Fault-Tolerant Breach-Free Sensor Barriers

Jorge A. Cobb

Department of Computer Science
The University of Texas at Dallas
Richardson, TX 75080-3021
U.S.A.
Email: cobb@utdallas.edu

Chin-Tser Huang

Department of Computer Science and Engineering
University of South Carolina at Columbia
Columbia, SC 29208
U.S.A.
Email: huangct@cse.sc.edu

***Abstract*—Consider an area that is covered by a wireless sensor network whose purpose is to detect any intruder trying to cross through the area. Given the limited battery power of wireless sensor nodes, the length of time during which intrusion-detection is possible can be maximized by dividing the sensors into disjoint sets, known as barriers. The area remains protected, or covered, by a sensor barrier if there exists a subset of sensors that divide the area into two regions, such that no intruder can move from one region into the other and avoid detection. By having only one barrier active at any time, the duration of the coverage is maximized. However, sensor barriers may suffer from *breaches*, which may allow an intruder to cross the area while one barrier is being replaced by another. This is dependent not on the structure of an individual sensor barrier, but on the relative shape of two consecutive sensor barriers. Centralized heuristics exist in the literature that separate sensors into breach-free barriers. In this paper, we present a distributed version of the best-performing heuristic for breach-free barriers. In addition to being distributed, the protocol is stabilizing, i.e., starting from any state, a subsequent state is reached and maintained where the sensors are organized into breach-free barriers.**

Keywords–Stabilization; Sensor networks; Sensor barriers.

I. Introduction

We consider a wireless sensor network consisting of a large number of sensor nodes distributed over a geographical area. Each sensor has a limited battery lifetime, and is capable of sensing its surroundings up to a certain distance. Data that is collected by the sensors is often sent over wireless communication to a base station [1].

The type of coverage provided by the sensors is either full or partial. In full-coverage, the entire area is covered at all times by the sensor nodes, and thus, any event within the area is immediately detected [2]–[5]. Partial coverage, on the other hand, has regions within the area of interest that are not covered by the sensors [6]–[8].

One form of partial coverage that received significant attention due to its application to intrusion detection is barrier coverage [9]–[16]. A barrier is a subset of sensors that divide the area of interest into two regions, such that it is impossible to move from one of the regions to the other without being detected by at least one of the sensors. Figure 1(a) highlights a subset of sensors that provide barrier coverage to the area.

In the specific case of intrusion detection, providing full coverage is not an efficient use of the sensor resources, and leads to a reduced network lifetime. Instead, multiple sensor barriers can be constructed, as illustrated in Figure 1(b). Only one barrier needs to be active at any moment in time; the remaining barriers can remain asleep in order to conserve energy. When a barrier is close to depleting all of its power, another barrier is placed in service. Given a set of sensors deployed in an aera of interest, finding the largest number of sensor barriers is solvable in polynomial-time [11].

Sensor barriers are susceptible to a problem, known as a barrier-breach, in which it is possible for an intruder to cross an area during the time that one barrier is being replaced by another [17], [18]. The existence of a barrier-breach is dependent not on the structure of an individual sensor barrier, but on the relative shape of two consecutive sensor barriers. The complexity of obtaining the largest number of breach-free sensor barriers is an open problem. Thus, heuristics have been presented in [17], [18].

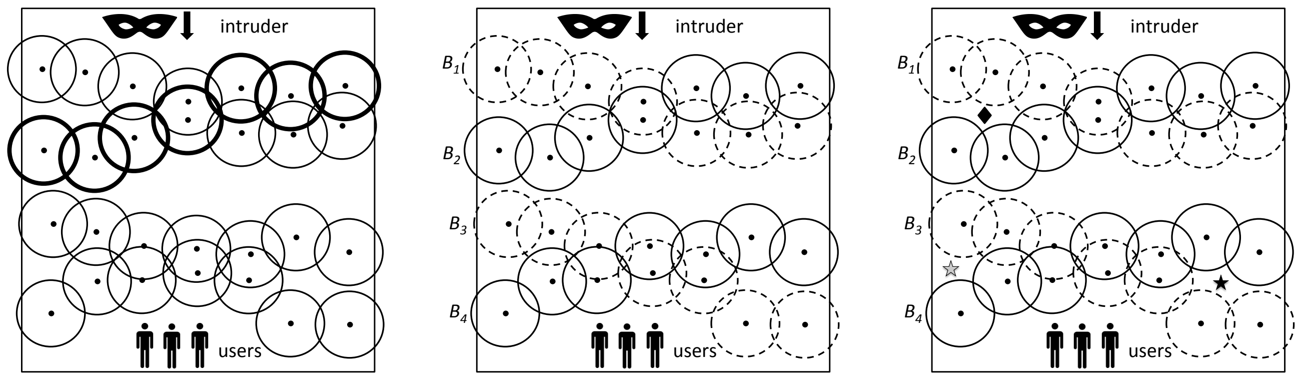
In [19], we presented a heuristic which outperforms those of [17], [18]. This heuristic, as well as those in [17], [18], are centralized. In this paper, we transform our heuristic from [19] into a distributed solution, where the sensor nodes organize themselves into breach-free barriers. In addition to being distributed, our solution is *self-stabilizing* [20]–[23], i.e., starting from any state, a subsequent state is reached and maintained where the sensors are organized into breach-free barriers. A system that is self-stabilizing is resilient against transient faults, because the variables of the system can be corrupted in any way (that is, the system can be moved into an arbitrary configuration by a fault) and the system will naturally recover and progress towards a normal operating state.

