Distributed Hole Detection Algorithms for Wireless Sensor Networks

Pradipta Ghosh
Bhaskar Krishnamachari
Ming Hsieh Department of Electrical Engineering
University of Southern California
{*pradiptg, bkrishna*}*@usc.edu*

Jie Gao
Computer Science,
Stony Brook University
jgao@cs.stonybrook.edu

Andrea Gasparri
Dipartimento di Ingegneria
Universit degli studi "Roma Tre"
gasparri@dia.uniroma3.it

***Abstract*—We present two novel distributed algorithms for hole detection in a wireless sensor network (WSN) based on the distributed Delaunay triangulation of the underlying communication graph. The first, which we refer to as the distance-vector hole determination (DVHD) algorithm, is based on traditional distance vector routing for multi-hop networks and shortest path lengths between node pairs. The second, which we refer to as the Gaussian curvature-based hole determination (GCHD) algorithm, applies the Gauss-Bonnet theorem on the Delaunay graph to calculate the number of holes based on the graph's Gaussian curvature. We present a detailed comparative performance analysis of both methods based on simulations, showing that while DVHD is conceptually simpler, the GCHD algorithm shows better performance with respect to run-time and message count per node.**

I. Introduction

A Wireless Sensor Network (WSN) is a group of small devices with sensing, wireless communication and computation ability, intended to monitor or gather information in many diverse situations. A WSN can be applied to a wide range of fields like environment monitoring, industrial operation management, health management and surveillance. There are many important and practical challenges in WSN such as scalable deployment, self-organization, routing, localization and clustering. Over last few decades, many researchers across the world have significantly contributed to each of these sub-domains [1]. However, most of these works assume uniformity in the distribution of the network over the region of interest, which is far from practical distributions. Due to non-uniformity of practical distribution, some regions can lack sufficient number of sensors. Moreover, there can also be regions that are completely deprived of sensors due to reasons like presence of obstacle, node failure and worm/hole attack. We denote these regions as *"holes"* in the network. The appearance of such holes make the sensor network organization difficult. Fang, Gao and Guibas have presented a definition of a hole or communication void as a face of a random directed graph (RDG) with 4 or more vertices [2].

There exist many situations that result in creation of holes in the WSN and the holes are also classified according to these situations. The most important kind of hole is *coverage hole* [3]. It can be created by improper node placement or physical obstacle. Among other kind of holes, routing holes [2], holes due to jamming signal [4] or malicious attack on nodes [5] are important. All these kind of holes can result in inefficiency or even hamper task completion in the network. Also, jamming holes or sink holes possess the potential threats of privacy and integrity compromise of the WSN. To mitigate these problems, many researchers have tried to either identify and patch holes or to prevent their creation. We make a contribution to the literature on this problem by providing two novel distributed techniques for the detection and counting of holes in the network. These algorithms can be used periodically as a distributed service to monitor the status of a network over a period of time.

In particular, both algorithms are graph based approaches and can be divided in two parts. While the first part, building a distributed Delaunay triangulation, is shared by both methods, the second part is very distinct. The first method, that we refer to as distance-vector hole determination (DVHD) algorithm, employs a novel weighting model for shortest distance path calculation between any pair of nodes[1] and leverages the path distance as metric for identifying the holes' boundaries. The second method, that we refer to as the Gaussian curvature-based hole determination (GCHD) algorithm, is based on distributed curvature calculation using Gauss-Bonnet theorem in discrete settings. We also provide a complete comparative analysis of both algorithms with respect to correctness, runtime analysis and message complexity via simulations.

