Locating and Bypassing Holes in Sensor Networks

Qing Fang[*] Jie Gao[†] Leonidas J. Guibas[‡]

April 8, 2005

Abstract

In real sensor network deployments, spatial distributions of sensors are usually far from being uniform. Such networks often contain regions without enough sensor nodes, which we call holes. In this paper, we show that holes are important topological features that need to be studied. In routing, holes are communication voids that cause greedy forwarding to fail. Holes can also be defined to denote regions of interest, such as the "hot spots" created by traffic congestion or sensor power shortage. In this paper, we define holes to be the regions enclosed by a polygonal cycle which contains all the nodes where local minima can appear. We also propose simple and distributed algorithms, the TENT rule and BOUNDHOLE, to identify and build routes around holes. We show that the boundaries of holes marked using BOUNDHOLE can be used in many applications such as geographic routing, path migration, information storage mechanisms and identification of regions of interest.

Keywords: Distributed Algorithms, Routing, Sensor Networks

1 Introduction

A sensor network is a collection of many small devices, each with sensing, computation and communication capability. It has many potential applications, such as building surveillance, industrial asset management, and environmental monitoring. However, designing scalable, self-organizing and energy-efficient sensor networks faces many challenges. Consequently, sensor networks have attracted researchers from a wide range of disciplines in recent years. A commonly used assumption in studying sensor networks is that sensors are uniformly densely distributed in the plane. However, in a real system deployment, this assumption does not generally hold. Even if sensors are distributed uniformly at random, there are still regions with sensor density much lower than others. Other factors such as terrain variation and sensor power depletion can also contribute to non-uniform sensor distributions. In practice, sensor networks usually have holes, i.e. regions without enough working sensors. Figure 1 shows an example of a large number of dead sensor nodes, creating a big hole in the network.

The appearance of such holes changes global network topology, and imposes additional difficulties in organizing the network. Many greedy algorithms that assume dense sensor deployment break down — most prominently, due to the *local minimum phenomenon* in *geographical greedy forwarding*. Geographical greedy forwarding, a simple, efficient and scalable strategy, is a promising routing scheme for large-scale sensor networks when sensor locations are available. In geographical greedy forwarding, a source node knows the location of the destination node, either by acquiring it from a location service [9], or by computing it using a hash function in a data-centric storage scheme [14]. 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, or the packet is stuck at a node whose 1-hop neighbors are all farther away from the destination. The node where a packet may get stuck is called a *local minimum*, or a *stuck node* in this paper. The existence of the local minimum phenomenon is due to the existence of *communication voids* in the sensor network, so that

[*]Department of Electrical Engineering, Stanford University, Stanford, CA 94305. Email: jqfang@stanford.edu.
[†]Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA 91125. E-mail: jgao@ist.caltech.edu. Work was done when she was at Stanford University.
[‡]Department of Computer Science, Stanford University, Stanford, CA 94305. E-mail: guibas@cs.stanford.edu.

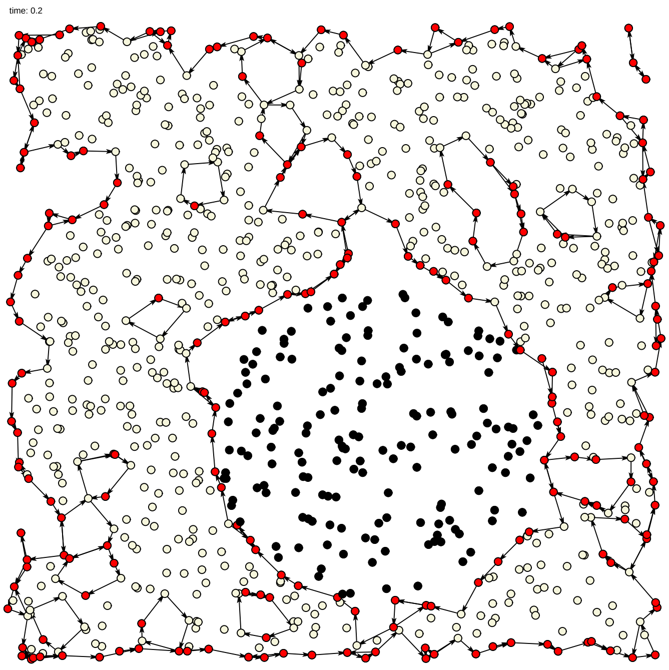

Figure 1. Hollow circles represent sensor nodes. The black nodes failed, leaving a large hole in a network with sensors randomly placed. There are also small holes due to low sensor density. Red nodes represent stuck nodes that will be explained later.

a packet cannot progress towards its destination by greedily examining only its local neighborhood. There are other examples of greedy algorithms that fail when the assumption of a dense sensor distribution no longer holds. For example, in a dense sensor field, we can use a local and greedy path migration algorithm to adjust a communication path between two moving objects to a homotopy-equivalent path. However, when the routing path is along the boundary of a hole, a local and greedy path migration algorithm faces the same difficulty as that in the greedy forwarding, i.e. there is no local improvement, although bypassing the hole will give a much shorter path.

In this paper, we give a mathematical definition of a communication void, i.e. a hole. We define holes to be simple regions enclosed by a (possibly concave) polygonal cycle which contains all the nodes where local minima can appear. We show that the local minimum phenomenon is simply caused by the existence of holes in the network. Routing in a network with all the holes identified beforehand can be very efficient. When a packet gets stuck at a node following geographic greedy forwarding, this node must be a stuck node on the boundary of a hole. By marking all the boundaries of holes, we actually find routes to get stuck packets out of the local minima — a stuck node can simply forward the packet to one of its neighbors that shares the same boundary. Thus, the boundary of a hole cached locally provides a "conduit" for stuck packets to get out of the local minimum and progress to its destination if a path towards the destination exists.

The phenomenon of holes is interesting in many ways. In disaster detection, for example, a forest fire may destroy all the sensors in the fire region and leave a hole in the network. Detection of the boundary for this hole indicates the fire region as shown in Figure 1. Massive drop-off of sensors on irregular terrains leaves holes that correspond to mountains or shadows. Hence the holes can indicate 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 significantly improve the connectivity of the network. Locating the holes in sensor networks will help us understand network topology and improve its connectivity.

Besides communication voids, holes can be defined differently for different applications. Generally, holes are simply regions in which certain measurements by sensors satisfy some local testable conditions. An algorithm that finds holes in networks thus can be used to identify regions of interests to the users. For example, if we let each sensor mark itself as unavailable if its local temperature exceeds a threshold, we can mark the boundaries that enclose all the sensors with local temperatures higher than the threshold. On the other hand, if we let each sensor keep a critical value which can be defined as the number of packets it has to send, or how busy its local wireless medium is, or simply the energy left at that node, all the sensors with their critical values higher/lower than a certain threshold are within regions called *hot spots* under their corresponding definitions. Therefore, it is possible to have a routing algorithm that bypasses these hot spots and adaptively routes packets, taking into consideration the real-time network traffic conditions and remaining power supplies.