The paper is organized as follows. Section II reviews the concept of a barrier breach, and our heuristic for breach-free barriers. In Section III, we discuss the basic mechanisms necessary to obtain a distributed version of the heuristic. Notation for our specification is given in Section IV, followed by the specification itself in Section V. Remarks on smaller components necessary to obtain a complete protocol are given in Section VI. An overview of the correctness proof is given in Section VII, followed by concluding remarks in Section VIII.

II. Barrier Breaches

A. Motivation

We first overview the problem of a barrier breach through the example in Figure 1(b). The figure shows four different sensor barriers, with each barrier displayed with different line types.



(a) Sensor Barrier (b) Multiple Sensor Barriers (c) Barrier Breaches

Figure 1. Sensor Barriers

Let us assume that the lifetime of each sensor is one time unit. Furthermore, assume all sensor nodes are operating simultaneously. In this case, the lifetime of the network is simply one time unit, after which an intruder is able to penetrate the area and reach the users.

An alternative approach is to divide the sensors into multiple barriers. In the example above, we can divide the sensors in four barriers, B_1 through B_4. Each of these barriers divides the area into two horizontal sections. If we use the barriers in a sequential wakeup-sleep cycle (B_1, B_2, B_3, and finally B_4), the users are protected for a total of four time units.

Although advantageous in terms of network lifetime, there is a potential drawback to this approach. Consider Figure 1(c), where specific points in the plane have been highlighted.

(a) The order in which the barriers are scheduled makes a significant difference, in particular, for barriers B_1 and B_2. If B_2 is scheduled first, followed by B_1, then an intruder could move to the point highlighted by a diamond, and after B_2 is turned off, the intruder is free to cross the entire area.
(b) Only one of B_3 and B_4 is of use. To see this, suppose that we activate B_3 first. In this case, the intruder can move to the location of marked by the black star. Then, when B_4 is activated and B_3 deactivated, the intruder can reach the users undetected. The situation is similar if B_4 is activated first, and the intruder moves to the location of the grey star.

B. Definitions

We begin by presenting the definition of a barrier breach, as originally proposed in [17].

Definition 1: **(Barrier-Breach)**. An ordered pair (B_1, B_2) of sensor barriers have a *barrier breach* if there exists a point p in the plane such that:

(a) p is outside the sensing range of B_1 and B_2,
(b) B_1 cannot detect an intruder moving from the top of the area to p, and
(c) B_2 cannot detect an intruder moving from p to the bottom of the area.

■

Before presenting our heuristic from [19], we begin with some definitions also introduced in [19].

Definition 2: **(Ceilings and Floors)** Given that a sensor barrier B divides the area of interest into an *upper region* and a *lower region*,

- The *ceiling* of B consists of all points p along the border of the sensing radius of each sensor in B such that one can travel from p to any point in the upper region without crossing the sensing area of any sensor.
- The *floor* of B consists of all points p along the border of the sensing radius of each sensor in B such that one can travel from p to any point in the lower region without crossing the sensing area of any sensor. ■

As an example, consider the sensor barrier depicted in Figure 2(a). The ceiling and floor of this barrier are depicted in Figure 2(b), where the ceiling is depicted with a solid line and the floor with a dashed line.

Using these definitions, we can obtain a condition that guarantees that a breach is not present [19].