II. Related Work

The research on problems concerning holes in sensor networks is relatively recent. In the following, an overview of the current state of the art is provided. In [2], Fang, Gao and Guibas presented an analytical model of a hole in sensor networks. Based on that model they proposed two distributed algorithms called Tent rule and BoundHole to find holes in a sensor network using the geographical location of the nodes. Connectivity information of the underlying communication graph topology without any locational information of the nodes was the core of the hole detection algorithm

[1]In this paper we interchangeably use the terms *node* and *vertex*.

proposed by Funke [6]. Later on, they published another improved method [7] with focus on correctly recovering geometric information from the connectivity graph. Yao *et al.* [8] used connectivity information gathered from one-hop neighbors for coverage hole detection in distributed manner. Yan, Martins and Decreusefond [9] presented a technique for detecting coverage hole using a connectivity based technique. They used two simplicial complexes, Rips complex and Cech complex, to model the network around hole and then used connectivity graph and concept of the Hamiltonian cycle to find nodes in the hole boundary. In [10], Ghrist and Muhammad presented an algebraic method of hole counting called homology. The work of Wang *et al.* [11] on identifying boundaries is also mentionable. Compared to these works which are developed for static networks, our algorithms are focused on mobile networks with geographical coordinates.

Huang and Tseng [3] are among the few researchers to present a method to solve the coverage hole problems. Wang *et al.* [12] proposed a heterogeneous network architecture containing both mobile and static sensor nodes to mitigate the coverage hole problem. They used static sensors to detect the holes and the mobile sensors to cover the holes to the greatest extent. Wood, Stankovic and Son presented technique for mapping and identifying jammed area in a wireless sensor network [4]. Unlike these methods, we do not restrict our algorithms to certain types of holes. However, like most of the hole detection algorithms, the core idea of our hole counting techniques is based on identifying distinct boundaries of the underlying communication graph.

III. Distributed Hole Detection

In this section, we describe our proposed methods for distributed hole detection. These methods rely on the Delaunay triangulation of the underlying communication graph of the WSN. Prior to detailed description of our methods, we define some terms used in the algorithms. Note that, the definitions of holes are based on the availability of a triangulation. In previous works, when a triangulation is not available the definition of a hole is by itself a research question.

Definition 1. Distinct Boundary: *Two hole boundaries based on the triangulation will be called* distinct boundaries *if and only if they have no common edges.*

Definition 2. Boundary Cycle: *A boundary cycle is the sequence of nodes in a distinct boundary.*

Definition 3. *A Delaunay triangulation is closed if the number of distinct boundaries is greater than 1 for a network with holes.*

The Delaunay triangulation is a special kind of triangulation of the convex hull of a set of points. The most important property of the Delaunay triangulation is the *'Empty Circle'* property which implies that no other point or vertex should be inside a circumcircle of a Delaunay triangle, except its vertices. Another important property is *'Planar Graph'* property which states that no two edges of the graph should cross each other. In this scenario, any Delaunay edge's length should also be less than or equal to the communication range of the nodes. Furthermore, the Delaunay triangulation of a network with holes should be *'closed'*. In [13], Bruck, Gao and Jiang presented a method for the Delaunay triangulation in a practical network containing multiple nodes. In their work, they assume that the nodes have accurate angle calculation capabilities. They show that all the short Delaunay edges, i.e., edges whose length are within the communication range, can be found in a distributed manner with only information from the neighbors. This way each node/vertex has only a local knowledge of the Delaunay triangulation.

Now, we briefly describe the steps of our distributed Delaunay triangulation technique. The first task of each node/vertex is to identify its one-hop neighbors and make a list of all possible triangles obtained with any two of the neighbors. Then based on the *'Empty Circle'* property, *'Planar Graph'* property, each node removes the non-Delaunay Triangles from its triangles list. For these purposes, we rely on angle information rather than coordinates. In this context, if two triangles, say $\triangle ABC$ and $\triangle ABD$, share an edge and this edge satisfies the local Delaunay property, then the sum of angles $\angle ACB$ and $\angle ADB$ is less than π. The next important task is to deliver the information about a triangle removal to the other nodes/vertices of that triangle, so that they can also remove it from their lists. By doing so, neighboring nodes will have unambiguous view of the Delaunay triangulation. But this restricts the parallelism to some extent, because we need synchronization between the neighbors to make sure that a node can run this algorithm only if none of its one-hop neighbors is running it. This problem can be solved using proper scheduling such as token ring approach where a node will run this algorithm only if it possesses the necessary token ring.

