whole sequence $t_0t_1\cdots t_k$ satisfies Property 3.1 too. □

By simple geometry, Property 3.1 implies,

Property 3.2. *The angle $t_{i-2}t_{i-1}t_i$ is greater than or equal to $\pi/3$. Also the forbidden region of t_i must be inside the visible range of t_{i-1}.*

Then we can prove the termination of the algorithm.

Theorem 3.10. *For any stuck node p,* BOUNDHOLE *terminates and gives a cycle $t_0t_1t_2\cdots t_kt_0$, $p = t_0$, which is not self-intersecting.*

Proof: To argue the termination of the algorithm, we only need to prove that any node u won't appear in the sequence infinitely many times. In fact, each node can only appear in the sequence at most 6 times.

Assume a node appears more than once, i.e., the path gets back to a node which it has seen. Assume the current path is $t_0t_1\cdots t_jt_{j+1}t_{j+2}$, with $t_i = t_{j+1}$, $i < j$.

If the angle $t_{i-1}t_it_{i+1}$ is greater than $t_{i-1}t_it_j$ (the angle is measured counterclockwise), t_i chooses t_{i+1} as the next hop, then t_j must be inside the forbidden region of t_i and therefore inside the visible range of t_{i-1}. In addition, $t_{i-2}t_{i-1}$ can not intersect with t_jt_i since we don't have edge intersections before. So t_{i-2} is inside the triangle $t_it_jt_{i-1}$, which must be inside the visible range of t_i. This contradicts with the fact that $t_{i-2}t_{i-1}t_i$ satisfy Property 3.1. See Figure 11 for an illustration.

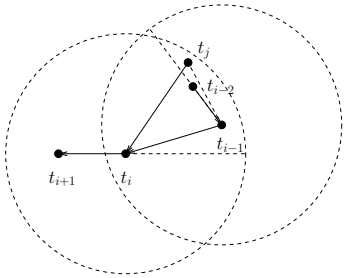

Fig. 11. t_j is inside the forbidden region of t_{i-1}.

If the angle $t_{i-1}t_it_{i+1}$ is smaller than $t_{i-1}t_it_j$, there are two possible cases, as shown in Figure 12. In this case, the angle $t_{i-1}t_it_{i+1}$ doesn't overlap with $t_jt_it_{j+2}$, where both the two angles are at least $\pi/3$, as shown by Property 3.2. So t_i can appear at most 6 times in the sequence. The total length of the sequence is at most $6n$. □

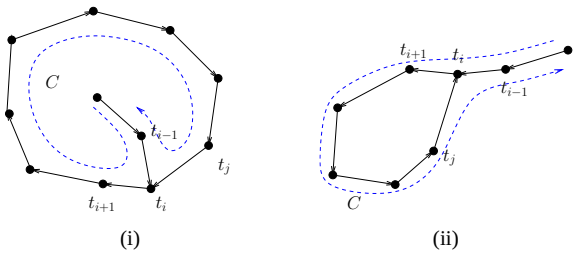

Fig. 12. Self-intersections at vertices.

IV. PROTOCOL AND VALIDATION

In the previous sections, we introduced a distributed algorithm that identifies the stuck nodes for greedy forwarding and connects the stuck nodes possibly along with some non-stuck nodes into cycles encircling areas that we call holes in a network topology. In this section, we discuss issues related to protocol design. Besides the implementation of BOUNDHOLE, some local optimizations are also made taking into consideration the network resource constraints. We simulated the TENT rule and BOUNDHOLE by a simulator we developed using C++ at the topology level to validate our algorithms and protocols.

For a protocol to be applicable to large-scale networks with limited resources, it should have the following properties. First, it should be distributed in nature for scalability. Second, it should not require synchronicity, which is hard to meet in a distributed wireless system. Third, it should converge quickly.

It this case, it is desirable to have the protocol converge in time proportional to the length of the perimeter of the hole.

At each stuck node, there is only one hole associated with it in each direction it is stuck. However, one node can be on the boundaries of multiple holes. One stuck node associates each stuck angle to the hole in the direction of that stuck angle, building a 1-to-1 mapping from the stuck angles to the holes. This is important for efficient implementation of the algorithm.