Lemma 1: **(Breach-Freedom)** An ordered pair (B_1, B_2) is breach-free iff the floor of B_2 is below the ceiling of B_1. ■

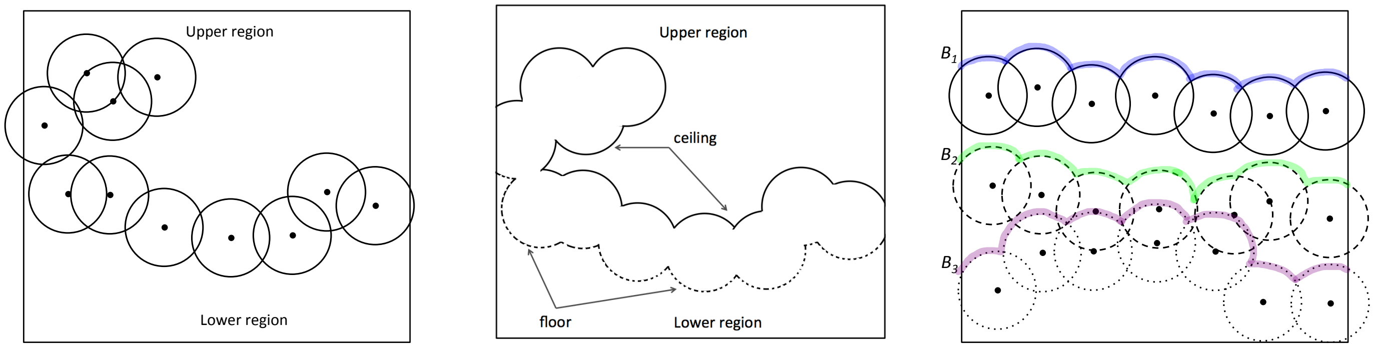
Theorem 1: **(Non-Penetrable)** A schedule (sequence) $(B_1, B_2, \ldots, B_n)$ of sensor barriers is non-penetrable iff, for each i, $1 \leq i < n$, the ordered pair (B_i, B_{i+1}) is breach-free [19]. ■

Consider for example Figure 1(c). The pair (B_1, B_2) does *not* have a barrier breach because the floor of B_2 never crosses over the ceiling of B_1. The pair (B_2, B_1) does have a breach.

Note also that both (B_3, B_4) and (B_4, B_3) have a breach. Thus, they cannot be scheduled one after the other. This, however, does not preclude them from being in a schedule together (although not in the network in Figure 1). For example, assume that we can add more sensor nodes that form a barrier (that is, from the left border to the right border of the area) and the sensors run along the middle of B_3 and B_4, closing the gaps between these barriers. If this new barrier is B', then the schedule (B_3, B', B_4) is a non-penetrable schedule.

C. Ordered Ceilings Heuristic

Our heuristic is based on the following observation, which follows from the above theorem.



(a) Sensor Set (b) Ceiling and Floor (c) Barriers Obtained

Figure 2. Ceiling-First Method

Observation 1: If a set of m sensor barriers does not have a pair of barriers whose ceilings intersect, then a non-penetrable schedule exists of duration m by scheduling the sensor barriers in order from top to bottom. ■

Our heuristic simply finds each barrier iteratively as follows. Consider the set of all sensor nodes as a barrier, and obtain its ceiling. The first barrier consists of all sensor nodes that take part of this ceiling. These nodes are then removed from the network, and a new ceiling is obtained, which yields a new barrier, etc.. Figure 2(c) shows a sample sensor network and the three barriers resulting from the heuristic.

III. Distributed Implementation

We next discuss how to obtain a distributed implementation of our heuristic described above. We begin by making some assumptions about the network.

A. Model

Each sensor node is assumed to be equipped with a global positioning system (GPS) or other means by which it can infer its location. We assume the sensing area of each node forms a circle, or can be approximated by the largest circle within its sensing area. The area of interest is assumed to be rectangular, as shown in Figure 1, and each sensor is able to determine if its sensing area overlaps either the left or right border of the area of interest. Finally, we assume that nodes whose sensing range overlaps are able to communicate wirelessly with each other, i.e., the transmission range is greater than the sensing range.

Self-stabilizing systems are assumed to run continuously, otherwise, they would not have time to recover from a transient fault. In our system, we assume that the batteries of the sensors can be recharged, such as by solar cells or by a station transmitting microwaves, and thus the network can run continuosly. However, being actively sensing depletes the battery of the sensor. Sensors must therefore have a period of rest to recharge.

From the above, we assume that the network operates as follows. If there are n barriers constructed, then each barrier, from top to bottom, is activated sequentially. By the end of the lifetime of network n, we assume that the first barrier has had enough time to recharge to be reactivated, and the schedule continues.

There is of course a period of vulnerability when switching from barrier n to barrier 1, since an intruder that moved closed to barrier n could reach the users once the barrier switch is performed. We assume that the users are aware of this vulnerability and will take additional protection measures during this small interval of time.

Finally, since nodes need to communicate to maintain their relationships, we assume that nodes, whether actively sensing or not, wake up at specified intervals and exchage messages with their neighbors to maintain or correct their state.