In this paper, we focus on defining and discovering holes in a sensor network, as well as building routes around them. We formulate the problem using holes encountered in routing as a special and important case. We first define the *stuck nodes* where packets can possibly get stuck in greedy multi-hop forwarding. The existence of stuck nodes indicates the existence of holes. For each node in the network, to test if it is a stuck node, we developed the TENT rule which only requires each node to know its 1-hop neighbors' locations. To help packets get out of the stuck nodes, we developed a distributed algorithm, BOUNDHOLE, to find the *boundary* of the hole, i.e. a closed cycle with no self-intersections that bounds a closed region. The *holes* are defined as the regions of the network with boundaries consisting of all the stuck nodes. Thus, a hole is delimited by its boundary nodes. 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. Depending on application requirements, the boundary of a hole can be found *a priori* or *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. Topological changes of a hole can be discovered and updated locally through careful design of protocols that handle node failures and additions.

Another merit of our BOUNDHOLE algorithm is that it only requires angle information within 1-hop neighbors. Such information is completely local. Obtaining accurate angle information is much easier and less costly than obtaining accurate location information. Angle information with rather high accuracy can be derived from beam shaping using directional radiation beams. Although greedy routing uses location information, inaccuracies in node coordinates only result in suboptimal routes being generated. The hole bypassing technique presented in our paper is a better practical solution for bypassing local minima in greedy routing, where accurate node coordinates are unavailable.

2 Related Work

To help packets get out of the local minimum in a greedy forwarding scheme, Kranakis *et al.* [6] introduced face routing , Bose *et al.* [1], and independently Karp and Kung [5], proposed the idea of combining greedy forwarding and perimeter routing on a *planar graph* that represents the same connectivity as the original communication network. A planar graph is a graph embedded in the Euclidean plane without crossing edges. When a packet gets stuck at a node, it is routed by the *right-hand rule* counter-clockwise along a face of the planar graph until either it can be routed by greedy forwarding again, or it goes back to where it enters the perimeter mode, which means that no route to the destination is available. This scheme solves the local minimum problem and guarantees the delivery of a packet if a path exists. However, perimeter routing requires the maintenance of the underlying planar graph, which introduces an extra cost. More importantly, based on the simulations that Karp and Kung did by using the greedy perimeter stateless routing (GPSR) protocol [5], most of the packets reach their destinations by greedy forwarding only. Therefore, maintaining a planar graph at every node (even nodes where packets never get stuck), and at all times (even when there are no routing requests) is unnecessary. 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.

3 Algorithms – the TENT Rule and BOUNDHOLE

Assume there are n wireless sensor nodes S in the plane, each with a communication range as a disk with radius 1. The communication graph is thus modeled by the *Unit Disk Graph (UDG)*. We try to identify the holes – regions without enough sensors, in the sensor network. We do this in two steps. First we want to identify that there exists a hole. This is done by identifying stuck nodes, where packets may possibly get stuck. With the stuck nodes identified, we find the boundary of the holes. We discuss two types of stuck nodes: *weakly stuck nodes* and *strongly stuck nodes*, as well as their hole-surrounding routes respectively, in each of the following subsections.

3.1 Weakly Stuck Nodes and Holes

3.1.1 Weakly Stuck Nodes

A node $p \in S$ is called a *weakly stuck node* if there exists a node $q \in S$ outside p's transmission range so that none of the 1-hop neighbors of p is closer to q than p itself. This definition of stuck node suits applications where routing destinations are always some nodes in the network. In such case, a packet can only get stuck at a stuck node. For weakly stuck nodes, we define a hole to be a face with at least 4 vertices in the Delaunay triangulation with all the edges longer than 1 removed.

3.1.2 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 an "empty-circle" property: the circumcircle of a Delaunay triangle contains no nodes of S inside [13]. It's known that geographical forwarding doesn't get stuck in a Delaunay triangulation when the destination is also a node in the point set. In particular, one of the Delaunay neighbors of u must be closer to the destination than u [6]. However, a Delaunay triangulation may contain edges longer than 1 which are not available in a unit disk graph. If all the Delaunay edges attached at a node u are no longer than 1, u is not a weakly stuck node.

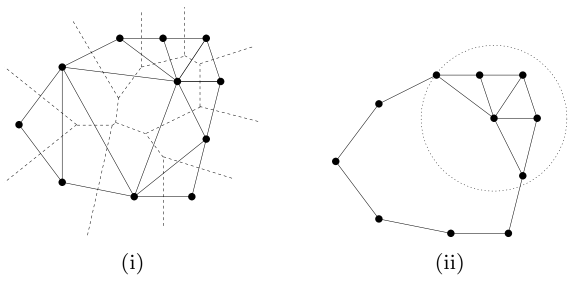

Figure 2. (i) A Voronoi diagram and corresponding Delaunay triangulation; (ii) A restricted Delaunay graph.

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[1]. We identify all the nodes with at least one adjacent Delaunay edge 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 analysis in the last paragraph we know that,

Theorem 3.1. *All the weakly stuck nodes must be on the boundaries of holes.*

Therefore, we can identify the weakly stuck nodes and the corresponding holes by computing the Delaunay triangulation and removing the edges longer than 1. Figure 3 shows an example outcome of the holes identified by the Restricted Delaunay Graph in a 300m by 300m area with 1000 sensors placed uniformly

[1]Notice that the restricted Delaunay graph defined in [4] may contain edges that are not in the Delaunay triangulation, as long as the graph is still planar.

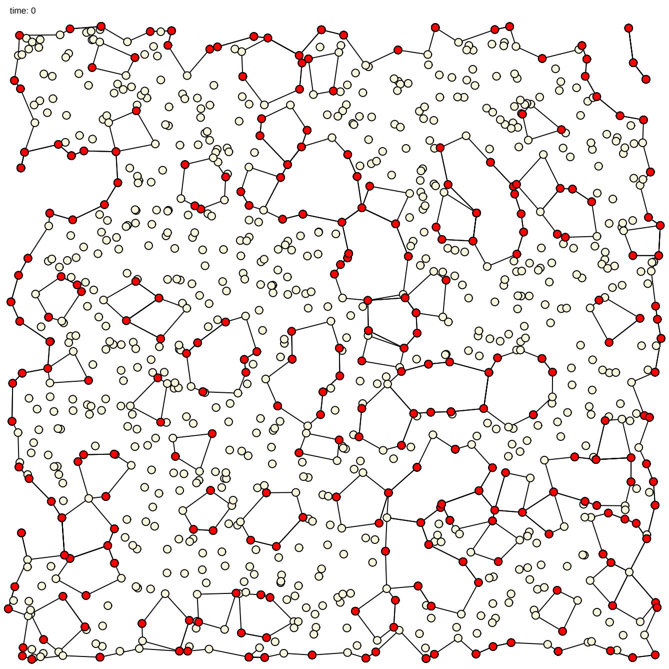

Figure 3. A super set of weakly stuck nodes (in red) and the holes identified by the Restricted Delaunay Graph.