We define a *messenger packet* of a stuck node v as a packet originated by v to mark the boundary of the hole that belongs to v in the direction of a stuck angle. The choice for the next hop node that this packet visits is dictated by BOUNDHOLE.

A. Suppressed start

In the initialization stage of a network, each node broadcasts its coordinate to its one hop neighbors. Each node gathers its 1-hop neighbors' information, such as their IDs and coordinates. Then each node determines if it is a stuck node and the direction(s) it is stuck in by following the TENT rule. After the stuck nodes are identified, BOUNDHOLE is used to find the boundaries of the holes. This process begins as follows: A stuck node initiates a messenger packet in each direction this node is stuck in. Each messenger packet is sent to the second neighbor in counterclockwise order facing the direction of a stuck angle. The ID of the originator is recorded in each messenger packet. From this point on, BOUNDHOLE is used at each hop the messenger packet hops until it returns to its originating node and completes the cycle.

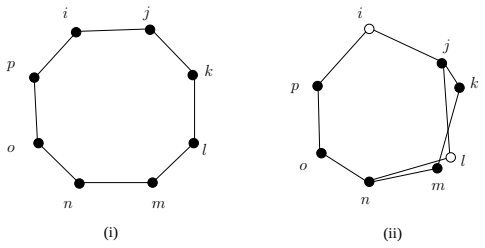

Fig. 13. Illustrations on cases for suppressed start: (i) the most common case in which suppressed start can effectively reduce traffic overhead; (ii) the case in which suppressed start may be overly suppressing, so that the hole belonging to node k may never get discovered as its messenger packet may get dropped by node such as o, p or i.

Such a cycle may consist of nodes that are not stuck nodes, in which case, they are only included to complete a cycle. There are, initially, no coordinations among the stuck nodes. They initiate their messenger packets in some random order. If we let each of them run the full algorithm to the end, many of the stuck nodes will find the identical hole. Figure 13 (i) shows an example. In such a scenario, eight messenger packets will be generated by eight stuck nodes on the boundary of a hole. Each packet will traverse all the nodes in a clockwise order and returns to its originating node. This causes unnecessary network traffic and worse yet, packet collisions. The situation gets worse especially for large holes shown in Figure 1. To avoid such overhead, we can suppress redundant hole finding processes. The basic idea is to drop redundant messenger packets as early as possible. The criterion for judging whether a messenger packet is redundant is as follows: at each node, if a messenger packet comes in via an ingress edge that an earlier messenger packet with a lower originator ID has taken, we consider the packet redundant and drop it. Consequently, the higher the ID of a stuck node, the less likely its messenger packet will travel far along the boundary. This effectively reduces the number of messenger packets in the network.

```
if (j receives messenger packet P via ingress edge e_ij)
    if (P's originator ID is not smallest among packets coming
            in via e_ij && P's enforce bit is not set)
        drop(P);
    else
        x = i;
        repeat
            x=TheFirstCCWNeighborSweepingFrom(x);
            // x is the next hop. in this case, x = l
        until (i is invisible from x)
        if (j had initiated a messenger packet to x and
            P's originator's ID is larger than j's ID)
            drop(P);
        else
            send(P) to x; // in this case, send to l
```

Fig. 14. Pseudo-code for suppressed start of BOUNDHOLE .

The downside of this simple scheme is that, in some rare case, some necessary messenger packets may also be dropped. Figure 13 (ii) shows such a case. In this figure, there are loop $j-l-n-o-p-i-j$ and loop $k-m-n-o-p-i-j-k$. These two sets of nodes share some common edges. For the edges they do not share, they will have crossing edges as shown in the figure. It is worth pointing out that edge crossings only happen between edges belonging to different stuck nodes. A loop belonging to one stuck node never intersects itself. If a messenger packet of stuck node k gets dropped along the shared part of the cycle, in this case, node o, p, i, node k may never get to find the hole belonging to it. Since the scenario shown here does not happen frequently in a randomly generated network topology, suppressed start is still valuable in reducing start up cost of the finding hole process in most of the cases. We solve the dilemma as follows. If a stuck node does not get its messenger packet returned within time T_m and it is not notified by other stuck nodes that it shares a common boundary with them, it re-sends a messenger packet with its *enforce bit* set to '1', so that other nodes relaying this packet will not intentionally drop this packet. This mechanism is also useful for fault tolerance purpose in case messenger packet gets lost due to packet loss. Details of the algorithm are shown in Figure 14. Please refer to Figure 13 (ii) for the scenario used for the pseudo-code. Again, the suppression of messenger packets is not required for the implementation of our algorithm. However, it is effective in reducing networking overhead and possible packet collisions, especially when large holes exist in the network.