One major requirement for this algorithm is that either each node knows its own location or each node is capable of accurate calculation of incident angle between any two of its one-hop neighbors. As a result of this algorithm, each node has a local view of the global Delaunay triangulation. That is all we need for our hole detection techniques. Based on this information, we can also determine the number of distinct boundaries that includes node $\{i\}$, say $\mathcal{DH}_i$. Given the number of Delaunay neighbors, $\mathcal{DN}_i$, and the number of Delaunay triangles, $\mathcal{DT}_i$, $\mathcal{DH}_i = \mathcal{DN}_i - \mathcal{DT}_i$. Next, we discuss the second part of our algorithms in details. Prior to that we introduce few more definitions as follows.

Definition 4. Boundary edge/link: *In the triangulation if an edge is part of only one triangle i.e no two triangles have this edge as common edge, then we refer to it as* boundary edge.

Definition 5. Interior vertex/node: *A vertex of the triangulation is an* interior vertex *if none of its adjacent edges is a boundary edge.*

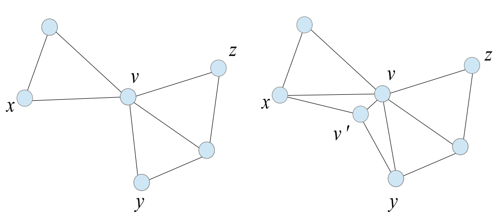
Definition 6. Boundary vertex/node: *A vertex of the triangulation is a* boundary vertex *if at least one of its adjacent edges is a boundary edge.*

Definition 7. Pinch vertex/node: *A vertex of the triangulation is a* pinch vertex *if it is a boundary vertex and part of more than one distinct boundaries.*

A. Distance Vector Hole Determination (DVHD)

This method is based on the shortest distance path between a pair of nodes in the weighted Delaunay graph of the underlying communication model. It uses traditional Bellman Ford algorithm for the shortest path calculation. The core of this method is the link/edge weighting model and modified distance vector table. We assign a weight of '1' to all the *boundary edges* and some high constant weight K (greater than number of nodes, N) to rest of the edges. Ideally, the value of K can be assumed as ∞ to make the algorithm independent of N. The edges in this context are obviously the Delaunay triangulation edges. The structure of the distance vector table is also modified to include a separate column (called *boundary flag*) to discern between *boundary vertices* and *interior vertices*. Similarly, the information shared between neighbors also include information about the destination being a *boundary* or an *interior* vertex.

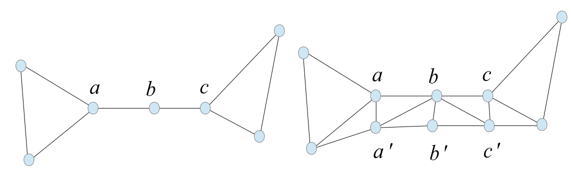
Now, based on this model, the shortest path distances between every pair of vertices is calculated. The *decision policy* used in DVHD can be described as follows. *If the shortest path distance between two boundary vertices is less than K, we say they are on the same boundary. If the shortest path distance between two boundary vertices is greater than or equal to K, we say they are on different boundaries.*



(a) (b)

Figure 1. Solving the 'pinch vertex' problem

This decision policy and algorithm will lead to an erroneous calculation if there exists a *pinch vertex*, as in Figure 1. Since a *pinch vertex* is part of more than one hole boundary cycles, the vertices on two different boundary cycles will try to find shortest part through the *pinch vertex* e.g., the distance between vertex x and vertex z will be 2 ($x \rightarrow v \rightarrow z$) even though they are in the different boundaries. To solve this problem, firstly, we identify a *pinch vertex*, say v, of the Delaunay triangulation. Then for each extra boundary it is part of, we add a dummy node v' and assign the constant weight, K, to the edge between node v and v'. Also connect the dummy node to the neighbors of the *pinch vertex* which are on the same boundary the vertex v' is now part of. Figure 1b clearly demonstrates the concept. Then assign a weight of '1' to those new edges. To this end, we need to distinguish which edges belong to the same boundary at vertex v. This problem can be solved straightforwardly by exploiting the angle information, i.e., sorting the edges clockwise and matching adjacent boundary edges. In general, it is also possible to have a chain of nodes as in Figure 2a. We fix this problem similarly by taking duplicates as in Figure 2b. Then we apply the previously mentioned weighting scheme on the modified graph. Now, we run the shortest path algorithm on the modified graph and apply the decision policy described before to get the number of distinct boundaries. The nodes on the same boundary use a unique identifier to identify the boundary which they are part of. This unique id can be chosen in many different ways such as taking the smallest node id in the boundary cycle(including the dummy node ids) to be the boundary cycle id. Each node can determine the smallest node id locally, by exploring its distance vector table. Next, based on application, they flood the entire network with this hole identifier to inform the rest of the network about the boundary's existence. After flooding is complete, every node will have same information, that is the number of distinct boundary identifiers in the network which is same as number of holes. The convergence of the distributed Bellman Ford algorithm is shown in [14].