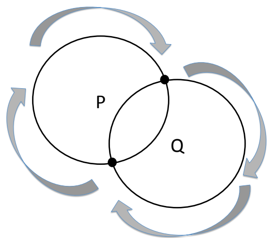
B. Method

Consider Figure 3(a). Any two sensor areas that overlap each other will intersect at only two points. We view these two points as “edges” (P, Q) and (Q, P). These edges are directed according to clockwise order, as indicated in the figure. Hence, the top dark circle corresponds to edge (P, Q) (from P to Q), while the bottom dark circle corresponds to edge (Q, P) (from Q to P).

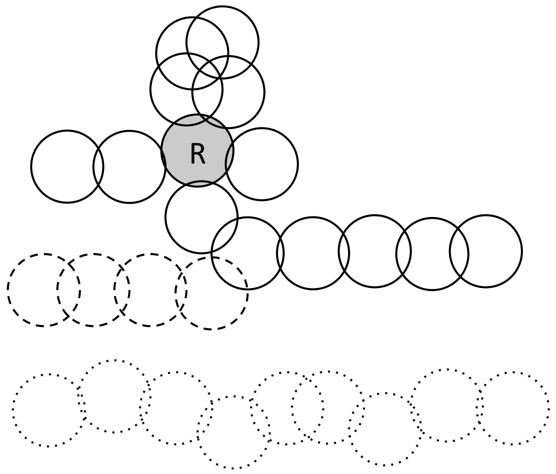
To form a barrier, a node whose sensing range overlaps the left border finds the outgoing edge clockwise that is closest to the point on the left border. This edge points to the next node on the barrier. This is process is then repeated. That is, the second sensor node chooses the edge that is closest clockwise the the incoming edge of the previous node, and so on. The process continues until the right border is found.

As an example, consider again Figure 2(a). The node overlapping the border begins by choosing as the next barrier node its neighbor higher up as opposed to its neighbor below. This is because the edge to the higher up neighbor occurs first clockwise, with respect to the point on the border, than the edge to the neighbor below. The process repeats, with the node higher up choosing the first clockwise outgoing edge (relative to the incoming edge of the previous node). The border obtained is given in Figure 2(b), which corresponds to the ceiling of the nodes.

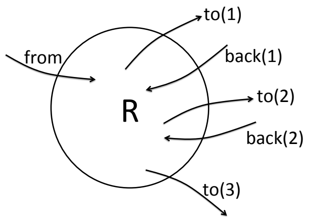
An interesting observation is that the ceiling may come back to the original node. This is the case in Figure 2(a), but not in Figure 2(c). This is illustrated more clearly in Figure 3(b). Consider the barrier drawn with solid lines. When the filled circle, R, is reached, the next barrier node is directly above it. As the barrier continues to be built, the barrier returns back to R. The next node is to the right of R, which immediately returns back to R. The barrier then proceeds along



(a) Sensing Intersection



(b) Detours



(c) Pointers to Neighbours

Figure 3. Neighbor Relationships

the bottom circle. Thus, we can say that there are two "detours" at R before continuing on with the barrier. These detours have to be taken into consideration when designing the distributed algorithm for barrier construction below.

Another observation from Figure 3(b) is that some sensors at the left border are unable to find a path to the right border. This is the case with the barrier attempt with dashed lines. However, it is still possible for a node further below to reach the right border.

C. Variables and Neighbor Relationships

To implement the above scheme, the main variables (pointers) of a sensor node R are shown in Figure 3(c). Variable *from* contains the identity of the previous sensor node in the barrier. Variables *to* and *back* are parallel sequences that contain the identities of the neighbors that follow node R in the barrier. In Figure 3(c), $to(1)$ and $back(1)$ correspond to the pair of nodes of the first detour of node R, and similarly, $to(2)$ and $back(2)$ correspond to the pair of nodes for the second detour of R. Finally, $to(3)$ corresponds to the next node in the barrier that is not part of a detour. Hence, in a stable state, $|to| = |back| + 1$, and the last element of to corresponds to the next node in the barrier.

Assume a node R must choose between two neighbors, P and Q, to become its *from* neighbor. That is, P and Q are both pointing towards R, and R must be able to distinguish which one is "best". If P's barrier originated at a higher point on the border than Q's barrier, then R will choose P. However, if both have the same origin point (especially during a stabilization phase), more information is needed to break the tie. Also, R must be able to determine if P and Q are pointing at it not because they occur before R in the barrier, but because they are returning to R from a detour of several hops.

One approach could be for neighbors to exchange the entire path from the border node to themselves when communicating with each other. This is sufficient but somewhat excessive, especially since detours are likely to be either short or non-existent in a barrier, and communication should be minimized in a wireless system. We choose instead to have each node maintain an abbreviated version of its path as follows.

