A Distributed Algorithm for Monitoring an Expanding Hole in Wireless Sensor Networks

Khanh-Van Nguyen
Hanoi University of Science and Technology, Vietnam
E-mail: vannk@soict.hust.edu.vn

Phi Le Nguyen
Department of Informatics, The Graduate University for Advanced Studies, Japan
E-mail: nguyenle@nii.ac.jp

Hau Phan
Hanoi University of Science and Technology, Vietnam
E-mail: pvhau93@gmail.com

Trong D. Nguyen
Iowa State University, USA
E-mail: trong@iastqae.edu

Keywords: wireless sensor networks, hole detection, expanding hole

Received: April 1, 2016

Holes in sensor networks are regions that have no operating nodes and that may occur due to several reasons, including cases caused by natural obstacles or disaster suffered areas. Determining the location and shape of holes can help to monitor these disasters (such as volcano, tsunami, etc.) or help to make smart, early routing decisions for circumventing a hole. There are many hole determination algorithms proposed in the literature, however, these only consider the networks with static holes i.e. with stable boundary nodes. Moreover, most of these are designed in a centralized manner which is not suitable to the unstable situation of networks with an expanding hole. In this paper, we propose an algorithmic scheme not only for determining the initial shape but also for monitoring and quickly reporting about the area of a hole gradually expanding. Our algorithms are designed in a distributed manner and our initial simulation results show that our protocol is lightweight and feasible with monitoring sensor networks with an expanding hole.

Povzetek: Razvita je nova metoda za obravnavo lukenj v brezžičnih omrežjih.

1 Introduction

Let start by considering an example scenario of a forest wherein fire rather frequently occurs in summer. Since it is very large and difficult to access and monitor by human, one thinks of using helicopters to disseminate thousands of wireless sensor nodes through out this forest area for deploying a fire monitor system. Due to the cost limitation, these pieces are equipped with just a temperature and humidity sensing device, a very simple processing unit and a small radio communication module that allows them to communicate with the others in their communication range. The mission of these sensor nodes is to monitor the environment, detect the fire and report to a central host via some special sink nodes (with more powerful resource). Paradoxically possibly when the fire occurs, the sensor nodes in the area of an on-going fire can be destroyed and thus cannot perform their monitor task and fail to report about the fire. To solve this problem, a common approach is letting the alive nodes surrounding the fire areas detect the boundary of the fire area fast enough and report to the central host. This scenario belongs to a well-known problem in sensor network which is called *hole detection and determination.*

The hole detection-determination (HDD) problem is not only used to detect the events that are being monitored as described above but also is, and in fact known more as, an important technical issue in geographic routing. Geographic routing [1][2] which exploits the local geographical information at the sensors is popular for its simplicity and efficiency. The traditional geographic routing strategy works well and can achieve the near-optimal routing path length in networks without holes, however with the occurrence of network holes, the path length can grow as much as $\theta(c^2)$ where c is the optimal path length [3]. In order to solve the problem of path enlargement, a common approach is to determine the hole boundary, describe it by a simpler shape whose information is disseminated to the surrounding area. This information will help to establish a hole awareness and mechanism to find short detour routes [4][5][6][7][8].

There are many HDD algorithms proposed however, these just consider the problem of detecting a typically static hole (hole whose boundary never changes). Moreover, almost of these algorithms are conducted in the centralization manner and is time ineffective. A very common approach is based on the possibility of *having a monitor packet to travel a full route around the hole, which could be unlikely possible in our problem scenario – the network hole can be expanding fast enough that could destroy any attempt to send a packet around in one simple loop*.

In this paper, we propose a novel approach to detect and regularly report about the hole boundary, that to the best of our knowledge is *the first one that targets expanding holes* (whose boundary change continually). Especially, we aim at scenarios where holes can occur and grow rapidly; i.e., we aim to design fast algorithms for detecting and monitoring hole boundaries. Our main idea is actually to distribute this task of capturing the dynamic hole perimeter into the hands of several boundary nodes and network cells that are carefully selected to take this joint responsibility.

In our protocol, the network is divided into small grid squares by a given grid. The squares (cells) are to be divided into three groups: black, gray and white. A black square is one which already belongs to or is heavily affected by a hole, a gray square is not yet touched by any hole but is detected being close enough and getting soon affected by an expanding hole, and a white is a currently safe one. A white square can later get turned into a gray and then a black square when facing an expanding hole, which can be seen as a set of the black squares. Each square in the black and the gray group is monitored by a local node, called a pivot, which is elected from the sensors in that square. These pivots monitor the status of the sensors and the squares and help to manage the square status changes.

The contribution of this paper are threefold:

- We propose a distributed hole detection algorithm which can determine a hole boundary efficiently with respect to time and energy saving.

- We propose a distributed hole updating algorithm which monitors the expansion of the hole boundary and reports regularly to the central host.

- We conduct initial experiments by simulation to analyze our protocol and evaluate its feasibility and advantages.

The remainder of the paper is organized as follows. Our hole detecting and updating protocols are proposed in section 2. Section 3 evaluates the performance of our protocol using simulation experiments. We also discuss further on related work in section 4 and finally conclude the paper in section 5 with further concerns about open issues.

A preliminary version of this paper appears in [12]. Besides a significant revision effort for improving the clarity, formality and preciseness we also add new substantial elements to this full version. That is we reshape and adjust some important parts in our proposed algorithm and thus make it more efficient; as a result, we redo the evaluation task but in a larger scale for obtaining a more insightful about the performance of our algorithm. Below we briefly mention the most important adjustments. In section 2.1, for the task of detecting a hole boundary we justify our decision to choose the approach used in [10] over the one in [26]. In section 2.2 we optimize the way we define and use Bitmap Presentation for reducing incurred communication overhead and delay. In 2.3 we tune the process of forwarding the hole boundary info towards the sink(s): the pivots fully involve in this process that increases the efficiency of the whole mechanism.

Compared to the previous version, moreover, we also extend our evaluation work to a significant deeper level where we redesign a new, significant larger set of simulation scenarios, inspired by new observations and thus obtain new findings. Most notably, our 3 main simulation settings (for studying the effects of 3 system parameters: Dead Node Threshold, Notification Threshold & Report Threshold) are all extended by our new deployment where we use two separate scripts to simulate an expanding hole: the Fast Expansion and the Slow Expansion scripts. We focus more on the Fast Expansion scenario (a hole expands fast for simulating a forest fire) and we identify some value region of the Dead Node Threshold that could optimize our algorithm. We also extend our study on these main parameters with some initial consideration of the relationship with another variable that is the grid cell size. In spite of this rather extensive evaluation analysis there still remain many unknowns that can be challenging enough for good results in future work.

2 Our hole monitor scheme

2.1 Scheme overview

Our goal is to detect and update the hole as fast as possible so that our hole monitor scheme can beat the expanding speed of a hole. Therefore, we aim to determine the approximate shape of the hole boundary rather than its exact shape. The approximate shape of the hole boundary is determined via a set of the unit squares of a given grid (with certain predefined unit length) which are affected by the hole, i.e. ether intersecting the hole boundary or staying inside the hole.

Our hole monitor scheme consists of two protocols: the hole boundary detection protocol, denoted by the HBD protocol, and the hole boundary update protocol, denoted by the HBU protocol. Let us discuss the basics of the HBD protocol first. The main idea is to have some nodes on the hole boundary detect that they are on a hole's boundary. This initial awareness helps to start the process of learning about the whole hole shape by arranging a special monitor packet to travel around the hole: this packet is being forwarded from one boundary node to another as a neighbor.

Technically, we know of two main approaches in detecting a hole boundary, one from the research works on geographic routing in WSNs e.g. in [10], and one from the research work on coverage hole e.g. [26]. We have done certain probing experiments and found out that the former approach appears more efficient (we show our experimental evaluation on this in section 3) and thus, we choose to use this approach for designing our HBD protocol where we aim at approximating the hole shape in a distributed manner.