at random. The algorithm for computing the Delaunay triangulation and the restricted Delaunay graph is centralized [13]. Leibeherr *et al.* [11] proposed a distributed implementation, which is still a heavy algorithm for sensor networks. Recently there has been local algorithms to compute variations of the restricted Delaunay graph [4, 15, 10]. However, they do not produce the restricted Delaunay graph nor does it identify Delaunay edges longer than 1. The distributed algorithm in [4] computes a planar subgraph of the unit disk graph that is a super graph of the restricted Delaunay graph. As an example in Figure 4, there are nodes inside the circumcircle of the triangle $\triangle vup$. In particular, pq is a Delaunay edge and uv is not a Delaunay edge. But this can not be found out by the algorithm in [4] and any algorithm with only the local information within a constant number of hops of p. Similar scenarios happen with the k-localized Delaunay graph defined in [10]

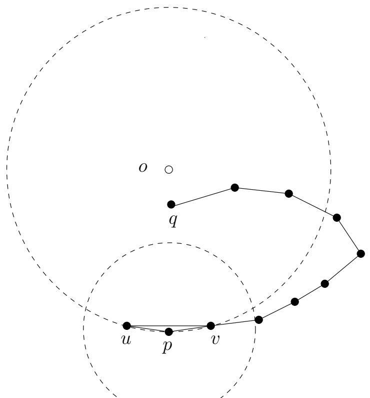

Figure 4. Algorithms by only local information cannot find out the existence of a long Delaunay edge.

that only uses k-hop local information.

Moreover, the set of nodes identified using this method is a super set of the set of real weakly stuck nodes. Furthermore, the definition of weakly stuck nodes is not inclusive enough for applications where the destination may not be a node in the network, as the scenarios discussed in geographic hash table [14]. This motivates us to find better definitions and algorithms.

3.2 Strongly Stuck Nodes and Holes

3.2.1 Strongly Stuck Nodes

We say a node $p \in S$ is a *strongly 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. By the definitions, all the weakly stuck nodes must be strongly stuck nodes.

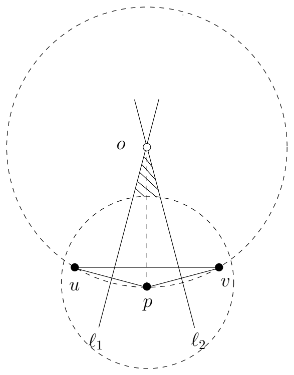

Figure 5. p is a strongly stuck node.

3.2.2 The Tent Rule

We use a simple rule to detect strongly stuck nodes. For each node p, we first order all the 1-hop neighbors of p counterclockwise. We note that the angle is defined counterclockwise. Denote by $B_r(x)$ the disk centered at point x with radius r. Now we focus on the case when the destination is inside the cone defined by vpu, where u, v is a pair of angularly adjacent nodes. We draw the perpendicular bisector of up and vp, ℓ_1, ℓ_2. 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. If o is inside the the communication range of p, for any point q inside the cone vpu and outside the transmission range of p, one of u, v is closer to q than p itself.

To formalize, the Tent rule checks whether the center of the circumcircle of $\triangle vpu$ is inside $B_1(p)$, the transmission range of p. If the test by the Tent rule is false, we call a *stuck angle* at a node p to be the angle spanned by a pair of angularly adjacent neighbors of node p, say $\angle vpu$, such that there is a point q inside the cone centered at p with angle $\angle vpu$ such that the distance between p, q is smaller than the distances from u, v to q. A node is not a strongly stuck node if it has no stuck angle. We note that the Tent rule is a necessary condition that guarantees to find all strongly stuck nodes. All the nodes where a packet may get stuck are marked by the Tent rule.

Lemma 3.2. *If the angle $\angle vpu$ is no more than 120 degrees, $\angle vpu$ is not a stuck angle, where u, v are angular adjacent neighbors. So one node can have at most 3 stuck angles.*

Proof: First nodes u, v are inside the communication range of p. If the angle $\angle vpu$ is no more than $2\pi/3$, then one of the angles, $\angle vpo$, $\angle opu$, must be at most $\pi/3$. Say $\angle vpo \leq \pi/3$, then the distance between p and o, $|po| = |ov| \leq |pv| \leq 1$. Thus the point o is inside the communication range of p. $\square$

We do this local check for all the pairs of adjacent neighbors of p and identify all the possible stuck angles. The computation can be performed with only information on 1-hop neighbors. Comparison of strongly stuck nodes with the nodes identified by the Restricted Delaunay Graph shows that both of them are supersets of the weakly stuck nodes, but neither is a superset of the other, as shown in Figure 6. From this point on, we will refer to strongly stuck nodes.

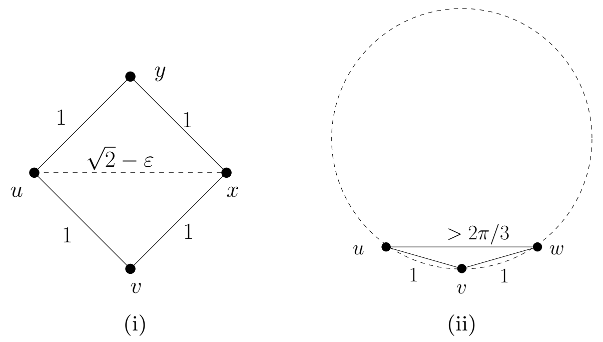

Figure 6. (i) $uvxy$ is a unit square. ux is a Delaunay edge with length $\sqrt{2}-\varepsilon$, with a small $\varepsilon > 0$, and therefore deleted in the restricted Delaunay triangulation. u, v are identified as possible weakly stuck nodes by the restricted Delaunay method, but they are not strongly stuck nodes; (ii) $\triangle wvu$ is a Delaunay triangle. uv and vw are edges of length 1. The angle $\angle wvu$ is larger than $2\pi/3$. v is a strongly stuck node by the TENT rule, but is not identified by the restricted Delaunay method.

3.3 BOUNDHOLE - the Finding Hole Algorithm

Unlike the case of weakly 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. Intuitively we want to define a hole as a region bounded by a Jordan curve[2], i.e., a simple closed curve in the plane. However, for a discrete sensor network, the boundary of a hole has to be a polygonal curve that consists of edges between discrete sensors. Thus it's not possible to avoid degeneracy, for example, the boundary of a hole can only be represented by a polygonal curve where a node or an edge appears multiple times. We define a directed polygonal curve C as a pseudo Jordan curve, if we can duplicate a node into multiple copies with infinitesimal perturbation, one corresponding to each appearance on the polygonal curve C, and the revised curve C' is a Jordan curve in the Euclidean plane. Thus a hole is a face bounded by a directed pseudo Jordan curve with no self edge intersections such that all the stuck nodes are on on the boundaries of all holes. Suppose node t_0 is a stuck node with stuck angle $\angle pt_0t_1$. The BOUNDHOLE algorithm starts at t_0 and finds a directed cycle $t_0t_1 \cdots t_k$, $t_k = p$, which is a pseudo Jordan curve and bounds a hole. To be specific,

Definition 3.3. *A* hole *is a region bounded by a directed pseudo Jordan curve, which is called the boundary of the hole. All the stuck nodes are on on the boundaries of all holes.*

Definition 3.4. *We call a node u the* upstream *node of node v, if uv is a directed edge on the boundary of a hole. In this case, v is called the* downstream *node of u.*

3.3.1 A Greedy Algorithm - BOUNDHOLE

Assume t_0 is a stuck node and the angle $\angle pt_0t_1$ is the stuck direction, we use the following algorithm to find the hole that contains t_0 on the boundary. Basically we are trying to find a directed closed cycle that goes back to p. The cycle is found by a local rule at each node. The algorithm works as described below.