The messenger packet originated by the stuck node with the lowest ID is guaranteed to be overlaid at each hop. After this

packet returns to its origin, the originating node carries out the following tasks:

- Generate a random ID for the hole;
- Claim itself as the leader node by sending refresh packets P_r recorded with its own ID and the hole ID to all the nodes sharing the same boundary every T_r time interval.

If a node's memory capacity allows, we may choose to cache some information about the shape of the boundary at each node to help optimize routing paths and support fast response to routing queries. We can also use the refresh packet to carry the boundary information and update each node it hops with the most current boundary node membership.

B. Handling node failures

Node failures may create additional holes in the network topology. Failure of a boundary node also changes the boundary of an existing hole. To detect node failures in the local neighborhood, each node periodically broadcasts its "heartbeat" to its 1-hop neighbors. The time interval of these announcements, T_h, is a system design parameter dictated by application requirements.

If a node v has not received such an announcement from one of its neighbors w for three consecutive T_h intervals, it first check whether it has an egress edge to node w.

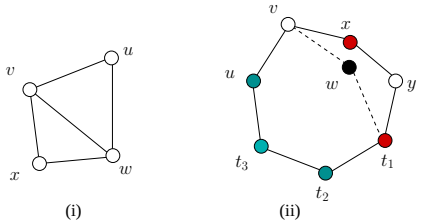

Fig. 15. (i) Node v is not stuck in the direction of $\angle uvx$ after the node w fails. (ii) There was a hole bounded by $vwt_1t_2t_3uv$. After the failure of node w, the hole is bounded by $vxyt_1t_2t_3uv$.

If the answer is 'no', v uses the TENT rule to test if it is stuck in the direction spanned by two new angularly adjacent neighbors u and x after the failure of w. If v is not stuck in that direction, the procedure stops. Figure 15 (i) shows such a case. If v is stuck in that direction spanned by $\angle uvx$, BOUNDHOLE is then used in finding the boundary of the hole that belongs to v in this direction.

If the answer is 'yes', such as that shown in Figure 15 (ii), v initiates a messenger packet starting from itself. The sweeping is done with the ingress edge being the edge from the previous neighbor of the failed node in a counterclockwise order, in this case, node u. This edge is also the previous ingress edge for the now defunct egress edge vw. Following BOUNDHOLE, v identify node x to be the next hop of the boundary. The finding hole process will continue from that point on. To avoid its packet being suppressed because of some node with smaller ID sharing the previous boundary, v sets the *enforce bit* in the packet to enforce each node the packet hops to overlay this packet. In the same figure, node t_2, t_3, u and w were the stuck nodes before the failure of w. The boundary of the hole was $v - w - t_1 - t_2 - t_3 - u - v$. After the failure, node x and t_1 become stuck in directions in which they were not stuck before. The boundary now becomes $v - x - y - t_1 - t_2 - t_3 - u - v$.

It is possible that a boundary node of a hole is no longer on the boundary after some topology changes because some failed nodes disconnect this node or newly added nodes repair the communication void. If a node is no longer chosen to be on the boundary of a hole, it will no longer receive refresh messages from its *upstream node*. This node then first check if it is a stuck node. If the answer is 'no', its status as a boundary node will retire according to a "retire timer" T_d. If the answer is 'yes', there are two possibilities: either the leader node died or part of the boundary failed. This node will then initiate a hole finding process using BOUNDHOLE. The choice of T_d depends on the applications. Generally speaking, the following relation is reasonable: $3T_h \leq T_r \leq 0.25T_d$. Such a mechanism makes the discovery and maintenance of holes a self-learning and adaptive process.