Based on the above mentioned approach, our HBD protocol is conducted by using the *stuck nodes* introduced in [10]: by definition, a node u is considered a stuck node if there exists a position w outside u's transmission range such that u is closer to w than all the neighbor nodes of u [1]. Note that a stuck node always stays on a hole boundary but not all the boundary nodes are stuck node. Each stuck node can detect its status as being stuck (using the TENT rule [10]) and then creates a hole-monitor packet, which contains a HBA message (denoted for Hole Boundary Approximation), and forwards it to the next neighbor, boundary node (determined by the right hand rule [10]). The mission of this HBA message is to record the approximate shape of the path it has gone through: at each boundary node, the HBA gets updated about the new, just traveled edge. Upon reaching the next stuck node, a HBA message has fulfilled its mission by capturing the approximate shape of the boundary segment between this pair of successive stuck nodes, which will be forwarded to the nearest sink node (via a cell pivot node that we will introduce later). Note that each boundary segment (a chunk between two successive stuck nodes) is determined and approximated by using a separate HBA message. The sink node combines information received from all these stuck nodes (via the pivot nodes) and sends the whole info to the central node.

We now discuss the HBU protocol. Because the holes can enlarge quickly we want to have each update action performed as fast as possible to reflect well with the changes. Therefore, instead of determining the changes after happening we predict and update the changes of the hole boundary based on the prediction. Specifically, we monitor the status of the nodes (i.e. of being alive or dead) in the unit grid squares (also called cells) around the hole boundary. When the amount of the dead nodes of a given cell exceeds a predefined threshold, this cell is predicted to belong the hole soon and thus the hole area gets updated by adding that cell. To make the mechanism distributed, the task of monitoring within each unit square is performed by a so-called pivot node of this cell. The cell pivot is a sensor node (by default, the one closest to the center of the square) that is elected from all the nodes locating inside the cell.

During the HBD protocol, when a HBA message is about to fulfill its mission (and then stop being forwarded), the receiving stuck node can initiate the election of the pivot node of its cell if not selected yet, then forward the HBA info to this pivot.

[1]Thus, a task of forwarding to destination w would gets 'stuck' at node u if greedy routing is being used

2.2 Bitmap representation

The main idea of our approximation mechanism is to describe the status of the cells (unit grid squares) in the network area by a bitmap representation where each bit would reflect if a corresponding cell belongs to a hole or not. More specifically, let $m\&n$ denote the width and length of the network; we use a grid with the edge length (of the unit squares) a that divides the network into $([\frac{m}{a}]+1)\times([\frac{n}{a}+1])$ cells. We use a two dimension array $bmp[.][.]$ to represent the status of the cells, where $bmp[i][j]$ corresponds to the unit grid square whose center has the coordinates $(1/2+i)a$; $(1/2+j)a$ and $bmp[i][j] = 1$ if the corresponding cell belongs to a hole (i.e. this cell stays inside this hole or intersects its boundary) or else, $bmp[i][j] = 0$.

During the execution of the HBD protocol, the chunks of a hole's boundary (boundary segments between two successive stuck nodes) are approximated and these approximation info pieces are recorded into the HBA messages, which then are later forwarded to the sink(s) for being combined. To facilitate this computing mechanism, below we formally define bitmap representations as data structures to keep data at the HBA messages as well as at the pivots and the sink(s).

Definition 1 (Bitmap representation)**.** *For a line segment l on the plane, the bitmap representation of l is defined as a two dimension array $bmp_l[.][.]$, where $bmp_l[i][j] = 1$ if and only if l intersects the unit grid square whose center's coordinates are $((1/2+i)a, (1/2+j)a)$; $bmp_l[i][j] = 0$, otherwise.*

We call the bitmap representation of the whole network a two dimension array bmp, where $bmp[i][j] = 1$ if and only if the unit grid square whose center's coordinates $((1/2+i)a, (1/2+j)a)$ belongs to a hole and $bmp[i][j] = 0$, otherwise.

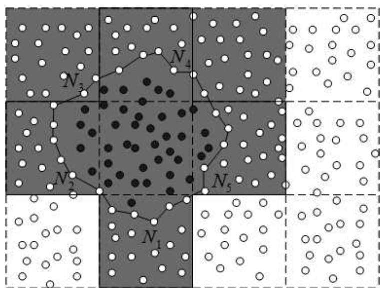


Figure 1: Illustration of bitmap representation

Fig. 1 illustrates a network with a big hole. The unit grid squares which intersect the hole are colored black. The bitmap representation of line segment N_1N_2 is

$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ and the bitmap representation of the network is $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.

In the HBD algorithm, the bitmap representations of the boundary segments are determined by using HBA messages and sent to the local cell pivots, which forward these info pieces to the sinks. During the HBU algorithm, the pivots of the newly alarmed cells (who have been alarmed about the expanding hole after the HBD's execution) keep monitoring the hole and reports its status to the sinks as a black cell (belonging to the hole) when the number of the dead nodes inside exceeds a predefined threshold. The sinks maintain their bitmap representations (as about the network from their viewpoints) and periodically report them to the central node, which can combine these pieces into the bitmap representation of the whole network by simple bitmap XOR operations.

2.2.1 Forwarding mechanism with compact hole info

The main goal of our scheme is to monitor the hole(s) and update the sink(s) with any new status of the network shape. Our algorithm scheme is fully distributed where several nodes independently and concurrently capture info about pieces of a hole's boundary and forward towards the sink(s) via the cell pivots – the intermediate hubs. The bitmap presentations of the network are only formed at the sink(s), however “pieces" of that being formed and forwarded at node- and pivot- levels, at which sending a hole bitmap would be too much a luxury to afford (by a piece we mean a chunk of adjacent nodes on the hole boundary). In fact, the bitmap presentation of a line segment, or even a full hole's boundary (a collection of segments) can be compactly described, i.e. specifically, stored into memory as follows. Starting with an end vertex of the line segment (or any of the hole boundary polygon) we store the full coordinates (indexes) of its cell and then continually check the adjacent vertices for newly separate cells, per each of which store just two bits for describing its relative position to the preceding cell [2]. By this mechanism, even a complex full hole boundary can be compactly described (as for reflecting the bitmap presentation of the network) by just the coordinates of a cell plus a rather short binary string (just a few bytes) as illustrated in 2.

2.3 Hole boundary detection algorithm

This section discusses further details of our HBD algorithm.

At the initial, all the network nodes detect if each is a stuck node by using the TENT rule described in [10]. Each stuck node then creates its HBA message and sends

[2]It only needs two bits to describe 4 possible directions that may involves left, right, up and down

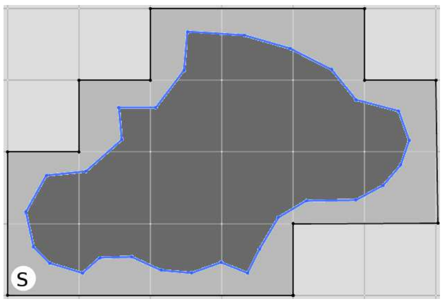


Figure 2: Compact Hole Info
This hole can be reflected by the coordinate of cell S plus a string of 24 bits to locate the remaining 12 cells at the hole boundary.

it to the left boundary neighbor node (using the right hand rule [10]) which then forwards it towards the next stuck node. Here we use the forwarding mechanism as suggested in [10] but the HBA message is customized to reflect bmp_s – the bitmap representation of the boundary segment between the two successive stuck nodes. When a node N on the hole boundary receives the HBA message, it determines the cells intersecting the line segment connecting N and the previous boundary node by the algorithm suggested in [11] and updates bmp_s and the HBA message accordingly. The details are described in algorithm 1.

In the deployment of our HBD algorithm, to reduce size of data transferred between nodes, we do not store in the HBD packet this full bitmap representation but instead use the compact form as discussed in section 2.2.1, which would take just a few bytes in a HBA packet.