1. Suppose the current node is t_i, we'll describe an algorithm to find the downstream node t_{i+1}. (At the first step, we take the current node as t_1 whose upstream node is t_0. The upstream node of t_0 is p.) We first define the forbidden region of t_i as follows. If the angle $\angle t_{i-2}t_{i-1}t_i$ is greater than π, then the forbidden region is empty. Otherwise, we take the ray ℓ at t_i with direction t_it_{i-1}, which intersects the communication disk of t_i at a point $\overline{t_{i-1}}$. The forbidden region of t_i is the intersection of the unit disk centered at t_i with the cone centered at t_{i-1} with angle $\angle\overline{t_{i-1}}t_{i-1}t_{i-2}$. See Figure 7. The downstream node of t_i can not be inside the forbidden region.

[2]A Jordan curve is a simple and closed curve in the plane which is topologically equivalent to (a homeomorphic image of) the unit circle.

We use the "right-hand rule" starting from t_i: we take a ray ℓ at t_i with direction $t_i t_{i-1}$ and rotate it angularly around t_i counterclockwise. t_{i+1} is the first node hit by ℓ such that t_{i+1} is not in the forbidden region of t_i. If $t_{i+1} \neq p$, then t_{i+1} will continue this process.

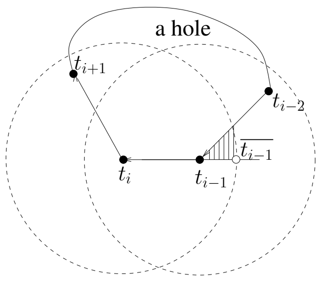

Figure 7. Greedy sweeping at t_1.

2. During the process we eliminate edge intersections. If an edge $t_j t_{j+1}$ intersects an edge $t_i t_{i+1}$, $j > i$, there could be two cases. For the edge intersection of the first kind, node t_j is not inside the communication range of both node t_i and t_{i+1}. For the edge intersection of the second kind, node t_i is not inside the communication range of both node t_j and t_{j+1}. We will show in the appendix that these two cases are the only possible cases. For the first kind, we simply delete the run of $t_{i+1} t_{i+2} \cdots t_j t_{j+1}$ and continue on $t_0 t_1 \cdots t_i t_{j+1} t_j$, as shown in Figure 8. For the second kind, we take t_{i+1} as the next

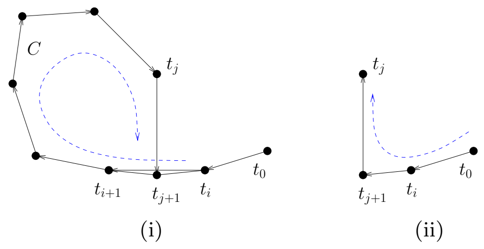

Figure 8. Eliminate an edge intersection. The dashed curve shows the sequence of directed edges before and after the elimination of the edge intersection.

hop for t_j and continue on $t_0 \cdots t_j t_{i+1} t_i$, as shown in Figure 9.

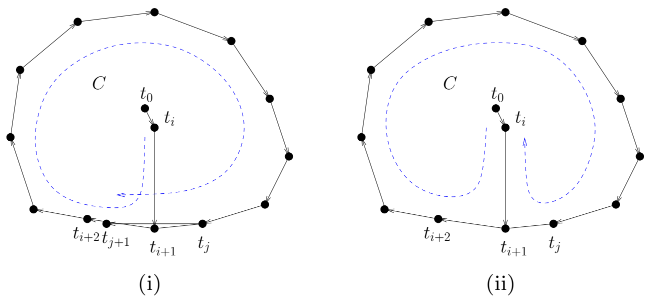

Figure 9. Eliminate an edge intersection. The dashed curve shows the sequence of directed edges before and after the elimination of the edge intersection. In this case, the boundary of the hole produced by BoundHole is $t_0 t_i t_{i+1} \cdots t_j t_{i+1} t_i t_0$. Notice that both t_i and t_{i+1} appear twice on the cycle. But the directed edges $t_i t_{i+1}$ and $t_{i+1} t_i$ appear once each.

3. This procedure is continued until we get a directed closed cycle $t_0 t_1 t_2 \cdots t_k$. This cycle has no self edge intersection and encloses a hole.

The algorithm BoundHole is simple and local. Each node stores information about its 1-hop neighbors. The computation at each node uses the 1-hop neighbor information and information carried with the packet that traverses the hole boundary in the bounding hole process. 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 as shown in Figure 3.

We compared the two approaches of detecting stuck nodes and holes. The result by BoundHole captures the topology of the holes more precisely. 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 smaller. The restricted Delaunay graph is a planar graph, thus the holes produced don't have crossing edges. The boundary of a hole produced by BoundHole is not self-intersecting. But the boundaries of two holes produced by BoundHole may cross each other.

These claims are supported by simulation results. We generated 50 network topologies, using the same setup as Figure 3 and 10. Each network has 1000 sensors, distributed uniformly at random in a 300m by 300m square field. The average number of stuck nodes detected by the Restricted Delaunay method is 256.4. Comparably, the average number of stuck nodes detected by the Tent rule is 208.5. We also observed that a significantly large fraction of the stuck nodes detected by the Tent rule are actually on the outer boundary of the field. For the finding hole methods, small holes detected by one method may not be holes by another. This is consistent with the analytical result. We compared the large holes detected by both methods. On average, the number of nodes on a hole boundary detected by BoundHole is about 14% smaller than that by the Restricted Delaunay method.

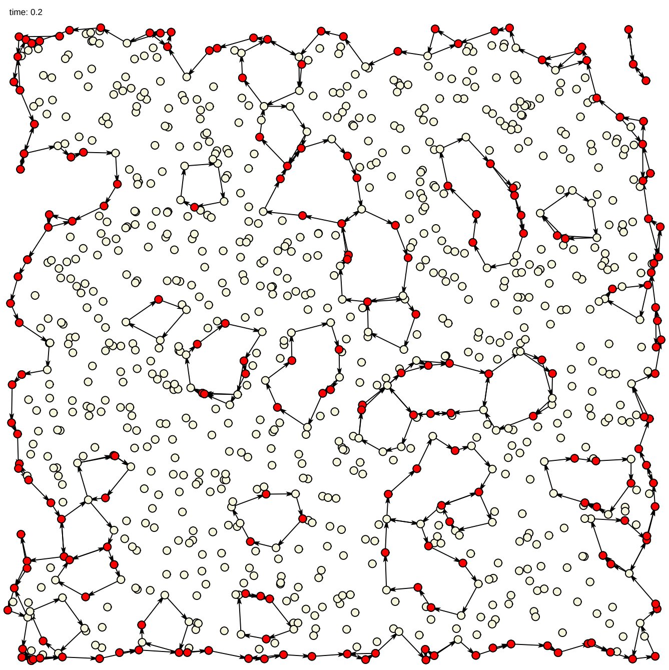

Figure 10. Strongly stuck nodes (in red) by the Tent rule and the holes identified by BoundHole.