(a) (b)

Figure 2. Solving the 'Chain vertex' problem

B. Gaussian Curvature-based Hole Determination (GCHD)

The Gauss-Bonnet theorem [15] is an elegant way to connect geometry, specifically the concept of curvature, to the Euler characteristics of a topological surface. Given the number of holes $\mathcal{Z}$ and number of handles $\mathcal{H}$, the total curvature of a surface M is a topological invariant: $C_{total} = 2\pi\chi(M)$ where $\chi = 2 - 2 * \mathcal{H} - \mathcal{Z}$. The Gauss Bonnet theorem has a discrete version for triangulated 2-manifolds, i.e., triangulations with no two boundary cycles sharing any common vertices.

Although, the Gauss-Bonnet Theorem is applicable to any triangulation, we focus only on *"topological"* triangulation for ease of implementation. The term *"topological"* is used to describe triangulations that need no knowledge of the edge lengths. For a triangulated graph, C_{total} can be expressed as $\sum_{v_i \epsilon V} C(v_i)$ where $C(v_i)$ is the curvature for vertex $\{v_i\}$. To simplify the curvature calculations, we artificially assign each edge a weight of 1. It implies that all the angles of the triangles are $\pi/3$.

Definition 8. Corner Angles: *A* corner angle *of a node v is the interior angle of a Delaunay triangle attached to it. Node v must also be the vertex of the angle.*

The curvature of an interior vertex $\{v\}$, is $C(v) = 2\pi - \sum_i \theta_i(v)$, where $\theta_i(v)$ refers to the i^{th} corner angles that v is involved. The curvature of an boundary vertex $\{v\}$ that stays on only one boundary cycle, is $C(v) = \pi - \sum_i \theta_i(v)$, where $\theta_i(v)$ refers to the i^{th} corner angles that v is involved. In our settings, all the corner angles are assigned a value of $\pi/3$. Thus for an interior vertex v, the curvature is $C(v) = 2\pi - k(v) \cdot \pi/3$, where $k(v)$ is the number of triangles that v is involved. Similarly, for a boundary vertex v that stays on only one boundary cycle, the curvature is $C(v) = \pi - k(v) \cdot \pi/3$, where $k(v)$ is the number of triangles that v is involved.

The triangulation we obtain as mentioned above may have multiple boundary cycles sharing common vertices thereby violating the property of a 2-manifold. One way of handling this is to add dummy vertices as is done in [15]. Here we are not going to maintain the dummy vertices but rather directly count/modify the curvature definition. Therefore, if a vertex v stays on multiple boundary cycles, as in Figure 1, we will imagine adding a dummy node v' to restore the property of a 2-manifold. Also connect the dummy node to the neighbor of the pinch node in the same way as mentioned in the DVHD algorithm. From a mathematical standpoint, for the case in Figure 1, we are adding an extra $-2\pi/3$ at vertex v, an extra $-\pi/3$ at vertex x and vertex y each, and an extra $\pi - 2\pi/3$ at vertex v'. This gives a total of extra $-\pi$ for the curvature. So for each agent that is part of more than one boundary, the modified Gaussian curvature will be $C(v) + (n(v) - 1) * (-\pi)$ where vertex$\{v\}$ is part of $n(v)$ distinct boundaries. Similarly, if there exists a chain of nodes as in Figure 2a, it should be fixed by taking duplicates similar to Figure 2b. Again the duplicated nodes do not need to be explicitly added. Each duplicated node contribute to an extra π. Suppose the chain has length m, there will be m duplicated nodes. All the corner angles of the dummy triangles contribute to $-2m\pi$. Thus the total extra curvature we need to add would be $-m\pi$. To solve this problem in a distributed manner, firstly we identify the nodes that form a chain (say, length= m). Then each node of the chain adds $-\pi$ to its local curvature calculation which results in a addition of $-m\pi$ to the network's curvature in total.