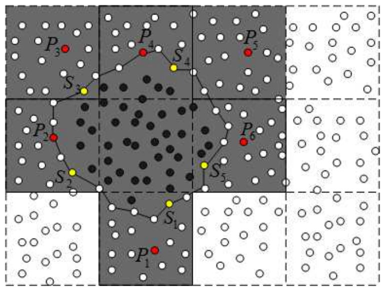


Figure 3: Illustration of hole boundary detecting algorithm

To prepare for the hole boundary update (HBU) phase, the HBD algorithm not only detects the hole boundary but also determines the pivots of the cells intersecting the hole boundary. Such a pivot is determined by the following algorithm. Each node u receiving a HBA message determines the cell(s) which intersect the line segment connecting u and the previous boundary node v but does not contain v; there may exist none, one, or more such cell; for each such a cell if exists, u sends a pivot election message towards this cell's center. Note that, such a message then stops at the node w where none of its neighbors is nearer to the cell center than w, i.e w can see it as elected as the cell pivot. Each pivot then broadcasts a pivot announcement

Algorithm 1 Algorithm to update compression form of bitmap representation bmp_{comp}

```
Require: P: HBA packet; $N_c(x_C, y_C)$: current node and its coordinate; $N_P(x_P, y_P)$: previous node and its coordinate
Ensure: updated $bmp_{comp}$
1: $bmp_{comp} \leftarrow$ the compression form from packet $P$
2: $tmp \leftarrow$ transfering $bmp_{comp}$ into bitmap
3: $U \leftarrow$ set of the unit grid squares which intersect $N_C N_P$
4: for all the unit grid square $U_i(x_i, y_i)$ in $U$ do
5:     $tmp[\lfloor \frac{x_i}{a} \rfloor][\lfloor \frac{y_i}{a} \rfloor] \leftarrow 1$
6:     if $(\lfloor \frac{x_i}{a} \rfloor \neq \lfloor \frac{x_P}{a} \rfloor)$ $and$ $(\lfloor \frac{y_i}{a} \rfloor \neq \lfloor \frac{y_P}{a} \rfloor)$ then
7:         Send a election packet to $U_i(x_i, y_i)$.
8:     end if
9: end for
10: $bmp_{comp} \leftarrow$ transfer $tmp$ into compression form
11: update $P$ with new $bmp_{comp}$
```

message to all the nodes in the cell.

Fig. 3 illustrates the algorithm. In the figure, the yellow nodes $S_1, S_2, \ldots, S_5$ found themselves as stuck nodes using the TENT rule. They initiate and forward the HBA messages along the boundary (using the right hand rule to determine the next boundary neighbor) until each arriving at the next stuck node. So, the HBA message initiated by S_i is forwarded until arriving at S_{i+1} wherein this message captures the bitmap representation of the line segment of the boundary between S_i and S_{i+1}. For example, the bitmap representation of the segment between S_1 and S_2 is $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$. After the HBD algorithm finishes, the approximate shape of the hole as well as the pivots of the cells surrounding the hole are determined. In Fig. 3, the approximate shape of the hole is colored black and the pivots are colored red.

2.4 Hole boundary update algorithm

We discuss further details on the HBU protocol. As mentioned above, se call an unit grid square, i.e. a network cell, belonging to a hole (i.e. staying inside the hole or intersecting its boundary) a *black* one. After the HBD phase finishes, each black cell has a pivot elected. All nodes in these black cells notify their status (i.e. whether alive or not) to its cell's pivot periodically.

When the ratio of the number of the dead nodes (over the total) of this cell exceeds a predefined threshold, which we call *Notification Threshold*(NTT), the situation with this expanding hole is seen serious. Thus, the cell's pivot sends an alert message to each of the four neighbor cells which share an edge with it to notify about the high possibility that they would soon get affected by the hole, i.e. these four are considered *gray* (becoming black). Note that some of these neighbors may have already been black cells wherein this alert simply gets ignored. Also, the destination of such an alert is set to be the center of the targeted gray cell and thus, the last node receiving the alert is elected as the pivot of this gray cell.

When the ratio of the number of dead nodes in a black cell exceeds a predefined threshold, which we call *Report Threshold* (RPT), the cell is considered severely damaged by the hole and thus, this dangerous state gets reported to the nearest sink by the cell pivot.

Similarly to the pivot of a black cell, the pivot of a gray cell also monitors the status of all the nodes within the cell. When the ratio of the number of the dead nodes of this cell exceeds the NTT, the cell situation gets serious and thus, it turns into a black one. At this same time the cell's pivot sends an alert to its four neighbor cells (notification) to make any white remainder of them to become *gray*. The details of these algorithmic operations executed at the white, gray, and black cells are described in algorithms 2a, 2b and 2c, respectively.

Fig. 4 illustrates an example. In this figure, the network is divided into 4×3 unit grid squares. The black nodes are dead and others are alive. After conducting the HBD algorithm, the hole boundary has been detected and the cells that belong the hole are colored black as shown in Fig. 4(a). The red nodes $P_1, P_2, \ldots, P_6$ represent the pivots of the black cells. Suppose that, the hole is expanding to the right, then after sometime some nodes in square (6) die, which is enough to make pivot P_6 send an alert to the neighbor squares as shown in Fig. 4(b). Among these neighbors, (0) and (5) are already black which ignore the alert; the others i.e. squares (7) or (8) are still white, so become gray. The cell center nodes P_7 and P_8 then become the pivots of (7) and (8), respectively – Fig. 4(a). When the ratio of the number of dead nodes in (7) exceeds NTT, it is considered black and P_7's pivot sends alerts to the neighbor cells – Fig. 4(d).

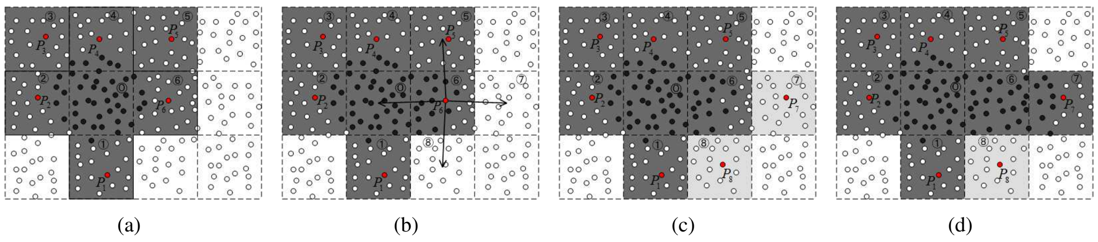


Figure 4: An example of hole boundary updating process

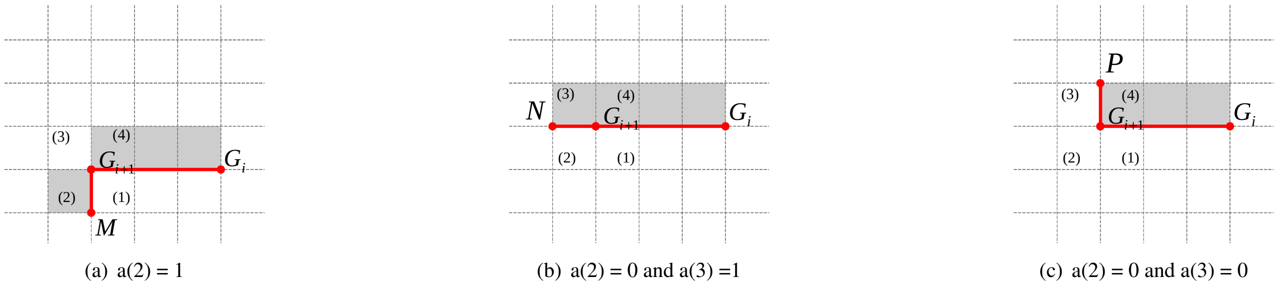


Figure 5: Algorithm to determine hole boundary from bitmap representation

Algorithm 2a HBU process at nodes in a white cell ($state = WHITE$) when receives a message

```
Require: $P$: received message, $C$: the current node
Ensure:
 1: if $P$ is an alert message then
 2:     $state \leftarrow GRAY$
 3:     if $C$ is the closest node of center then
 4:         $C$ takes responsibility of the cell pivot
 5:         Broadcast a notification to all nodes in the cell
 6:     else
 7:         if $C$ has not received a pivot notification then
 8:             Forward $P$ to the center of cell (Vote the closest
                node of center as the pivot node)
 9:         end if
10:     end if
11: else if $P$ is a notification message then
12:     Be informed that the source of message as the
        elected pivot
13: else
14:     Send HELLO packet
15: end if
```

Algorithm 2b HBU work at nodes in a gray cell ($state = GRAY$)

```
Require: $C$: current node
Ensure:
 1: loop periodically
 2:     if $C$ is a pivot then
 3:         $d \leftarrow$ the number of dead nodes in the cell
 4:         if $d > NTT$ then
 5:             $state \leftarrow BLACK$
 6:             Send alert packet to 4 neighbor cells
 7:         end if
 8:     else
 9:         Send HELLO packet to update node's state
10:     end if
11: end loop
```

Algorithm 2c HBU work at nodes in a black cell ($state = BLACK$)

```
Require: $C$: current node
Ensure:
 1: loop periodically
 2:     if $C$ is a pivot then
 3:         $d \leftarrow$ the number of dead nodes in the cell
 4:         if $d > RPT$ then
 5:             Report to the nearest sink
 6:         end if
 7:     else
 8:         Send HELLO packet to update node's state
 9:     end if
10: end loop
```

2.5 Hole information combination

At the central node, we obtain the bitmap representation of the whole network. Although this bitmap gives us a view of location of the holes, in some applications (e.g. in geographic routing) geometric expressions (i.e. polygonal shapes) of the holes are required. We now describe an algorithm to determine the boundary of a hole approximate shape as a polygon, called an A-polygon, which satisfies the following condition:

- This A-polygon vertices are the vertices of the grid squares which intersect the hole
- All the centers of the unit grid squares belonging to the hole stay inside this A-polygon

We will compute the (coordinates of) the vertices of this A-polygon in the clockwise order. The following fact helps to show the process during each step for obtaining the next vertex; this fact is quite simple so we omit the proof.