Another merit of the BoundHole algorithm is that it can be implemented by using only angles between adjacent edges. As discovered in [2], by using only local angle information, we can detect all the crossing edges in the communication graph. The greedy sweeping in BoundHole can also be implemented by using only local angles. The angle information is completely local and can be measured using local interactions such as Angle of Arrival (AoA) or using directional antennas. Comparably, localization algorithm is expensive and inaccurate. The BoundHole algorithm, being less dependent on location information and more tolerant of measurement inaccuracy, is a better choice for practical implementation.

3.3.2 Validity of the Algorithm

As simple as the BOUNDHOLE algorithm is, the proof that it terminates and indeed generates a finite hole boundary is highly non-trivial. BOUNDHOLE generates a sequence of nodes $t_0t_1\cdots t_k$. We are going to prove that the sequence is finite, i.e. the algorithm terminates. The proof is based on two properties, which says that we are indeed making progress in generating the sequence.

Property 3.1. *The forbidden region of t_i is always inside the communication range of t_{i-1}. If $t_{i-2}t_{i-1}t_i$ are consecutive nodes in the sequence, then t_i is not inside the communication range of t_{i-2}; or, the angle $\angle t_{i-2}t_{i-1}t_i$ is greater than or equal to π.*

Property 3.2. *The angle $\angle t_{i-2}t_{i-1}t_i$ is greater than or equal to $\pi/3$.*

By simple geometry, we can show that Property 3.1 implies Property 3.2. The details appear in the appendix. We then show that the sequence generated by BOUNDHOLE satisfies Property 3.1 and 3.2.

Theorem 3.5. *The sequence $t_0t_1\cdots t_k$, generated by* BOUNDHOLE *satisfies Property 3.1, for all $i = 2, \cdots k$.*

Proof: We first show that each subsequence indeed has the Property 3.1 by the following three lemmas.

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 $j+2 < i \leq k$.*

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 $t_0t_1\cdots t_{j-1}t_jt_{i+1}t_i$), both $t_{i-1}t_it_{j+1}$ and $t_it_{j+1}t_j$ ($t_{j-1}t_jt_{i+1}$ and $t_jt_{i+1}t_i$) satisfy Property 3.1.*

With the help of the above lemmas, whose proofs are in the appendix, we can prove the theorem. We chop the sequence $t_0t_1\cdots t_k$ into a set of runs. Each run stops at an edge intersection, i.e. when t_jt_{j+1} intersects with t_it_{i+1}, we stop the current run at $t_i(t_j)$, and the next run 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 runs sequentially. For a run of $t_\ell t_{\ell+1}t_{\ell+2}\cdots t_m$, we assume that all the runs 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 $t_\ell t_{\ell+1}t_{\ell+2}$ satisfy Property 3.1. Then Lemma 3.7 says that the run of $t_\ell t_{\ell+1}t_{\ell+2}\cdots t_m$ satisfies Property 3.1. Therefore the whole sequence $t_0t_1\cdots t_k$ satisfies Property 3.1 too. □

Lemma 3.9. *If a node p appears multiple times on the boundary of a hole, for example, xpy and $x'py'$ are two pieces of the hole boundary, then the angles $\angle xpy$ and $\angle x'py'$ don't overlap.*

Proof: 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 $\angle t_{i-1}t_it_{i+1}$ is greater than $\angle t_{i-1}t_it_j$, 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 communication range of t_{i-1}. In addition, since we eliminate all edge intersections so the two edges $t_{i-2}t_{i-1}$ and t_jt_i don't intersect . So t_{i-2} is inside the triangle $\triangle t_it_jt_{i-1}$, which must be inside the communication range of t_i. This contradicts with the fact that $t_{i-2}t_{i-1}t_i$ satisfies Property 3.1. See Figure 11 for an illustration.

If the angle $\angle t_{i-1}t_it_{i+1}$ is smaller than $t_{i-1}t_it_j$, as shown in Figure 12. In this case, the angle $\angle t_{i-1}t_it_{i+1}$ doesn't overlap with the angle $\angle t_jt_it_{j+2}$. □

Then we can prove the termination of the algorithm.

Theorem 3.10. *For any stuck node p,* BOUNDHOLE *terminates and gives a cycle $t_0t_1t_2\cdots t_k$ (with t_k adjacent to t_0) with length at most $6n$, which has no self edge intersections.*

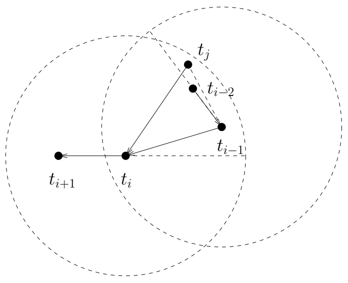

Figure 11. t_j is inside the forbidden region of t_{i-1}.

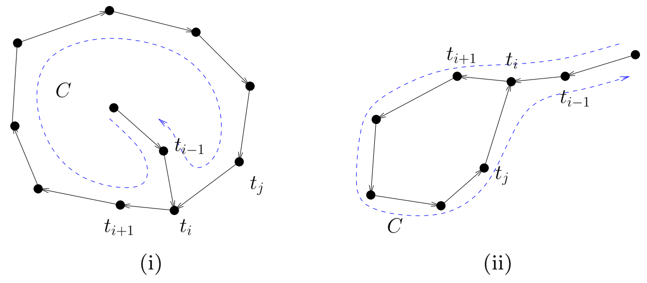

Figure 12. Self-intersections at vertices.

Proof: We only need to prove that any node u will not appear in the sequence infinitely many times. In fact, each node can only appear in the sequence at most 6 times. By Lemma 3.9, if a node appears multiple times in the sequence generated by BOUNDHOLE, e.g., xpy and $x'py'$ are two pieces of the sequence, then the angles $\angle xpy$ and $\angle x'py'$ don't overlap. Both angles are at least $\pi/3$, as shown by Property 3.2. So each node can appear at most 6 times in the sequence. The total length of the sequence is at most $6n$. □

Now we know that a cycle produced by BOUNDHOLE is of finite length. The next theorem says that this directed cycle is indeed a valid boundary of a region, i.e., a pseudo Jordan curve that bounds a hole. Not every directed cycle is a valid boundary of a hole. For example, a directed cycle with the shape of an '8' figure that goes counterclockwise on the top and clockwise on the bottom does not bound a non-empty positive face. Fortunately, the cycles generated by BOUNDHOLE are guaranteed not having this shape.