For a node on the left border of the area, its path is simply the pair $(d, 1)$, where d is the distance from the top of the area to the point on the border where the barrier begins. The second number in the pair is a hop count. Thus, assuming the barrier has no detours, then a node h hops from the left border will have a path equal to (d, h). Also, notice that if there are no detours, then variable $to(1)$ always points to the next node on the barrier.

Assume now that detours do exist. Let $S = R.to(3)$, i.e., S is the beginning of the third detour of R. Then

$$S.path = R.path : (2, 1)$$

where colon denotes concatenation. The first number denotes the number of complete detours in its predecessor, R, and the second number denotes the hop count from the point of the detour. Hence, the number of pairs in a path correspond to the number of nodes encountered that had at least one complete detour. In consequence, if there are no detours after S, then the nodes after S have the same path as S, except that the hop count in the last pair increases with each hop.

Given the paths of two nodes, R and S, we denote by $R \prec S$ if R occurs first in the barriers before S. That is, either R occurs in a barrier above the barrier of S, or they occur in the same barrier and R occurs first in the barrier. This is straightforward to determine from the paths as follows.

- If $R.path$ and $S.path$ are equal except in the hop count of the last pair, then $R \prec S$ if the hop count of R is smaller.
- Let (d, h) and (d', h') be the first pair in $R.path$ and $S.path$ where $d \neq d'$. Then, $R \prec S$ if $d < d'$.

IV. Protocol Notation

We choose to specify our protocol using the notation from [22], [23]. The behavior of each node is specified by a set of inputs, a set of variables, a set of parameters, and a set of actions.

The inputs declared in a process can be read, but not written, by the actions of that process. The variables declared in a process can be read and written by the actions of that process. For simplicity, we assume a shared memory model, i.e., each node is able to read the variables of its neighbors. We discuss how this can be relaxed to a message passing model in the concluding remarks. Parameters are discussed further below.

Every action in a process is of the form:

$$\langle\text{guard}\rangle \rightarrow \langle\text{statement}\rangle.$$

The <guard> is a boolean expression over the inputs, variables, and parameters declared in the process, and also over the variables declared in the neighboring processes of that process. The <statement> is a sequence of assignment statements that change some of the variables of the node.

The parameters declared in a process are used to write a set of actions as one action, with one action for each possible value of the parameters. For example, if we have the following parameter definition,

```
par g : 1 .. 2
```

then the following action

```
$x = g$  →  $x := x + g$
```

is a shorthand notation for the following two actions.

```
  $x = 1$  →  $x := x + 1$
▯
  $x = 2$  →  $x := x + 2$
```

An execution step of a protocol consists in evaluating the guards of all the actions of all processes, choosing an action whose guard evaluates to true, and executing the statement of this action. An execution of a protocol consists of a sequence of execution steps, which either never ends, or ends in a state where the guards of all the actions evaluate to false. We assume all executions of a protocol are weakly fair, that is, an action whose guard is continuously true must be eventually executed.

We say a network *stabilizes* to a predicate P iff, for every execution (regardless of the initial state) there is a suffix in the execution where P is true at every state in the suffix [22], [23].

To distinguish between variables of different nodes, we prefix the variable names with node names. For example, variable $x.v$ corresponds to variable v in node x. If no prefix is given, then the variable corresponds to the node whose code is being presented.

V. Protocol Specification

Below, we present the specification of a stabilizing protocol that organizes sensors into breach-free barriers. The sensor barrier of a node can be obtained by following its pointer variables, i.e., the left node is indicated by variable *from* and its right node is indicated by the last entry in its variable *to*.

The code below does not organize the barriers, i.e., assign to each a natural number to indicate its position on the schedule of barriers. This is a simple addition that will be overviewed in Section VI.

To simplify the presentation of the code, we do not directly have actions for the case when a node's sensor area overlaps the borders, i.e., when a node is a potential endpoint of a barrier. Instead, we assume that there are two virtual nodes S and T, where S is beyond the left border and T is beyond the right border. Any sensor node P overlapping the left border is assumed to have an incoming edge (S, P). Furthermore, the path that S advertises to P is of the form $(d, 0)$, where d is the depth of the point where P's sensor range overlaps the left border. That is, the distance from the top of the region to this point. In this way, no two sensors on the border will have the same path. In the case when sensors are located right next to each other, ties can be broken by node id's.

The inputs and variables of a sensor node u are as follows. We will describe the actions further below.

```
node $u$
inp
   $G$      :  set of node id's          {sensing neighbors}
   $L$      :  natural number            {max. barrier length}
var
   $from$   :  element of $G$;
   $to$     :  sequence of element of $G$;
   $back$   :  sequence of element of $G$;
   $path$   :  sequence of $(N^+,\ 1 \ldots L)$;
par
   $g$      :  element of $G$    {$g$ is any neighbor of $u$}
   $i$      :  $1 \ldots |G|$
begin
           <actions>
end
```

The node has two inputs. Input G is the set of neighboring sensor nodes. We assume a sensor can determine its neighbor set via a simple hello protocol. The second input, L, is the maximum number of hops that is allowed in a barrier. I.e., the sum of the hop counts of all elements of a path should be at most L. This bound is not necessary to break loops, but it may be used to speed up convergence.