In fact 1, we consider the scenario where we have just determined that G_i and G_{i+1} are two consecutive vertices of the A-polygon (in the clockwise order), and we need to determine the next vertex. Let (1), (2), (3), (4) denote the unit squares that has G_{i+1} as a vertex and let M, N and P denote the adjacent vertices of G_{i+1} in (2), (3) and (4), respectively as shown in Fig. 5. Let $(x_i,\ y_i)$ be the coordinates of the center of square (i) and $\alpha(i)$ be the corresponding bit value of square (i) in the network bitmap, then:

Fact 1. *Having G_i and G_{i+1} just determined as the vertices of an A-polygon from the given network bitmap, the next vertex of this A-polygon (in the clockwise order) can be determined as follows:*

- *M if and only if $\alpha(2) = 1$ (Fig. 5(a)).*
- *N if and only if $\alpha(2) = 0$ and $\alpha(3) = 1$ (Fig. 5(b)).*
- *P if and only if $\alpha(2) = 0$ and $\alpha(3) = 0$ (Fig. 5(c)).*

Using fact 1, we come up with algorithm 3 below to determine an A-polygon of a hole from the network bitmap. It is easy to observe that for a given hole the vertex of the unit square (intersecting it) with the lowest y-coordinate must be a vertex of the A-polygon.

Algorithm 3 Determining A-polygon from array bmp

```
Require: bitmap $bmp$; $(x_A, y_A)$: coordinates of boundary
    node A that has lowest $y - coordinate$
Ensure: $G$: the set of vertices of the A-polygon
1: Denote $r$ as the edge length of the unit square
2: $G \leftarrow G \bigcup \left(\lfloor \frac{x_A}{r} \rfloor r + r, \lfloor \frac{y_A}{r} \rfloor r\right)$
3: $G \leftarrow G \bigcup \left(\lfloor \frac{x_A}{r} \rfloor r, \lfloor \frac{y_A}{r} \rfloor r + r\right)$
4: while the top element of $G \neq \left(\lfloor \frac{x_A}{r} \rfloor r + r, \lfloor \frac{y_A}{r} \rfloor r\right)$ do
5:     $U \leftarrow$ the top element of $G$
6:     $(x_u, y_u)$ is the coordinate of U
7:     $V$ is the previous element of U
8:     $(x_v, y_v)$ is the coordinate of V
       //the following code is for the case when $y_u = y_v$,
       the other case ($x_u = x_v$) is similar
9:     $i \leftarrow \lfloor \frac{x_v}{r} \rfloor; j \leftarrow \lfloor \frac{y_v}{r} \rfloor - 1$
10:    if $bmp[i-1][j] = 1$ then
11:        $G \leftarrow G \bigcup (x_v, y_v - r)$
12:    else
13:        if $bmp[i-1][j+1] = 1$ then
14:            $G \leftarrow G \bigcup (x_v - r, y_v)$
15:            $G \leftarrow G \setminus (x_v, y_v)$
16:        else
17:            $G \leftarrow G \bigcup (x_v, y_v + r)$
18:        end if
19:    end if
20: end while
```

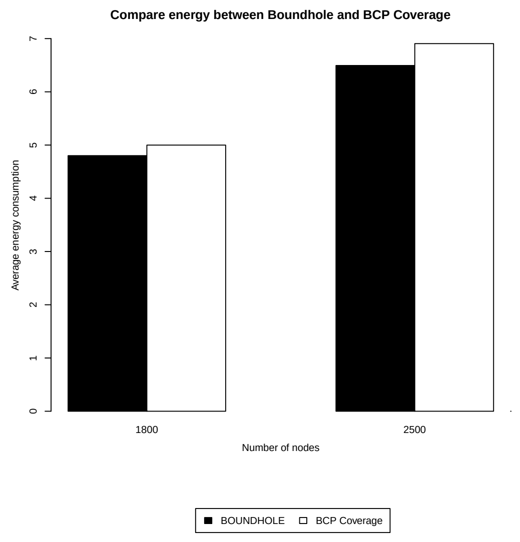
3 Performance evaluation

3.1 Comparison between the approaches in detecting hole boundary

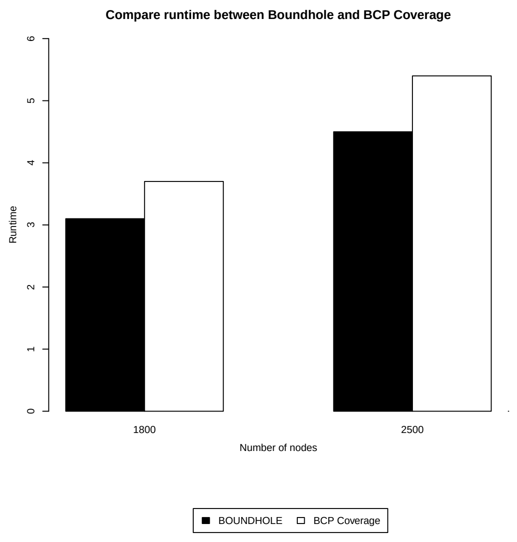
As we mentioned above in section 2.1, we now compare by simulation experiments between the two HBD mentioned approaches, i.e. the Boundhole approach from [10] and the BCP approach from [26] to find out which suits better to our main goal: we need our HBD algorithm to be as fast as possible and to consume energy as less as possible (to help with monitoring expanding holes). [3] In this evaluation, we run the experiments with 2 network models: 1800 nodes and 2500 nodes. Table 1 summarizes the parameters which are suggested by [13]:

The results of experiments are evaluated by these metrics: average consumed energy and running time (the lower is better). We can see in Fig 6, the Boundhole approach gives better result with slightly less consumed energy and running time. It appears that the Boundhole uses simpler operations while the BCP uses a bit too many floating-point operations (especially, when computing circle-circle intersections).

[3]Note that here we do not compare the original algorithms from the two mentioned papers, instead we compare two versions of our HBD algorithm each of which is based on each of these two approaches.



(a) Comparison of consumed energy



(b) Comparison of running time

Figure 6: Comparison between the BoundHole and the BCP approaches

Variables	*Values*
Communication range	40m
Voltage	3V
Receiving current	15mA
Transmitting current	29.5mA
Idle current	3.2mA
Number of nodes	29.5mA
Period of HELLO packet	30s
Simulation time	1050s

Table 1: Simulation parameters for comparing the BoundHole and BCP approaches

3.2 Experiment deployment and settings

We have proposed our very first algorithmic solution to deal with a possibly new and challenging problem of hole expansion in WSN. Bellow we show our initial results in evaluating our algorithm through experiments by simulation. Here we propose to use the following metrics to evaluate our algorithm performance:

– The *Approximation Ratio* (AR) is the ratio between the area that our algorithm approximately describes of a hole and the true area of this hole.

– The *Death Report Error* (DRE) reflects how well we monitor the dead nodes, which is calculated as $\frac{|RN-TN|}{TN}$ where RN is the number of nodes reported as dead by our algorithm and TN is the true number of the dead nodes.

– The *Consumed Energy* (CE) of network which is computed as the average energy consumed by any node in the network.

More specifically, we study the effect of choosing the following parameters, crucial in our algorithm, in achieving good performance by the above mentioned metrics.

– *Dead Node Threshold* (DNT): All the alive nodes periodically sends the HELLO packets to their neighbor nodes, hence in our algorithm after waiting for a time period called DNT if a node does not receive the next HELLO from a certain neighbor, this node can decide that this neighbor is dead.