Theorem 3.11. *A directed cycle generated by* BOUNDHOLE *is a a pseudo Jordan curve that bounds a hole.*

Proof: The cycle generated by BOUNDHOLE is a cycle with no edge intersections. If the cycle doesn't have self intersections at nodes, then it is a Jordan curve that bounds a hole. If a node p appears multiple times on the cycle, by Lemma 3.9, these angles must be disjoint. Thus we can virtually spit p into multiple copies, one for each appearance on the cycle such that the revised cycle has no self intersections. See Figure 13. Thus the cycle generated by BOUNDHOLE is a pseudo Jordan curve that bounds a hole. □

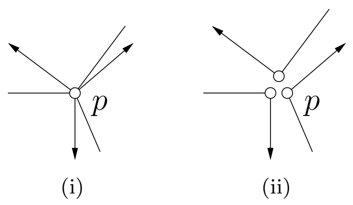

Figure 13. A directed cycle generated by BOUNDHOLE is a consistent boundary surrounding a hole. If p appears three times on the cycle, we virtually spit p into multiple copies, one for each appearance on the cycle. The revised cucle is a Jordan curve.

We summarize the properties of a hole identified by BOUNDHOLE as follows.

Theorem 3.12. *Suppose a hole is found by* BOUNDHOLE *with boundary $t_0 t_1 \cdots t_k$. Then the followings are true.*

1. *$t_0 t_1 \cdots t_k$ is a directed polygonal cycle of length at most $6n$ such that each angle $\angle t_{i-1} t_i t_{i+1} \geq \pi/3$.*
2. *The hole is tight in the sense that t_i and t_{i+2} are of distance at least 1 away.*
3. *If a node p appears multiple times on the boundary of a hole, for example, xpy and $x'py'$ are two pieces of the hole boundary, then the cones defined by the angles $\angle xpy$ and $\angle x'py'$ don't intersect.*
4. *A directed cycle generated by* BOUNDHOLE *is a pseudo Jordan curve surrounding a hole.*

3.4 Routing with Holes

Greedy forwarding with holes identified by BOUNDHOLE 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 the case that the destination is outside the hole, such a node u must exist.

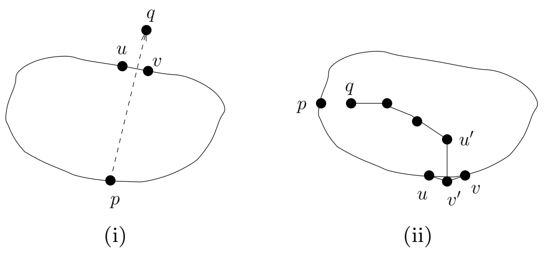

Figure 14. (i) The destination q is outside the hole. (ii) Restricted flooding inside the hole.

Lemma 3.13. *Suppose a packet is stuck at a node p and the destination is outside the hole where p stays on. Then there must be a node on the boundary of the hole whose distance to the destination is smaller than that of the stuck node p. That is, by routing along the boundary of the hole the packet will reach a node where greedy forwarding can be done.*

Proof: If a packet is stuck at p, first $|pq| > 1$. Also the destination q is inside a cone defined by a stuck angle at p. Thus, if we connect the line pq, it crosses the boundary of the hole at an edge uv. p, q are on different sides of the line defined by uv. We argue that one of u and v are closer to q than p itself. If otherwise, $|uq| > |pq| > 1$, $|vq| > |pq| > 1$. Thus $\angle qpu > \angle puq$, $\angle vpq > \angle qvp$. Since uv is an edge, $|uv| \leq 1$. Thus the angle $\angle uqv < \pi/3$. Therefore $\angle vpu = \angle vpq + \angle qpu > 5\pi/6$. Therefore $\angle puv < \pi/6$, $\angle uvp < \pi/6$. This implies that p is not adjacent to u or v on the boundary of the hole, since the inner angles of any three adjacent nodes on the boundary are at least $\pi/3$, by Theorem 3.12. However, since $\angle vpu > 5\pi/6$, $|pu| < 1$, $|pv| < 1$. Without loss of generality we assume the edge u is the upstream node of v. Thus we have a contradiction since u should have chosen p instead of v as the next hop. To summarize, the packet will leave the hole boundary and continue greedy routing when the destination is outside the hole. □

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 14 (ii), p is the closest node to q among all the nodes on the boundary of the hole. We note that a hole is a closed region so all the nodes inside the hole, if reachable from p, must be connected to p via some paths crossing the hole boundary. Say such a path connecting p, q has an edge $v'u'$ crossing an edge uv on the hole boundary, with u' inside the hole and v' outside the hole.

Then one of u', v' must be a 1-hop neighbor of one of u, v[3]. In such a case, the routing along the boundary of the hole may 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 14 (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.

4 Protocol

In the previous section, we described in detail the TENT rule and BOUNDHOLE. In this section, we discuss the protocol to implement them, and optimizations to achieve better network performance. We simulated the TENT rule and BOUNDHOLE at the topology level, using a simulator we developed in C++.

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 be asynchronous; third, it should converge quickly. Our protocol converges in time proportional to the length of the perimeter of the hole.

4.1 Protocol Overview

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. At each stuck node, we associate 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 algorithms.

In the initialization stage, each node broadcasts its coordinates to its 1-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 in which direction(s) it is stuck by applying the TENT rule.

After the stuck nodes are identified, BOUNDHOLE is used to find the boundaries of the holes. 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 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 neighbor on its left 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 reaches until it returns to its originating node, thus completes the cycle. After the messenger packet returns to its origin, the originating node carries out the following tasks:

- Generate a random ID for the hole, the *hole ID*;
- Claim itself as the *leader node* by sending a refresh packet P_r to announce its own ID, i.e. the *leader ID*, and the hole ID to all the nodes sharing the same boundary. The route that a refresh packet follows is determined by BOUNDHOLE. Such a refresh packet needs to be sent periodically by the leader node if changes in network topology has to be considered, such as node failures and additions. We use T_r to denote this time interval.

Up to this point, each node on the boundary of the hole knows its membership identified by the hole ID, the ingress and egress edges through which the messenger packet passes this node.

4.2 Suppressed Start

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 BOUNDHOLE to the end, many of the stuck nodes will find the same enclosed region. Figure 15 (i) shows such an example. In such a scenario, eight messenger packets will

[3]This is because for a pair of crossing edges uv, $u'v'$ with lengths no more than 1, at least one node is within distance 1 from the other three nodes.

be generated by eight stuck nodes on the boundary of a hole. Each packet will traverse all the nodes in a clockwise order and return to its originating node. This causes unnecessary network traffic and worse yet, packet collisions. Situations can get worse especially for large holes as shown in Figure 1.