C. Information storage and memory requirement

Our algorithm is a practical solution for the local minimum problem. In the scheme proposed, a node does not need to store any additional information to support the algorithm, unless it is on the boundary of a hole. If a node finds itself indeed bordering a hole, depending on the application requirements, we can have a boundary node: 1) only store its upstream and downstream neighbors along the boundary of a hole; or 2) store information about all the nodes sharing the boundary; or in between, 3) store the size and a summary about the shape of the boundary. Storage cost for Option 1) is only O(1). Option 2) and 3) have higher demand on node memory, but capture the geometric shape of the hole and can be used to help improve the quality of routing paths, as well as support fast and efficient path migration. The planar graph approach can also be augmented for nodes to record such kind of information. But since we only compute and store the holes at the problematic parts of the network where there are indeed communication voids, we would expect to save both computation and storage compared with the planar graph approach. Furthermore, the cost is justifiable especially for applications where fast response to routing requests is needed and optimal paths are desirable.

V. APPLICATIONS

A. Routing

Greedy forwarding with holes identified can be done as follows. If a packet gets stuck with greedy forwarding, the packet is at a stuck node p. Then it must be on the boundary of a hole. We then route the packet along the boundary of the hole. when the packet gets to a node u whose distance to the destination q is closer than that of p, this packet follows greedy forwarding again. In case that the destination is outside the hole, such a node u must exist. In fact, if we connect the line pq, it crosses the boundary of the hole at an edge uv. Both u and v are closer to q than p itself. In this case the

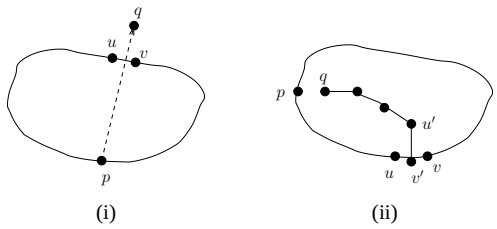

Fig. 16. (i) The destination q is outside the hole. (ii) Restricted flooding inside the hole.

packet always get closer to the destination and therefore the packet will finally reach the destination. If the destination is inside the hole, then it is possible that all the nodes on the hole are not closer to the destination than p is. For example, in Figure 16 (ii), p is the closest node to q among all the nodes on the boundary of the hole. q is indeed reachable via nodes v' and u' with u' inside the hole and v' outside the hole. But node u' is not inside the communication range of any node on the boundary of the hole. In such a case the routing along the boundary of the hole will come back to node p without being able to find a node closer to the destination. We then initiate a "restricted flooding" stage where each node on the boundary of the hole sends the packet to all its 1-hop neighbors, who will flood the nodes within the hole. For the example in Figure 16 (ii), node v' outside the hole will get the packet and then send it to q through u' eventually. In summary, by greedy forwarding with holes identified and with possible restricted flooding inside the hole, a packet will always get to the destination if such a path exists in the network.

B. Identifying regions of interest

The BOUNDHOLE algorithm was motivated by and developed for identifying regions with sparse network connectivity. However, the algorithm is applicable in identifying regions of any kind that can be defined according to some criteria testable locally, such as temperature, gas concentration, etc. One example is to find the regions in a sensor field with temperature higher than a constant. To the extreme, if we imagine a sensor field with infinite sensor density, then BOUNDHOLE can be used to find the isothermal contour in the field. For such an application, in addition to that each sensor node needs to know its neighbors' locations, it also need to know the temperature at each neighbor's location. If the temperature at a neighbor's location doesn't not pass the local test, we consider that neighbor does not exist and run BOUNDHOLE on the reduced neighbor set. Another interesting and important application for this algorithm is to identify traffic congestion regions in a network and build detour routes for transit packets. There are two aspects in solving such a problem. First, determine if the destination is inside the congested area. This can be readily handled by caching boundary information (detailed or summarized) at each boundary node. Second, build a route around the congested area so that route to the destination is guaranteed, should such a route exist. To do this, we need a way to measure the degree of traffic congestion. Once this is established, using predefined threshold to reduce the neighbor set and run BOUNDHOLE will yield the route for bypassing the congested area in the network. This is of interest for our future research.