– *Notification Threshold* (NTT): As described in section 2, the pivot of a black/gray square decides to send a NOTIFY packet to each of its 4 neighbor squares (to notify about the hole expansion) when the ratio of the number of dead nodes (per the total) in its square exceeds a NTT.

– *Report Threshold* (RPT): The pivot of a black square decides to send a REPORT packet to the nearest sink when the ratio of the number of dead nodes in this square exceeds a RPT.

We run the experiments on the ns-2 simulator (802.11 as MAC protocol). Table 2 summarizes the parameters which are suggested by [13]. In our simulation, about 2500 sensor nodes are deployed randomly in an area of $1400m \times 1400m$.

We conduct three different simulation settings to evaluate the effect of these parameters on three metrics described above. Also in our simulation settings, we deploy two different scripts of a large hole expanding: one is called Fast Expansion (EF), where we focus on, and the other

Variables	*Values*
Communication range	40m
Voltage	3V
Receiving current	15mA
Transmitting current	29.5mA
Idle current	3.2mA
Network size	$1400m \times 1400m$
Number of nodes	2500
Period of HELLO packet	30s
Simulation time	1050s for FE
	2500s for SE

Table 2: Simulation parameters for our protocol experiments

Slow Expansion (SE). In each script we program so that the hole gradually expands from a chosen center point toward a fixed given shape, which is a random set of points with distances to the center varying randomly from a minimum value (300 m) to a maximum value (450 m). The hole reaches its maximum, final shape in 1000 seconds in our FE script, and 2500 seconds in our SE script. We believe that these scripts would reflect reasonably to the expansion of a typical forest fire.

3.3 Simulation results

1. *Effect of Dead Node Threshold:* We conducted the simulation with four values of the Dead Node Threshold (DNT): 1x, 1.3x [4], 2x, 3x, where nx means the DNT equals n times of the HELLO period (at which the HELLO packets being periodically sent from a node to a neighbor).

Fig. 7 and 8 show the simulation results of our first simulation settings (with the Fast Expansion script) on the effect of DNT on the Death Report Error (DRE) and the Approximation Ratio (AR), respectively. It can be seen that the DRE tends to decrease with the increasing of the time values during the simulation. That is, for all the different DNT values, the trend is that the performance keeps improving (DRE gets closer to 0% and AR gets closer to 1) along the the simulation process (better performance for captures at later simulation moments - larger timestamps).

With the case of 1x, the DRE is very large at first: it can be explained as because the HELLO packets and responses may get delayed or stuck (and dropped) at some nodes; therefore, when the threshold is too short, the nodes whose HELLO packets are delayed or dropped are wrongly judged as dead nodes (so DRE very large, initially). However later the HELLO packets (and its responses) can be retransmitted and thus, the nodes which have been judged as dead can be reconsidered as alive and so the DRE decreases.

[4]A HELLO packet may be sent a bit later than expected due to some random delay in processing; thus, we take this into account by using DNT= 1.3x.

The simulation results also shown that $FN - TN$, the difference between the number of reported dead nodes and the true number of dead nodes, tends to decrease when the DNT increases. Especially, with the DNT $=$ 2x or larger $FN - TN$ is even negative, i.e. the number of the dead nodes determined by our algorithm is less than the true number of real dead nodes. This is because, when the DNT increases, the pivots tend to become dead too early to send alerts about its cell situation to the neighbor cells, thus many dead nodes may not be recognized. It can be seen that, the DRE in case of 1.3x is closest to 0 and thus, parameter DNT $=$ 1.3x can be considered (near) the best for maintaining low DRE.

Fig. 8 represents the relation between the approximation ratio (AR) and the DNT. Similarly as with the DRE, the AR decreases with the increasing of the DNT. The AR with the DNT $=$ 1x is quite greater than 1 and the AR with the DNT as 2x/3x is often/always (much) smaller than 1. It can be seen that the hole area determined by our algorithm is always much larger than the real hole with the DNT as 1x and always much smaller than the real hole with the DNT $=$ 3x (or quite smaller with 2x). For the later phase of hole expansion, both 1.3x and 2x seem competing for the best AR (but in both sides of 1). Overall, it can be seen that a small enough DNT value may lead to a high DRE because of many alive nodes being wrongly judged as dead while a large enough DNT value may make unreasonably small AR because of many dead nodes being unrecognized. For our experiment setting with the FE script, the DNT$=$ 1.3x seems (near) the optimum option.

So far we have just analyzed the simulation with the FE script, where the considered hole expands to the maximum distance $450m$ (from the center) in only $1000sec$, we now take a quick look at the results with the SE script (reaching maximum distance in $2500s$). As can be seen in table 3 and table 4, the general trend is that the performance keeps improving (DRE gets closer to 0% and AR gets closer to 1) along the the simulation process for most of cases. Most notably, the choice of DNT$=$ 2x results in poor AR while still quite good at keeping low DRE. For keeping AR close to 1, the ideal DNT value should be between 4x and 5x, however with this range the DRE is rather high (compared to that of DNT$=$ 2x). This is because for a larger DNT the network acts slower with the hole expansion and hence, a number of new dead nodes cannot get reported fast enough. Thus, in fact for the SE script we face a complexity in choosing good DNT (further work needed in future work).

2. *Effect of Notification Threshold:* The effect of the NTT on the Consumed Energy (CE) and Approximation Ratio (AR) is shown in Fig. 9. In this simulation setting with the FE script we fix DNT$=$ 1.3x (as seen best from simulation results above). Another parameter needs taken into account is the size of the unit grid square which we choose within $80m - 120m$ (ranges in small multiples of the sensor transmission range, $40m$).

It is clear that, the consumed energy becomes lesser for

	Death Report Error (%)			
Timestamp - S.moment (s)	1x	1.3x	2x	3x
560	44.94	17.98	35.58	75.66
620	27.49	15.41	37.76	76.13
680	17.17	12.37	30.3	71.46
740	13.95	10.52	33.26	72.75
800	7.33	11.54	34.07	72.89
860	1.3	10.73	32.03	77.89
920	7.05	7.78	33.92	73.57
980	1.49	6.79	28.13	65.9
1040	3.72	4.52	21.94	60.51

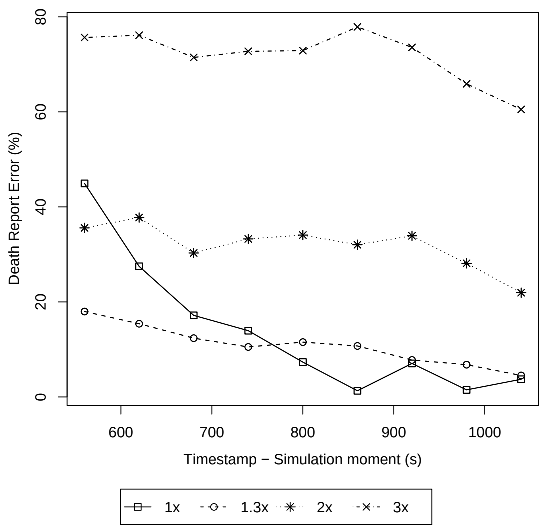


Figure 7: Effect of DNT on the Death Report Error

	Approximation Ratio			
Timestamp - S.moment (s)	1x	1.3x	2x	3x
560	1.6674	1.1672	0.9588	0.0834
620	1.6422	1.1632	0.9238	0.1711
680	1.4422	1.1826	0.8941	0.2019
740	1.3117	1.089	0.8167	0.2227
800	1.2255	1.141	0.8663	0.2113
860	1.1099	1.1099	0.9218	0.2822
920	1.1422	1.1251	0.8865	0.2898
980	1.1289	1.1289	0.9408	0.3136
1040	1.1815	1.1815	1.0127	0.3529

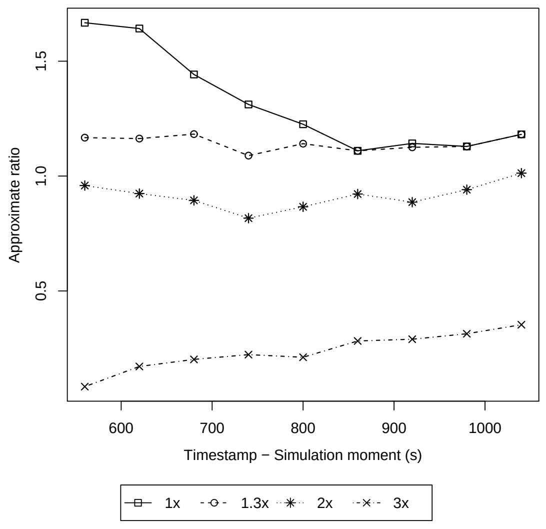


Figure 8: Effect of DNT on Approximation Ratio

larger NTT as well as for smaller grid square (cell) sizes. That is for a larger NTT, the pivots of the black cells need to wait longer to notify its neighbor squares, i.e. the squares become gray later rather than sooner, lessening the monitoring work and hence, reducing the CE incurred due to our monitor mechanism. Also, for smaller cell sizes, obviously the total area being monitored is also smaller and thus, reducing the consumed energy as well.