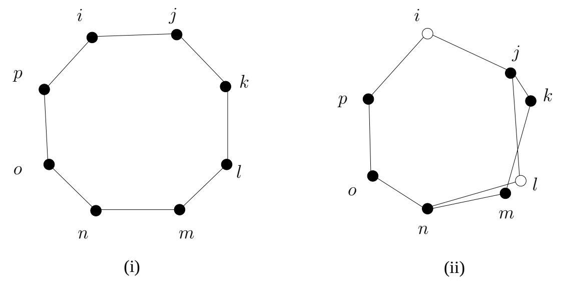

Figure 15. Illustrations on cases for suppressed start: (i) the most common case in which suppressed start can effectively reduce traffic overhead; (ii) the case in which suppressed start may be overly suppressing, so that the hole belonging to node k may never get discovered as its messenger packet may get dropped by nodes such as o, p or i.

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. The comparison of the originator IDs is necessary to guarantee that at least one messenger packet traverses the whole boundary of the hole. 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.

The downside of this simple remedy is that, in certain rare cases, some necessary messenger packets may also be dropped. Figure 15 (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 occur 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 segments of the cycle shared with some other nodes, i.e. o, p, i, node k may never be able to find the hole belonging to itself.

We solve this dilemma as follows. If a stuck node does not have 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 a messenger packet gets lost due to packet collision. Details of the procedure are shown in Figure 16. Please refer to Figure 15 (ii) for the scenario used for the pseudo-code.

4.3 Handling Node Failures and Additions

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 determined by application requirements.

If a node v has not heard the heartbeat of its neighbor w for several consecutive T_h intervals, it first checks whether it has an egress edge to node w.

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 17 (i) shows such a case. If v is stuck in the 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', as shown in Figure 17 (ii), v initiates a messenger packet. 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.

```
if ($j$ receives messenger packet $P$ via ingress edge $e_{ij}$)
    if ($P$'s originator ID is not smallest among packets coming in via $e_{ij}$ and $P$'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 outside the communication range of $x$)
        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 $l$
```

Figure 16. Pseudo-code for suppressed start of BoundHole .

Following BoundHole, v identifies node x to be the next hop of the boundary. The finding hole process will continue from that point on. In the 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 w fails, node x, t_1, t_2, t_3, u are the stuck nodes. The boundary now becomes $v - x - y - t_1 - t_2 - t_3 - u - v$.

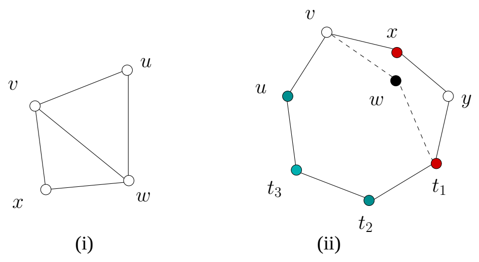

Figure 17. (i) Node v is not stuck in the direction of $\angle uvx$ after the node w fails. (ii) There was a hole bounded by $vwt_1t_2t_3uv$. After the failure of node w, the hole is bounded by $vxyt_1t_2t_3uv$.

It is possible that a boundary node of a hole is no longer on the boundary because some failed nodes have disconnected this node or newly added nodes have repaired the hole. If a node is no longer on the boundary of a hole, it will no longer receive refresh messages from its *upstream node*. When this occurs, this node first checks if it is a stuck node. If the answer is no, its status as a boundary node will retire after a "retirement timer" T_d expires. If the answer is yes, there are two possibilities: either the leader node died or part of the boundary failed. This node then initiates a hole finding process using BoundHole. The choice of T_d depends on the application. 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.

4.4 Information Storage and Memory Requirement

The solutions we proposed in this paper are practical for sensor networks. For the finding hole process, a node does not need to store any additional information, 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 the 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 path quality, as well as support fast and efficient path migration. The planar graph approach can also be augmented to have nodes record such information. But since we only compute and store the holes at problematic parts of the network where there are indeed communication voids, we would expect to save both computation and storage cost compared to the planar graph approach.

5 Applications

5.1 Generating Contour Lines

The BoundHole algorithm can be used to identify regions of interest, as long as the conditions for including a sensor in a region can be tested locally, such as temperature, gas concentration, electromagnetic signal strength, etc. One example is to find the regions with temperatures higher than a constant. To accomplish this, a sensor needs to know not only its neighbors' locations, but also the temperature at each neighbor's location. If the temperature at a neighbor's location is higher than the constant, we consider that neighbor does not exist and run BoundHole on the reduced neighbor set. When sensor density is high, BoundHole can be used to find temperature iso-contour lines in the field by repeating the above process using different temperature thresholds. For military applications, contour lines of electromagnetic signal strength emitted from enemy vehicles may help us differentiate how many vehicles are present.

5.2 Avoiding Network Hot Spots

The Tent rule and the BoundHole algorithm can also be used to discover hot spots in the network, and build detour routes for transit packets. To accomplish this, we need a way to determine if there exists local traffic congestion. One possible way is to use MAC layer information on wireless media availability. Once the local test is established, we can use this test to reduce a node's neighbor set and run BoundHole to obtain the route for bypassing the congestion area in the network. When a packet encounters a node lying on the boundary of a congested region, the node determines if the destination is inside the region. This can be handled by caching boundary information (detailed or summarized) at each boundary node. If the destination is not inside the region, the packet must be a transit packet. Therefore, the packet should be detoured along the boundary of the congested area. Consequently, while avoiding network hot spots, a route to the destination is also guaranteed if such a route exists. Similarly, hot spots can be defined as regions with sensor nodes with low power levels.

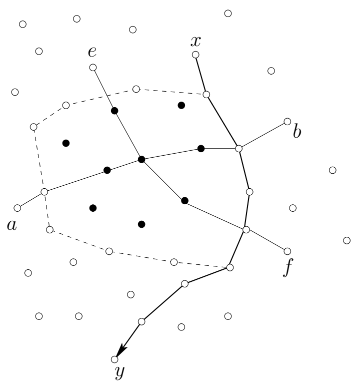

Figure 18. A multi-hop path between node a and b crosses a multi-hop path between node e and f, creating a hot spot including all the sensor nodes in dark shade. The perimeter enclosing these dark nodes is the boundary marked using the Tent rule and the BoundHole. Later, for a packet starting at node x to be routed to node y, the connected line segments shown in bold will be the actual route taken in order to avoid further increasing network traffic in the region.

Because hot spots are prone to high interference and long network delay, a routing algorithm adaptive to the real-time traffic conditions is able to route packets in healthier parts of the network. Therefore, the network performance under heavy work load degrades more gracefully.

5.3 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 the shortest homotopy equivalent path through greedy adjustment of the previous path. However, when communication voids exist in the network, path migration by greedy adjustment based on only 1-hop neighbor information is impossible.

Figure 19 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 19 (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.

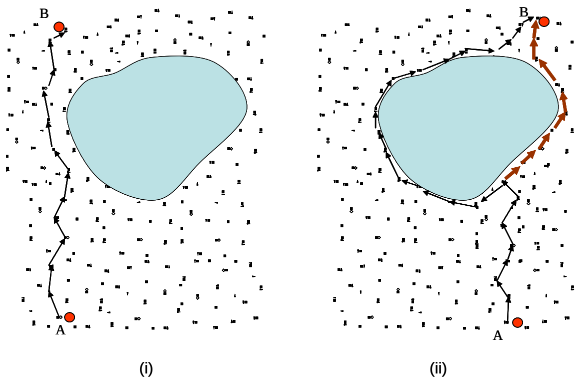

Figure 19. A path migration scenario: (i) a multi-hop communication path is established between two mobile agents A and B, using the sensor network; (ii) as both agents move to the right, the path, shown as the thin line, gets stuck at the the boundary of the hole. The improved path is shown as the bold line.