C. Supporting path migration

Some sensor network applications require maintenance of virtual connections among a set of moving agents. For example, in a pursuer and evader scenario, a sensor network is used to monitor certain moving objects of interest, e.g., the evader. Information about the evader is constantly sent to the location where the pursuer is querying the network through a multi-hop path between the two. As the evader and pursuer move around, the communication path needs constant updates. Network infrastructure supporting fast response to path migration is desirable. In a dense network, path migration can be accomplished by finding shortest homotopy equivalence of the original path through greedy adjustment of the previous path. However, when communication void exists in the network, path migration by greedy adjustment based on only 1-hop neighbor information is impossible.

Figure 17 shows a scenario in which a communication path between two agents, A and B, is maintained. There is one "hole" in the immediate neighborhood of the path between A and B. As both A and B move towards the right, it is desirable to migrate the existing communication path to the right as well. However, because of the existence of the hole, there are no nodes to overlay the path between node A and B and the communication path is "stuck" at the boundary of the hole, shown as the narrow black line in Figure 17 (ii). Greedy decisions through local homotopy cannot overcome such irregularity in the network topology. In the scheme we proposed, information about the boundary of the holes can be cached locally at the nodes along the boundary. When the path migrates to the boundary, the two nodes at which the path crosses the boundary decide if the path should further migrate to the other side of the hole, based on the locally cached information about the shape of the hole.

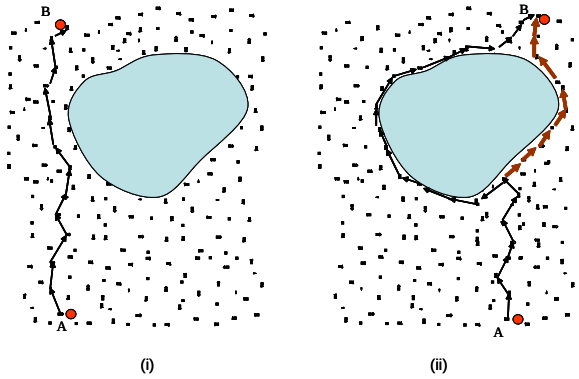

Fig. 17. A path migration scenario: (i) a multi-hop communication path is established between two mobile agents A and B, using the sensor network; (ii) as both agents move to the right, the path, shown as the thin line, gets stuck at the the boundary of the hole. The improved path is shown as the bold line.

VI. Discussions

A. Improvement of path quality

Although GPSR guarantees the delivery of a packet to any connected destination within the same network, it may use a long de-tour compared with the shortest path to the destination [14]. This is because the right-hand-rule used by the perimeter routing always guides the packet by going counter-clockwise along a face. But routing clockwise along a face may generate a much shorter path. Greedy forwarding combined with localized search and backtrack [15], [16] routes a packet along a path with length $O(k^2)$ if the shortest path is $O(k)$. A simple and more practical approach to improve the quality of a path is to store the shape of the hole on the boundary of the hole. When a packet gets stuck, it computes the better side to route around the hole. To do this, we do not need to save the exact shape of the hole, an approximation suffices. This approach could also be incorporated in the GPSR protocol where each node remembers the shape of the face it is on. Since we have a lot fewer holes than the number of faces in the planar graph, we can expect to get better performance in both storage and running time.

B. Impact of non-uniform transmission range

In the previous sections we assumed that the transmission ranges of the wireless nodes are uniform. If the transmission ranges are non-uniform, efficient routing problem in general becomes extremely hard. Indeed, if the radii of the communication coverage differ, the communication graph is no longer undirected. It is possible that one node u can send a packet to node v but node v can never send a packet to u, if v has a smaller communication range. Therefore u can not ever know v's existence. In the case where nodes have no global information of the distribution of the other nodes, the only way to send a packet to a sink like v is to flood the network blindly, let each node to send the packet to its neighborhood, and hope this packet would luckily get there sometime.