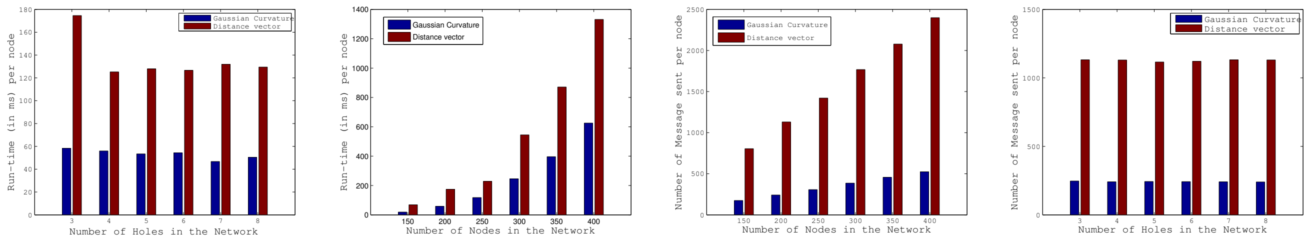
To use the Gauss-Bonnet theorem for hole detection we need to add all the curvatures, $C(v)$. The calculated sum is equal to $2\pi(2 - h)$, where h is the number of holes. So if there is only one hole (the outer surrounding of the network is also a large hole) the sum becomes 2π. Thus for counting the number of internal holes, we need to use $C_{total} = 2\pi(1 - h)$. To calculate C_{total} in distributed manner, each vertex floods its own Gaussian curvature. After flooding is complete, each vertex just add the curvature values received, to get the C_{total}. An alternative method is to use some consensus technique [16] to converge to the average curvature value in the network and multiply it by N. However, simple flooding is used to avoid the asymptotic convergence problems of average consensus algorithms.

IV. Simulation & Analysis

In this section, we evaluate the performance of both GCHD and DVHD algorithms in a range of different scenarios. The simulations are performed on MATLAB 8.1 in a Windows 8.1 computer with a 3.40 GHz processor and 12 GB RAM. We have taken average of 20 simulations for each plot. Note that, in the comparative simulations we do not consider the flooding part of the algorithms as it provides similar outcomes for both algorithms.

In Figure 3a, we plot the variation of the average simulation run-time per node with increasing number of holes in a network with 200 nodes. From the simulation, it is clear that the runtime remains almost the same or rather it decreases with increasing number of holes for a network with fixed number of nodes. The decrease of the algorithm runtime can be explained by the fact that the number of boundary nodes is likely to increase when the number of holes increases. So the average number of neighbors per node decreases which thereby slightly reduce the average calculation. Another important point to note in Figure 3a is that the time taken by the DVHD algorithm is always larger than the GCHD algorithm. The extra time required in the DVHD algorithm is due to the distance vector algorithm's runtime to find the shortest paths. We present the variations of average run-times required per node with increasing number of nodes in Figure 3b. The invariants in this plot are the number of holes in the network, which is set to be 3, and the area per node. It shows that, with increasing number of nodes the time required also increases. In addition, it can be noticed that the time required by the DVHD algorithm is always larger than the GCHD algorithm due to same reason as before.

The simulation also points out that the number of message exchanges per node in the network increases with increasing number of nodes in the network. This is clear from Figure 3c, where the number of holes is equal to 3. However, the average number of message remains almost same with the increasing number of holes, as shown in Figure 3d. In both cases the number of messages required in the DVHD method is greater than the GCHD method due to the shortest path calculations. However, one of the major advantages of the DVHD algorithm over the GCHD algorithm is that it is



(a) (b) (c) (d)

Figure 3. Performance plots (a) run-time with increasing number of holes (b) run-time vs increasing number of nodes (c) number of messages sent per node with increasing number of nodes (d) the number of messages sent per node with increasing number of holes

independent of the number of nodes (N) in the network, provided $K = \infty$ (or very high value compared to any feasible N).