The effect of the NTT on the AR is reported in Fig. 9(b). It can be seen that the AR decreases with the increasing of the NTT. It can be explained as below. For smaller NTT values, the notification of the expanding hole is done earlier and hence, the gray area tends to get large earlier, i.e. the AR tends to increase. For AR greater than 1, the approximate area of the hole is larger than the real hole area. When the NTT is larger enough (from about 15%) the AR can be smaller than 1 (for grid size 100m or less). This is because the times to send NOTIFY packets may come too late (with a speed slower than the expansion speed of the hole) and thus, many black pivots may get dead before the time to notify their neighbor cells. Consequently, some should-be-black squares get unrecognized and the approximate hole area becomes smaller than the real hole area.

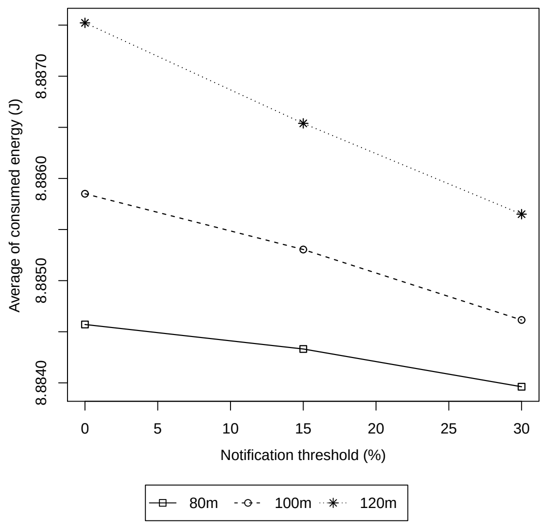
It can be remarked that the NTT should be selected in accordance with the speed of the expansion of the hole. When the hole expands fast, we should choose a small NTT. Par-

	Death Report Error (%)			
Timestamp - S.moment (s)	2x	4x	5x	6x
760	18.75	31.25	39.58	45.83
1000	14.63	28.05	31.71	36.59
1240	15.27	26.72	31.3	38.93
1480	11.29	21.51	25.81	37.1
1720	8.8	20	24.8	35.6
1960	8.18	17.3	22.01	35.53

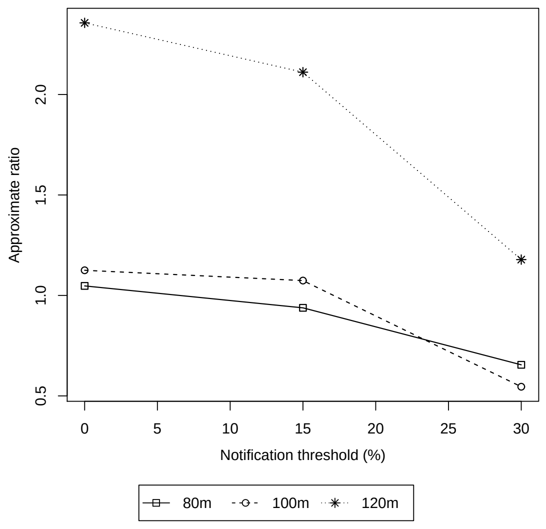
Table 3: Effect of DNT on DRE with the Slow Expansion script

	Approximation Ratio			
Timestamp - S.moment (s)	2x	4x	5x	6x
760	1.6393	1.4572	1.0929	0.7286
1000	1.2247	1.1134	1.002	0.8907
1240	1.3219	1.175	0.9547	0.7344
1480	1.2502	1.1958	1.0327	0.7066
1720	1.2382	1.1144	0.9493	0.7429
1960	1.3046	1.1742	0.9785	0.6849

Table 4: Effect of DNT on AR with the Slow Expansion script



(a) Effect of NTT on Consumed Energy (with 3 different cell size-settings)



(b) Effect of NTT on Approximation Ratio (with 3 different cell size-settings)

Figure 9: Effect of Notification Threshold (NTT) with FE script

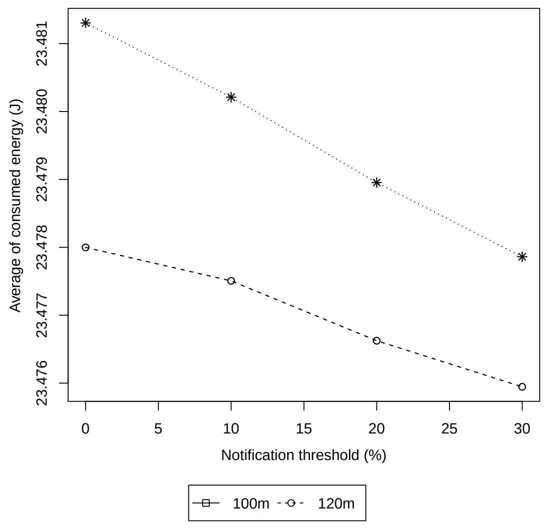
ticularly, in this simulation setting with the FE script, the best NTT is about 15%.

For our simulation with the SE script (see Fig.10), the AR becomes close to 1 for NTT about $25-30\%$, larger than the setting with the FE script, which is an advantage for keeping the gray squares in smaller number. However, the CE looks quite higher to that in the setting with the FE script. This is because the gray/black squares live longer in this SE setting and hence more monitor traffic would get incurred. Of course, one would consider to use a smaller grid size but this would result in need of low NTT which is a disadvantage. Thus, again we face some complexity in choosing proper parameter values that needs further consideration in our future work.

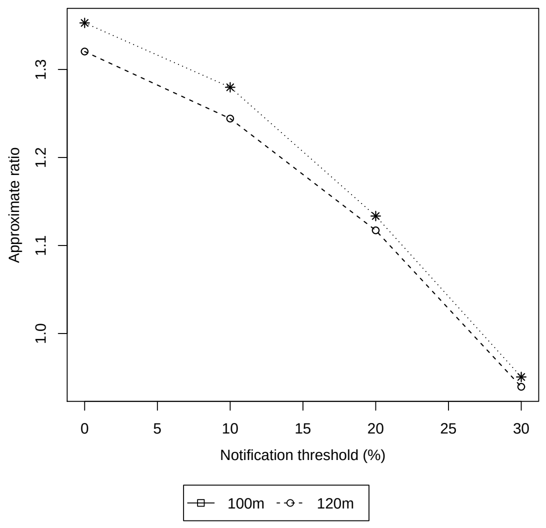
Fig. 11 illustrates the growth of the real hole area (represented by the orange line) and the approximate hole area determined by our algorithm (represented by green line) with images captured at 4 different time values. This also illustrates the same general trend as before: the performance keeps improving (AR gets closer to 1) along the the simulation process.

3. *Effect of Report Threshold:* The simulation result in this aspect is shown in Fig. 12 where we also fix DNT $=$ 1.3x and we evaluate only the AR because the energy consumption is not affected much by the RPT. It can be seen that the AR tends to decrease when the RPT increases; especially if the RPT $\leq$ 30%, the AR becomes $\geq$ 1, i.e. the approximate hole area is larger than the real hole area. However if the RPT is $\geq$ 35%, the AR becomes less than 1.