Although the non-uniform transmission range imposes fundamental difficulty to the routing problem. We should be aware that our finding hole algorithm will still do reasonable things in such situation. We let two nodes to claim each other as 1-hop neighbor only when both can hear from each other. The local algorithm will still find a cycle back to the original stuck node, although we can no longer say that greedy routing with the help of the holes can always get a packet to its destination if there does exist a path. Similarly, non-uniform communication range will cause the planar graph to be disconnected in the GPSR protocol, and thus delivery is not guaranteed.

VII. Conclusion and Future Work

This paper initiates the research on studying the structures in sensor networks. We believe that the stuck nodes and the holes defined in this paper have more applications than what have been mentioned here. Some of the applications in this paper, for example, network traffic congestion avoidance, deserve further study.

References

[1] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris, "A scalable location service for geographic ad-hoc routing," in *Proc. 6th Annu. ACM/IEEE International Conference on Mobile Computeing and Networking.*, 2000.

[2] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "GHT: A geographic hash table for data-centric storage in sensornets," in *1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002, pp. 78–87. [Online]. Available: citeseer.nj.nec.com/ratnasamy02ght.html

[3] B. Karp and H. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Proc. MobiCom*, 2000, pp. 243–254.

[4] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," in *3rd Int. Workshop on Discrete Algorithms and methods for mobile computing and communications (DialM '99)*, 1999, pp. 48–55.

[5] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," UCLA/CSD-TR-01-0023, Tech. Rep., 2001.

[6] D. Ganesan and D. Estrin, "DIMENSIONS:why do we need a new data handling architecture for sensor networks?" in *First workshop on Hot Topics in Networks*, 2002.

[7] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–667, August 2001.

[8] A. Savvides, C.-C. Han, and M. B. Strivastava, "Dynamic fine-grained localization in ad-hoc networks of sensors," in *Proc. MobiCom*, 2001, pp. 166–179.

[9] A. Savvides and M. B. Strivastava, "Distributed fine-grained localization in ad-hoc networks," *IEEE Transactions of Mobile Computing*, 2003.

[10] A. Ward, A. Jones, and A. Hopper, "A new location technique for the active office," *IEEE Personnel Communications*, vol. 4, no. 5, pp. 42–47, October 1997.

[11] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. New York, NY: Springer-Verlag, 1985.

[12] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Geometric spanner for routing in mobile networks," in *Proc. 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing*, 2001, pp. 45–55.

[13] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicasting with delaunay triangulation overlays," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, October 2002.

[14] M. Mauve, J. Widmer, and H. Hartenstein, "A survey on position-based routing in mobile ad hoc networks," *IEEE Network Magazine*, vol. 15, no. 6, pp. 30–39, November 2001. [Online]. Available: citeseer.nj.nec.com/article/mauve01survey.html

[15] F. Kuhn, R. Wattenhofer, and A. Zollinger, "Asymptotically optimal geometric mobile ad-hoc routing," in *Proc. Dial-M*, 2002, pp. 24–33.

[16] ——, "Worst-case optimal and average-case efficient geometric ad-hoc routing," in *Proc. MobiHoc*, 2003, pp. 267–278.

VIII. Appendix

a) Proof of Lemma 3.6: Property 3.1 is true for t_2, where $t_0 = p$, the original stuck node, since there are no nodes inside the fan defined by the angle spt_1 which is greater than $2\pi/3$. So t_2 is either outside the communication range of p or the angle pt_1t_2 is greater than π.

b) Proof of Lemma 3.7: We prove by induction. Property 3.1 is true for the case of $i = j+2$, by the assumption. Assume Property 3.1 is true for t_m, $m \leq i$. We take a look at the case of t_{i+1}. Assume otherwise, i.e., t_{i+1} is inside the communication range of t_{i-1} and the angle $t_{i-1}t_it_{i+1}$ is less than π, as shown in Figure 18. Then the reason that t_{i-1} choose t_i instead of t_{i+1} as the next hop must be that t_{i+1} is inside the forbidden region of t_{i-1}, which implies that t_{i-3} is inside the convex polygon bounded by $t_{i-2}t_{i-1}t_it_{i+1}$. Since t_{i+1}, t_i and t_{i-1} are all inside the communication range of t_{i-1}, so is t_{i-3}. This contradicts with the induction hypothesis.