In summary, according to the simulation results, the GCHD algorithm outperforms the DVHD algorithm in almost every aspect, except simplicity of implementation and independence from number of nodes in the network. However, the independence of DVHD from number of nodes in the networks, is a huge advantage in terms of scalability and dynamic nature of the algorithm.

V. Conclusion and Future Work

We have presented two novel methods for hole detection in WSN called DVHD and GCHD. Both of them are based on the topological information of the underlying communication model. A detailed comparative analysis of these methods is also presented in this paper. There is still some work left to do with our proposed method including theoretical analysis of the message complexity and practical implementation on a real WSN testbed. Another future direction may be to flesh out similar analysis in case of mobile wireless sensor networks along with analysis of robustness and fault tolerance. Also our current simulations rely on the integrity of the underlying communication model. The performance analysis of our methods on dynamic scenarios with some probability of link failures is another direction yet to be considered.

Acknowledgment

This project is funded in part through NSF award numbers 1217260 and 1201198. Jie Gao would like to acknowledge the support of NSF through CNS-1116881, CNS-1217823, DMS-1221339, DMS-1418255.

References

[1] N. Trigoni and B. Krishnamachari, *Sensor Network Algorithms and Applications : Philosophical Transactions of the Royal Society A*, 2012, vol. 370 (1958).

[2] Q. Fang, J. Gao, and L. J. Guibas, "Locating and bypassing holes in sensor networks," *Mobile Networks and Applications*, vol. 11, no. 2, pp. 187–200, 2006.

[3] C.-F. Huang and Y.-C. Tseng, "The coverage problem in a wireless sensor network," *Mobile Networks and Applications*, vol. 10, no. 4, pp. 519–528, 2005.

[4] A. Wood, J. A. Stankovic, and S. H. Son, "Jam: A jammed-area mapping service for sensor networks," in *24th Real-Time Systems Symposium.* IEEE, 2003, pp. 286–297.

[5] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Wormhole detection in wireless ad hoc networks," *Department of Computer Science, Rice University, Tech. Rep. TR01-384*, 2002.

[6] S. Funke, "Topological hole detection in wireless sensor networks and its applications," in *Proceedings of the 2005 joint workshop on Foundations of mobile computing*. ACM, 2005, pp. 44–53.

[7] S. Funke and C. Klein, "Hole detection or: how much geometry hides in connectivity?" in *22nd SoCG*. ACM, 2006, pp. 377–385.

[8] J. Yao, G. Zhang, J. Kanno, and R. Selmic, "Decentralized detection and patching of coverage holes in wireless sensor networks," in *SPIE Defense, Security, and Sensing*, 2009.

[9] F. Yan, P. Martins, and L. Decreusefond, "Connectivity-based distributed coverage hole detection in wireless sensor networks," in *IEEE GLOBECOM*, 2011, pp. 1–6.

[10] R. Ghrist and A. Muhammad, "Coverage and hole-detection in sensor networks via homology," in *IPSN 2005*. IEEE Press, p. 34.

[11] Y. Wang, J. Gao, and J. S. Mitchell, "Boundary recognition in sensor networks by topological methods," in *Proceedings of 12th MobiCom*. ACM, 2006, pp. 122–133.

[12] G. Wang, G. Cao, P. Berman, and T. F. La Porta, "Bidding protocols for deploying mobile sensors," *Mobile Computing, IEEE Transactions on*, vol. 6, no. 5, pp. 563–576, 2007.

[13] J. Bruck, J. Gao, and A. A. Jiang, "Localization and routing in sensor networks by local angle information," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 1, p. 7, 2009.

[14] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data networks*. Prentice-Hall International, 1992, vol. 2.

[15] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu, "Greedy routing with guaranteed delivery using ricci flows," in *IPSN 2009*, pp. 97–108.

[16] D. Bauso, L. Giarre, and R. Pesenti, "Distributed consensus in networks of dynamic agents," in *44th IEEE CDC-ECC*, 2005, pp. 7054–7059.