5.4 Information Storage Mechanisms

Holes can also be used in information storage mechanisms in sensor networks, such as the geographic hash table (GHT) [14] and DIMENSIONS [3]. 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. 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.

6 Discussions

6.1 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 detour compared with the shortest path to the destination [12]. 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 backtracking [7, 8] 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.

6.2 Impact of Non-uniform Transmission Range

In the previous sections we assume that the transmission ranges of the wireless nodes are uniform. If the transmission ranges are non-uniform, the efficient routing problem in general becomes extremely hard. Indeed, if the radii of the communication coverage differ, the communication graph is no longer undirected. For example, non-uniform communication range will cause the planar graph in the GPSR protocol to be disconnected, and thus delivery is not guaranteed. For non-uniform communication range, we let two nodes claim each other as 1-hop neighbor only when both can hear from each other. The BoundHole algorithm will find a cycle back to the original stuck node, although we can no longer guarantee that greedy routing with the help of the holes can always get a packet to its destination if there does exist a path.

Acknowledgement The authors wish to thank Alon Efrat and the anonymous reviewers for their suggestions in improving the presentation of this paper.

References

[1] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and methods for mobile computing and communications (DialM '99)*, pages 48–55, 1999.

[2] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *Proc. 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*, May 2005.

[3] D. Ganesan and D. Estrin. DIMENSIONS: Why do we need a new data handling architecture for sensor networks? In *Proceedings of the ACM Workshop on Hot Topics in Networks*, pages 143–148, Princeton, NJ, USA, October 2002. ACM.

[4] 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*, pages 45–55, 2001.

[5] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom)*, pages 243–254, 2000.

[6] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11 th Canadian Conference on Computational Geometry*, pages 51–54, Vancouver, August 1999.

[7] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and methods for mobile computing and communications*, pages 24–33, 2002.

[8] F. Kuhn, R. Wattenhofer, and A. Zollinger. Worst-case optimal and average-case efficient geometric ad-hoc routing. In *Proc. 4th ACM Symposium on Mobile Ad Hoc Networking and Computing*, pages 267–278, 2003.

[9] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th annual international conference on Mobile computing and networking (MobiCom)*, pages 120–130, 2000.

[10] X. Li, G. Calinescu, P. Wan, and Y. Wang. Localized delaunay triangulation with application in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed System*, 14(10):1035–1047, October 2003.

[11] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, October 2002.

[12] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, November 2001.

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

[14] 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 *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 78–87, 2002.

[15] J. Urrutia. Routing with guaranteed delievery in geometric and wirelss networks. In I. Stojmenovic, editor, *Handbook of Wireless Networks and Mobile Computing*, pages 393–406. John Wiley & Sons, 2002.

7 Appendix

7.1 Proof that Property 3.1 Implies Property 3.2

Lemma 7.1. *If three consecutive nodes $t_{i-2}t_{i-1}t_i$ in the sequence generated by* BoundHole *algorithm have Property 3.1, then they have Property 3.2.*

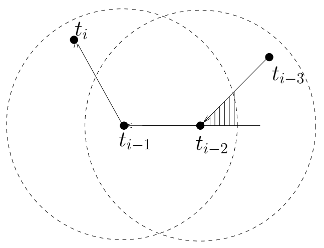

Figure 20. If $t_{i-2}t_{i-1}t_i$ have Property 3.1, then $\angle t_{i-2}t_{i-1}t_i \leq \pi/3$.

Proof: Notice that node t_{i-1} is within distance 1 from both t_i and t_{i-2}. If the angle $\angle t_{i-2}t_{i-1}t_i < \pi/3$, then the node t_i must be inside the communication range of node t_{i-2}. This contradicts with Property 3.1. □

7.2 Proof of Lemma 3.6

Property 3.1 is true for t_2, the original stuck node, since there are no nodes inside the fan defined by the angle $\angle t_k t_0 t_1$ which is greater than $2\pi/3$. So t_2 is either outside the communication range of p or the angle $\angle t_0 t_1 t_2$ is greater than π.

7.3 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 $\angle t_{i-1}t_i t_{i+1}$ is less than π, as shown in Figure 21. 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 $t_{i-2}t_{i-1}t_i t_{i+1}$. 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.

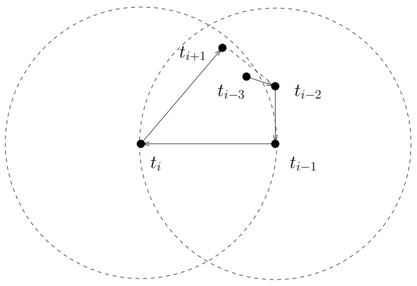

Figure 21. If t_{i+1} is inside the communication range of t_{i-1}.

7.4 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 $t_j t_{j+1}$ and edge $t_i t_{i+1}$, $i < j$. With the same argument as in Theorem 3.5, 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 inside the communication range of 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 22 (i); or, t_j is inside the communication range of t_i but t_j is also inside the forbidden region of t_i, as shown in Figure 22 (ii).

Figure 22 (i) implies that Property 3.1 is true for the three consecutive nodes $t_i t_{j+1} t_j$. In addition, we claim that $t_{i-1} t_i t_{j+1}$ also satisfies Property 3.1. Assume otherwise, then the angle $\angle t_{i-1} t_i t_{j+1}$ is less than π and t_{j+1} is inside the communication range of 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 $\triangle t_{i-2} t_{i-1} t_{j+1}$, 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 22 (ii), since t_j is inside the forbidden region of t_i, then t_j must be inside the communication range of 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 $t_{i-2} t_{i-1} t_i$ satisfies Property 3.1.

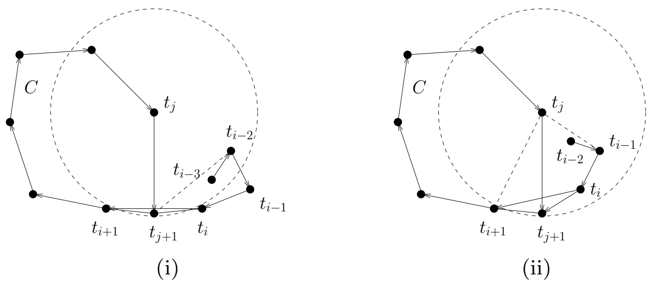

Figure 22. (i) t_j is not inside the communication range of t_i; (ii) t_j is inside the communication range of t_i but t_j is inside the forbidden region of t_i.

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 $t_0 t_1 \cdots t_j t_{i+1} t_i$ satisfies Property 3.1. Thus the Lemma 3.8 is proved.