The variables of each process are as described earlier. Variable *from* points to the previous node on the barrier, and variable *to* is a sequence pointing to the next nodes on the barrier: one entry for every detour and the final entry points to the true next node. Variable *back* contains the return nodes from the detours, and *path* is the abbreviated path of the node.

Each node has ten actions, which we present in groups. The first two actions are as follows.

```
  $from = nil \lor u \notin from.to \lor$
  $\neg coherent(from, to, back)) \lor HC(path) > L$   →
           $from := nil;$
           $to := \emptyset;\ back := \emptyset;\ path := \emptyset;$
▯
  $from.to(i) = u\ \land$
  $path \neq extend\text{-}one\text{-}hop(from.path, i)$   →
           $from := nil;$
           $to := \emptyset;\ back := \emptyset;\ path := \emptyset;$
```

These actions are sanity actions for variables $from$, to, and $path$. In the first action, if there is no previous node ($from = nil$), then all variables should be set to nil and empty, because all values depend on having a previous node. Also, the values are reset when $coherent(from, to, back)$ is false. This is a function that checks that the values of $from$, to, and $back$, follow the clockwise pattern as shown in Figure 3(c), i.e., starting at variable $from$, we have an alternating clockwise sequence of to and $back$ pointers. The hop count of the path is also checked.

In the second action, we have a sanity check on the path variable. In particular, the path should be derived from the path of the previous node (pointed by $from$), according to the rules of Section III-C. This is obtained from function

$$extend\text{-}one\text{-}hop(path, i)$$

that returns the same path with an increased hop count of 1 when $i = 1$, or returns $path : (i-1, 1)$ when $i > 1$.

The third, fourth, and fifth actions check sanity on variables to and $back$.

```
  $g = to(i) \wedge back(i) \neq nil \wedge u \neq g.from$   $\rightarrow$
        $to := to(1 : i-1);$
        $back := back(1 : i-1);$
▯
  $g = back(i) \wedge u \notin g.to$   $\rightarrow$
        $to := to(1 : i);$
        $back := back(1 : i-1);$
▯
  $back(i) = g \wedge g.path \notin extend\text{-}multiple\text{-}hops(path, i)$ $\rightarrow$
        $to := to(1 : i);$
        $back := back(1 : i-1);$
```

In the third action, if u has a detour whose first node is neighbor g, but g is not selecting u as its left neighbor, then the detour, and any that follow it, are invalid and must be deleted.

In the fourth action, if a detour appears to return via neighbor g, and g is not pointing towards u at all, then also this detour, and any that follow it, are invalid and must be deleted.

Finally, the fifth action ensures that if there is a detour, then the node from where the detour is returning (neighbor g) has a path that is an abbreviated extension of the path of u. The set of all possible extensions of the path of u are denoted by the function $extend\text{-}multiple\text{-}hops(path, i)$.

The sixth action below attempts to find a more suitable left neighbor, as follows.

```
  $from \neq g \wedge u = g.to(i) \wedge$
  $extend\text{-}one\text{-}hop(g.path, i) \preceq path \wedge HC(g.path) < L$ $\rightarrow$
        $from := g;$
        $to := \emptyset;$
        $back := \emptyset;$
        $path := extend\text{-}one\text{-}hop(g.path, i);$
```

In this action, if the a neighbor g is pointing at node u, and the path that u would obtain via g is better than its current path, and the hop-count is not violated by the new path, then g is chosen as the new left neighbor. Since all other variables depend on the left neighbor, the to and $back$ variables are reset.

The seventh and eighth actions below find the next element in the to and $back$ sequences, as follows.

```
  $from \neq nil \wedge |to| = |back| \wedge$
  $extend\text{-}one\text{-}hop(path, |to|) \preceq g.path \wedge HC(path) < L \wedge$
  $(back(|to|), u) \rightsquigarrow (u, g) \rightsquigarrow (from, u)$   $\rightarrow$
        $to := to : g;$
▯
  $|to| > |back| \wedge u \in g.out \wedge$
  $g.path \in extend\text{-}multiple\text{-}hops(path, |to|) \wedge$
  $(from, u) \rightsquigarrow (u, to(|to|)) \rightsquigarrow (g, u)$   $\rightarrow$
        $back := back : g;$
```

In the seventh action, if all detours are complete ($|to| = |back|$) and the path u offers to g is better than whatever path g currently has, and g is in the correct clockwise order (denoted by $\rightsquigarrow$), i.e., it is between the end of the last detour and before the $from$ neighbor, then u points to g as a possible next neighbor in the barrier.