This is because, for RPT small enough the pivots report to the sink about the approximate hole soon enough (after discovering a small enough fraction of the dead node in the their squares), thus the approximate hole area, which is the area defined by the black squares (called the black area), tends to contain the real hole area and hence, the AR is greater than 1. The smaller is the RPT, the larger is the difference between the black area and the real hole are and

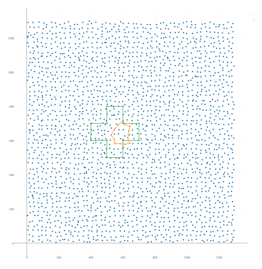


(a) Effect of NTT on Consumed Energy (with 2 different cell size-settings)

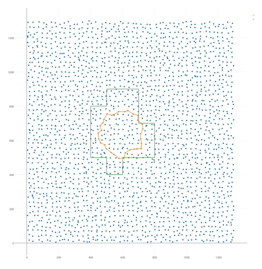


(b) Effect of NTT on Approximation Ratio (with 2 different cell size-settings)

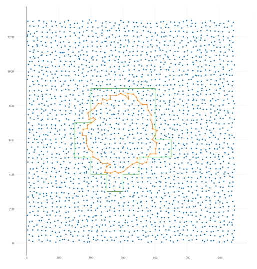
Figure 10: Effect of Notification Threshold(NTT) with the SE script



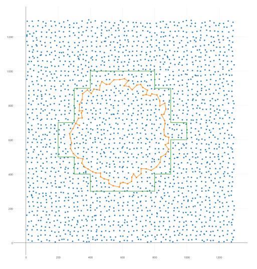
(a) Captured at 120s



(b) Captured at 360s



(c) Captured at 600s



(d) Captured at 840s

Figure 11: The growth of the approx. and the real hole areas
(DNT= 1.3x; NTT= 15%; RPT= 15%)

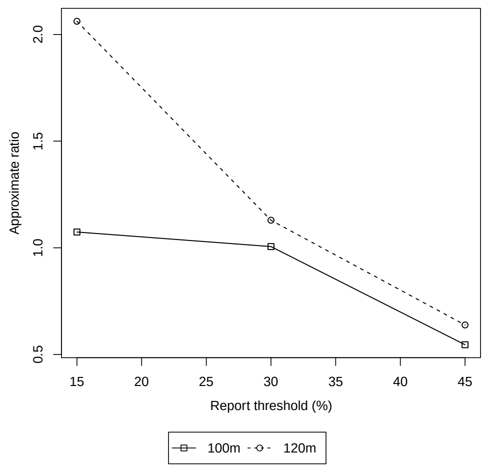


Figure 12: Effect of RPT on the Approx. Ratio

thus, the larger is the AR. However, for RPT large enough (e.g. $\geq 35\%$), some pivots may get dead before the time to send a REPORT packet to the sink and thus, their black squares are not reported to the sink, i.e. the AR becomes less than 1.

Similarly as with the NTT evaluation, it can be concluded that the RPT should be selected in accordance with the speed of expansion of the hole. For example, in this simulation setting, the best RPT is about 30%.

4 Related work

An algorithmic approach for locating holes in WSN has been firstly introduced in [10] where Fang *et al.* propose a procedure to obtain the exact boundary of the hole as a polygon with vertices being adjacent nodes on the hole side. The basic idea is to have a packet traveling around the hole, learning the position of the nodes on its side. Other follow-up works [27, 9, 28] study a generalized problem in computing the exact shape of the network as collection of boundaries, i.e. the outer perimeter of the whole network area and boundaries of inside holes. In [11], we introduce an efficient and flexible approach and techniques to approximate the shape of a hole.

However, this approach is based on the possibility of having a packet to travel a full route around the hole, which could be unlikely possible in our problem scenario – the network hole can be expanding fast enough that could destroy any attempt to send a packet around in one simple loop. Our main solution is actually to distribute this task of capturing the hole perimeter: several HBA packets (sent by the stuck nodes) and later, the cell pivots are responsible to pick up a collection of segments of the hole perimeter.

There are many HDD algorithms have been proposed which can be classified into three categories: geometric methods, statistical methods and topo-logical methods. Geometrical approach uses the coordinates of the nodes and standard geometric tools (such as the Delaunay or Voronoi diagrams) to detect coverage holes and their boundaries. Several simple distributed algorithms have been proposed in [29, 30, 31] to detect the boundary nodes (i.e. *non-covered sensor* in our definitions) and build the routes around holes. In [33], Zhang *et al.* proposed an algorithm to detect the hole boundary nodes on the basis of Voronoi Diagram. The authors also described a method to calculate the accurate location of hole boundary by analyzing the sensing edges of the boundary nodes. In the topological approach, there are neither coordinates nor localization of nodes are required. Instead, this approach use topological properties such as the information of connectivity to identify the boundary nodes; e.g. Ghrist *et al.* [32] proposed a purely connectivity-based coverage hole detection method. Recently, Chu *et al.* [35] exploited information of three-hop neighboring nodes and propose a distributed protocol to identify sensor nodes nearby the hole. In this protocol, the hole is determined on the basis of the so-called *2-hope-neighbor graph* maintained at each sensor node. Although it is said that, the protocol can detect all hole boundaries with a small overhead, it is not suitable for our expanding hole scenario because it requires all nodes running the protocol on their *2-hope-neighbor graph* to detect the expanded hole.

Routing hole is a critical issue in geographic routing where data packets are forwarded based on the positional information of the sensor nodes (assuming that they are equipped with GPS devices). Early approaches are based on greedy and perimeter routing where with the former a packet is forwarded to the 1-hop neighbor that is closest to the destination. However, greedy forwarding can lead into the local minimum phenomenon whenever the packet encounters a routing hole. To bypass such a hole, the traditional schemes appropriately switch between greedy and perimeter forwarding modes (initiated by the GPSR proposal in [2]), in the later of which the data packets are forwarded along the hole boundary [2][14][15][16][17]. These proposals require a specific embedding of a planar graph (e.g. Gabriel Graph). Recently, Yu *et al* [34] proposed a scheme to relieve the local minimum problem by allowing a node in a concave area of a hole to mark itself as a potential stuck node and thus do not participate in data delivery. Through this policy, the packets are prevented from entering the concave area of the hole and thus can avoid the long detour path problem.

There are quite a few papers in geographic routing which propose to learn about the hole shape then disseminate this hole info to the surrounding area for supporting routing tasks performed later. This approach as we call learn-and-disseminate strategy has been used in e.g. [25][5][4], however, the main aim there is to achieve a bounded route stretch in their routing algorithms. As a result, although achieving a desired route stretch, these schemes can incur heavy extra communication in broadcast and significant load imbalance due to congestion on these major routes. For example, in [4], the holes are compactly described as a set of extreme points of the convex hull covering the boundary nodes and thus, the hole detour will be along these points, that may cause the traffic concentration around the boundary of the convex hull.

We briefly review the two mentioned (in section 2.1) approaches in detecting a hole boundary, i.e. the work in [10] for supporting geographic routing in WSNs and the work in [26] for solving the problem of wireless hole coverage. The BOUNDHOLE algorithm [10] can be shortly described as follows. Each stuck node p initiates a HBD message (denoted for Hole Boundary Detection) which includes its location and sends it to p's closest neighbor node with respect to the stuck angle. The closest neighbor can be defined as follows: consider the stuck angle at p (facing the hole area by this angle) if we use the angle's bisector line to conduct counter-clockwise sweeping then the closest neighbor is the first one that is met by the sweeping line. Upon receiving the HBD message, this newly identified hop writes its location into the message and passes it to the next-hop in the same manner mentioned. The HBD message will finally come back to the origin as basically, the HBD message creates a closed cycle of traveling.

The coverage hole detection algorithm in [26] is based on the concept of Boundary Critical Points (BCPs). A boundary critical point is defined as the intersection point of two sensing circles that cannot be covered by any other sensing circles. Each sensor node first computes its BCPs. Note that the sensors that are not along the boundary of any coverage holes cannot have any boundary critical point. Then each consecutive boundary critical points is connected by constructing an boundary line along the border of the nodes having those boundary critical points. The construction of boundary line is continued until the starting boundary critical point is revisited or border of the monitoring region is touched.

5 Conclusion and future work

In this paper we have proposed an algorithmic scheme for detecting, determining and monitoring the shape and boundary of an expanding hole in sensor networks. Our algorithms are designed in a distributed manner to help their surviving of the expanding of a hole and reporting regularly about the hole status.

We have also conducted the simulation experiments to evaluate the effects of the important thresholds (DNT, NTT, RPT) on the performance metrics of our algorithms. The simulation results show that the performance metrics are strongly dependant on how suitably we choose the mentioned threshold parameter for a given network setting. Obviously, the thresholds should be selected in accordance with the characteristics of the network, especially the expansion speed of the considered hole.