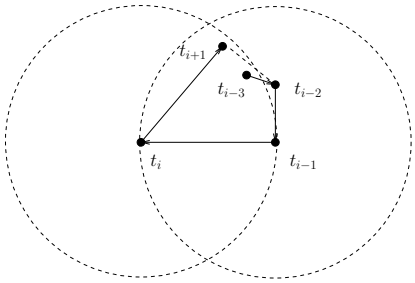

Fig. 18. If t_{i+1} is inside the communication range of t_{i-1}.

c) Proof of Lemma 3.8: Notice that due to BOUNDHOLE, whenever an edge intersection was detected, the current sequence of nodes can not have edge-intersections. Assume that the current edge intersection is edge t_jt_{j+1} and edge t_it_{i+1}, $i < j$. With the same argument as in Theorem 3.9, Property 3.1 is always true for all the three consecutive nodes before the current edge intersection.

We first argue that there are only two cases of edge intersections. The reason is that since t_j chooses t_{j+1} as the next hop, it must be (1) t_{i+1} is angularly closer with t_j than t_{j+1} in counter-clockwise order; or (2), t_{i+1} is angularly further away from t_j than t_{j+1} in counter-clockwise order. These two cases corresponds to Figure 8 and 9 respectively.

If t_{i+1} is angularly closer with t_j than t_{j+1} in counter-clockwise order, t_{i+1} is either inside the forbidden region of t_j, or is not visible to t_j. The first case is not possible, since otherwise we should have self-intersections already. In addition, t_i chooses t_{i+1} instead of t_j as the next hop, then either t_j is not inside the communication range of t_i, as shown in Figure 19 (i); or, t_j is visible to t_i but t_j is also inside the forbidden region of t_i, as shown in Figure 19 (ii).

Figure 19 (i) implies that Property 3.1 is true for the three consecutive nodes $t_it_{j+1}t_j$. In addition, we claim that $t_{i-1}t_it_{j+1}$ also satisfied Property 3.1. Assume otherwise, then the angle $t_{i-1}t_it_{j+1}$ is less than π and t_{j+1} is visible to t_{i-1}. Since t_{i-1} chooses t_i instead of t_{j+1}, the reason must be that t_{j+1} is inside the forbidden region of t_{i-1}. This implies that t_{i-3} is inside the triangle $t_{i-2}t_{i-1}t_{j+1}$, which is fully inside the communication range of t_{i-1}. Therefore t_{i-3} is also inside the communication range of t_{i-1}. This leads to contradiction that $t_{i-3}t_{i-2}t_{i-1}$ satisfies the Property 3.1.

For Figure 19 (ii), since t_j is inside the forbidden region of t_i, then t_j must be visible to t_{i-1}, by Property 3.2 and the assumption. Also t_{i-2} must be inside the convex polygon bounded by $t_j, t_{i+1}, t_i, t_{i-1}$, and therefore inside the communication range of t_i. This contradicts with the assumption that $t_{i-2}t_{i-1}t_i$ satisfied Property 3.1.

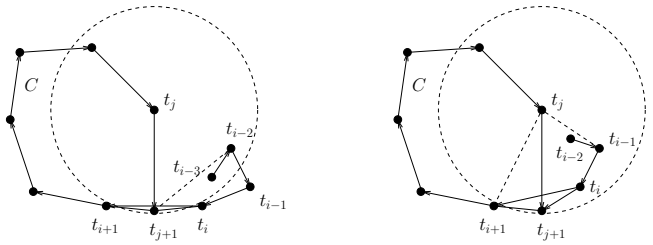

Fig. 19. (i) t_j is not inside the communication range of t_i; (ii) t_j is visible to t_i but t_j is inside the forbidden region of t_i.

If t_{i+1} is angularly further away from t_j than t_{j+1} in counter-clockwise order, as shown in Figure 9, by the same argument as above we can show that any three consecutive nodes in the sequence $t_0t_1 \cdots t_jt_{i+1}t_i$ satisfies Property 3.1. Thus the Lemma 3.8 is proved.