In the eighth action, a neighbor g is checked to see if it completes the last detour, and if so it is added to the $back$ sequence.

The remaining ninth and tenth actions below attempt to improve upon neighbors that have been chosen in the to and $back$ arrays.

```
  $|to| \geq i \wedge extend\text{-}one\text{-}hop(path, i) \preceq g.path \wedge$
  $HC(path) < L \wedge (from, u) \rightsquigarrow (u, g) \rightsquigarrow (u, to(i))$ $\rightarrow$
        $to := to(0 : i-1) : g;$
        $back := back(0 : i-1);$
▯
  $|back| \geq i \wedge u \in g.out \wedge$
  $extend\text{-}one\text{-}hop(path, i) \preceq g.path \preceq back(i).path \wedge$
  $(u, to(i)) \rightsquigarrow (g, u) \rightsquigarrow (from, u)$   $\rightarrow$
        $to := to(0 : i);$
        $back := back(0 : i-1) : g$
```

In the ninth action, if a neighbor g is to be chosen to replace $to(i)$, then the path of g must improve with the change, and g has to be in the correct clockwise order. In particular, it must occur before the current $to(i)$ node.

In the tenth action, if a node g is chosen to replace $back(i)$ (that is, the node completing the i^{th} detour) then the path of g must be better than that of $back(i)$, and it must also occur in the correct clockwise order.

Given that detour j, where $j > i$, depends on detour i, all detours greater than i are deleted.

VI. Completing the Protocol

The protocol presented in Section V organizes the sensors into disjoint breach-free barriers, but it does not organize them into a schedule. However, each sensor node must know the number of its barrier (counting from top to bottom) to be able to turn its sensing feature at the right time.

To accomplish the above, the nodes at the right barrier can organize themselves in a simple sequence from top-to-bottom. Any of these nodes that is pointed by a to entry of a neighbor becomes the end point of the barrier. A simple diffusing computation from top to bottom can assign numbers to all the nodes that are the end point of a barrier. These numbers can then be propagated to the other sensor nodes in the direction of the left border by following the $from$ variable at each node.

VII. Correctness

Due to space limitations, the proof of correctness of the protocol is deffered to [24]. The proof follows the following overall steps.

First, due to the sanity actions, predicate $coherent(from, to, back)$ will hold and continue to hold, in addition to $|back| \leq |to| \leq |back + 1|$. That is, the variables satisfy what is depicted in Figure 3(c).

Next, although the upper bound L on the hop count is enforced, the bound L is not necessary to break loops. A loop exists if we follow the $from$ variables and reach the same node a second time. Loops are broken quickly, because the hop counts must be consistent (differ by exactly one) between nodes, or otherwise all variables are reset to nil and empty values. The total order $\preceq$ on paths prevent new loops to be formed.

The following step is to show that all nodes only contain abbreviated paths that have as their first entry a non-fictitous entry point along the left border. The path with a fictitous first entry and with the smallest hop count will not match the path of its $from$ neighbor, and thus will reset its values. Thus, in L steps all path with fictitious first entries disappear.

Next, due to the total order of $\preceq$, the nodes along the top barrier will overcome any other path value in the system, thus completing the top barrier in L steps. The remaining barriers will be constructed similarly in top-down order.

VIII. Concluding Remarks

Our execution model is based on shared memory. However, a message passing implementation is straightforward using the techniques described in [22] due to the low level atomicity of the actions, that is, each action refers to variables of only a single neighbor at a time.

One possible weakness is the case in which the sensor nodes are so sparse that the nodes bordering the right wall form a disjoint network, and thus cannot coordinate with each other the number that should be given to each barrier. This could be mitigated if barrier numbers originate also from the nodes on the left border. Another mitigating factor is that even though two barriers may not be able to communicate with each other at the borders, they might be able to do so in the middle of the area of interest if their respective sensors are close to each other. This may provide aid in coordinating the numbers. We leave these issues for future work.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, no. 12, Aug 2008, pp. 2292–2330.

[2] C. Huang and Y. Tseng, "The coverage problem in a wireless sensor network," in ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA), 2003, pp. 115–121.

[3] H. Zhang and J. Hou, "On deriving the upper bound of α-lifetime for large sensor networks," in Proc. of The 5th ACM Int'l Symposium on Mobile Ad-hoc Networking and Computing (MobiHoc), 2004, pp. 121–132.

[4] Cardei, M., Thai, M.T., Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in INFOCOM 2005, vol. 3, March 2005, pp. 1976–1984.