It is obvious that although we have done some complicated experiments which require some heavy simulation work there still remain many unanswered interesting questions and important technical issues. There are 3 important threshold parameters yet there still are other important parameters such as the grid square size or the speed of the hole expansion; thus, it seem challenging to mix the selection of these to a good effect. Most challenging, perhaps is the concern that the hole expansion speed is normally unforeseeable and hence, we face a big trouble: a parameter set that is nicely suitable to a fast expansion setting can perform very poorly in a slow expansion setting. Thus, it is natural to think of further work with improved algorithms where the important parameters such as the grid size can be adjusted dynamically during the execution process.

Acknowledgement

This work was partially supported by a grant from Hanoi University of Science and Technology under the project titled "Detecting Holes and Monitoring Their Growth in Wireless Sensor Networks" (PI: Nguyen Khanh Van).

References

[1] Y. Ko and N. H. Vaidya (1998) Location-Aided Routing (LAR) in Mobile ad hoc Networks, *Proc. of MOBICOM'98*.

[2] B. Karp and H. T. Kung (2000) GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *Proc. of MOBICOM'00*, pp. 243–254.

[3] Fabian Kuhn, Roger Wattenhofer, Yan Zhang, and Aaron Zollinger (2003) Geometric ad-hoc routing: of theory and practice, *PODC*, pp. 63–72.

[4] Myounggyu Won, Radu Stoleru, and Haijie Wu (2011) Geographic routing with constant stretch in large scale sensor networks with holes, *Proc. of WiMob*, pp. 80–88.

[5] Z. Zheng, K. W. Fan, P. Sinha, and Y. Wang (2008) Distributed roadmap aided routing in sensor networks, *5th IEEE International Conference*, pp. 347–352.

[6] Y. Tian et al (2008) Energy-Efficient Data Dissemination Protocol for Detouring Routing Holes in Wireless Sensor Networks, *Proc. of IEEE Intl. Conf. on Communications, ICC'08*, pp. 2322–2326.

[7] F. Yu et al (2008) Efficient Hole Detour Scheme for Geographic Routing in Wireless Sensor Networks, *Proc. of the 67th IEEE Vehicular Technology Conference, VTC'08*, pp. 153–157.

[8] M. Choi and H. Choo (2011) Bypassing Hole Scheme Using Observer Packets for Geographic Routing in WSNs, *Proc. of Intl. Conf. on Information Networking, ICOIN'11*, pp. 435–440.

[9] Yue Wang, Jie Gao, and Joseph S.B. Mitchell (2006) Boundary recognition in sensor networks by topological methods, *5MobiCom'06*.

[10] Q. Fang, J. Gao, and L. J. Guibas (2004) Locating and Bypassing Routing Holes in Sensor Networks, *Proc. of INFOCOM'04*.

[11] Le Nguyen, Quan Bui, Hieu Nguyen, and Khanh-Van Nguyen (2011) Efficient approximation of routing holes in wireless sensor networks, *ACM Proc. of the Second Symposium on Information and Communication Technology*.

[12] Trong Nguyen, Le Nguyen, Hau Phan and Khanh-Van Nguyen (2015) A Distributed Protocol for Detecting and Updating Hole Boundary in Wireless Sensor Networks, *ACM Proc. of the Second Symposium on Information and Communication Technology(SoICT 2015)*.

[13] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoffrey Werner-Allen, and Matt Welsh (2004) Simulating the power consumption of large scale sensor network applications, *SenSys*, ACM, pp. 188–200.

[14] P. Bose, P. Morin, I. Stojmenovir, and J. Urrutia (2008) Routing with guaran-teed delivery in ad hoc wireless networks, *5th IEEE International Conference*, pp. 347–352.

[15] F. Kung, R. Wattenhofer, and A. Zollinger (2002) Asymptotically Optimal Geometric Mobile Ad-hoc Routing, *Dial-M*.

[16] F. Kung, R. Wattenhofer, and A. Zollinger (2003) Worst-Case Optimal and Average-Case Efficient Geometric Ad-hoc Routing, *Proc. of ACM MobiHoc*.

[17] F. Kung et al (2003) Geometric Ad-hoc Routing: Of Theory and Practice, *Proc. of ACM PODC*.

[18] S. Subramatian, S. Shakkottai, and P. Gupta (2007) On Optimal Geographical Routing in Wireless Networks with Holes and Non-Uniform Traffic, *Proc. of IEEE INFOCOM*.

[19] Piyush Gupta and P. R. Kumar (2000) The capacity of wireless networks, *IEEE Transactions on Information Theory*, pp. 388–404.

[20] A. Rao, C. H. P., S. Shenker, and I. Stoica (2003) Geographic Routing without Location Information, *Proc. of ACM MOBICOM 2003*, pp. 96–108.

[21] Q. Fang, J. Gao, L.J. Guibas, V. de Silva, and L. Zhang (2005) Glider: Gradient Landmark-based Distributed Routing for Sensor Networks, *Proc. of IEEE INFOCOM*, pp. 339–350.

[22] R. Fonseca, S. Ratnasamy, J.Zhao, C.T. Ee, D.E. Culler, S. Shenker, and I.Stoica (2007) Beacon Vector Routing: Scalable Point-to-point Routing *Wireless Sensornets*.

[23] A. Caruso et al (2007) GPS Free Coordinate Assignment and Routing in Wireless Sensor Networks *INFOCOM*, pp. 150–160.

[24] R. Kleinberg (2007) Geographic Routing Using Hyperbolic Space, *IEEE INFOCOM*, pp. 1902–1909.

[25] G. Tan, M. Bertier, and A.-M. Kermarrec (2009) Visibility-graph-based shortest-path geographic routing in sensor networks, *Proc. of IEEE INFOCOM*, pp. 1719–1727.

[26] Zhiping Kang, Honglin Yu and Qingyu Xiong (2013) Detection and Recovery of Coverage Holes in Wireless Sensor Networks, *JOURNAL OF NETWORKS, VOL. 8, NO. 4, APRIL 2013*, pp. 822–828.

[27] A. Kroller and S. P. Fekete and D. Pfisterer and S. Fischer (2006) Deterministic Boundary Recognition and Topology Extraction for Large Sensor Networks, *Proc. of SODA '06, the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, 2006.

[28] S. Lederer and Y. Wang and J. Gao (2009) Connectivity-based Localization of Large-scale Sensor Networks with Complex Shape, *ACM Transactions on Sensor Networks (TOSN)*.

[29] Yen-Wen Chen Hwa-Chun Ma, Prasan Kumar Sahoo (2011) Computational geometry based distributed coverage hole detection protocol for the wireless sensor networks, *J. Network and Computer Applications*, pp. 1743–1756.

[30] E. D. Zhao and Y. T. Lv J. Yao, H. Wang (2011) A coverage hole detection method and improvement scheme in wsns, *the International Conference on Electric Information and Control Engineering (ICEICE 11)*, pp. 985–988.

[31] S. Commuri M. Watfa (2006) Energy-efficient approaches to coverage holes detection in wireless sensor networks, *the International Conference on Electric Information and Control Engineering (ICEICE 11)*, pp. 131–136.

[32] Robert Ghrist and Abubakr Muhammad (2005) Coverage and hole-detection in sensor networks via homology, *Proceedings of the 4th international symposium on Information processing in sensor networks*, IEEE Press, pp. 34.

[33] Yunzhou Zhang, Xiaohua Zhang, Zeyu Wang, Honglei Liu (2013) Virtual Edge Based Coverage Hole Detection Algorithm in Wireless Sensor Networks, *IEEE Wireless Communications and Networking Conference (WCNC): NETWORKS*, pp. 1488–1492.

[34] Fucai Yu, Shengli Pan, and Guangmin Hu (2015) Hole Plastic Scheme for Geographic Routing in Wireless Sensor Networks, *IEEE ICC 2015 - Ad-hoc and Sensor Networking Symposium*, pp. 6444–6449.

[35] Wei-Cheng Chu, Kuo-Feng Ssu (2012) Decentralized Boundary Detection without Location Information in Wireless Sensor Networks, *IEEE Wireless Communications and Networking Conference: Mobile and Wireless Networks*, pp. 1720–1724.