[5] M. Thai, Y. Li, and F. Wang, "O(log n)-localized algorithms on the coverage problem in heterogeneous sensor networks," in IEEE Int'l Performance, Computing, and Communications Conference, 2007. IPCCC 2007., April 2007, pp. 85–92.

[6] S. Gao, X. Wang, and Y. Li, "p-percent coverage schedule in wireless sensor networks," in Proc. of 17th Int'l Conference on Computer Communications and Networks, 2008. ICCCN '08., Aug 2008, pp. 1–6.

[7] C. Vu, G. Chen, Y. Zhao, and Y. Li, "A universal framework for partial coverage in wireless sensor networks," in Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th Int'l, Dec 2009, pp. 1–8.

[8] Y. Li, C. Vu, C. Ai, G. Chen, and Y. Zhao, "Transforming complete coverage algorithms to partial coverage algorithms for wireless sensor networks," Parallel and Dist. Systems, IEEE Trans. on, vol. 22, no. 4, April 2011, pp. 695–703.

[9] S. Kumar, T. Lai, and A. Arora, "Barrier coverage with wireless sensors," in Proc. of the 11th Annual Int'l Conference on Mobile Computing and Networking (MobiCom), 2005, pp. 284–298.

[10] A. Saipulla, C. Westphal, B. Liu, and J. Wang, "Barrier coverage of line-based deployed wireless sensor networks," in INFOCOM 2009, April 2009, pp. 127–135.

[11] S. Kumar, T. Lai, M. Posner, and P. Sinha, "Maximizing the lifetime of a barrier of wireless sensors," Mobile Computing, IEEE Transactions on, vol. 9, no. 8, Aug 2010, pp. 1161–1172.

[12] H. Yang, D. Li, Q. Zhu, W. Chen, and Y. Hong, "Minimum energy cost k-barrier coverage in wireless sensor networks," in Proc. of the 5th Int'l Conf. on Wireless Algorithms, Systems, and Applications (WASA), 2010, pp. 80–89.

[13] H. Luo, H. Du, D. Kim, Q. Ye, R. Zhu, and J. Zhang, "Imperfection better than perfection: Beyond optimal lifetime barrier coverage in wireless sensor networks," in Proc. of The IEEE 10th Int'l Conference on Mobile Ad-hoc and Sensor Networks (MSN 2014), Dec 2014, pp. 24–29.

[14] D. Li, B. Xu, Y. Zhu, D. Kim, and W. Wu, "Minimum (k,w)-angle barrier coverage in wireless camera sensor networks," Int'l Journal of Sensor Networks (IJSNET), vol. 19, no. 2, 2015.

[15] L. Guo, D. Kim, D. Li, W. Chen, and A. Tokuta, "Constructing belt-barrier providing quality of monitoring with minimum camera sensors," in Computer Communication and Networks (ICCCN), 2014 23rd Int'l Conference on, Aug 2014, pp. 1–8.

[16] B. Xu, D. Kim, D. Li, J. Lee, H. Jiang, and A. Tokuta, "Fortifying barrier-coverage of wireless sensor network with mobile sensor nodes," in Proc. of the 9th Int'l Conference on Wireless Algorithms, Systems, and Applications (WASA 2014), Jun 2014, pp. 368–377.

[17] D. Kim, J. Kim, D. L. abd S. S. Kwon, and A. Tokuta, "On sleep-wakeup scheduling of non-penetrable barrier-coverage of wireless sensors," in Proc. of the IEEE Global Communications Conference (GLOBECOM 2012), Dec 2012, pp. 321–327.

[18] H. B. Kim, "Optimizing algorithms in wireless sensor networks," Ph.D. dissertation, The U. of Texas at Dallas, Advisor: J. Cobb, May 2013.

[19] J. A. Cobb, "Improving the lifetime of non-penetrable barrier coverage in sensor networks," in International Workshop on Assurance in Distributed Systems and Networks, 2015, pp. 1–10.

[20] M. Schneider, "Self-stabilization," ACM Computing Surveys, vol. 25, no. 1, March 1993, pp. 45–67.

[21] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," Commun. ACM, vol. 17, no. 11, 1974, pp. 643–644.

[22] S. Dolev., Self-Stabilization. Cambridge, MA: MIT Press, 2000.

[23] M. G. Gouda, "The triumph and tribulation of system stabilization," in WDAG '95: Proceedings of the 9th International Workshop on Distributed Algorithms. London, UK: Springer-Verlag, 1995, pp. 1–18.

[24] J. A. Cobb and C. T. Huang, "Stabilizing breach-free sensor barriers," in Technical Report, Dept. Computer Science, The University of Texas at Dallas, May 2015.