Stabilizing 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. The sensors can be divided into multiple subsets, known as barriers. The area remains protected, or covered, by a sensor barrier if the barrier divides 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. Breaches are not dependent on the structure of an individual sensor barrier. Instead, they are dependent on the relative shape of two consecutive sensor barriers. In this paper, the best-performing centralized heuristic for breach-free barriers is transformed into a distributed protocol. Furthermore, 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. A detailed proof of the stabilization of the protocol is also given. Finally, it is shown how the barriers can organize themselves into a sleep-wakeup schedule without centralized support.**

Keywords–Stabilization; Sensor networks; Sensor barriers.

I. INTRODUCTION

Earlier work [1] outlined a distributed protocol for obtaining a set of breach-free sensor barriers in a fault-tolerant manner. In particular, the protocol in [1] is stabilizing. In this paper, the protocol is presented in greater depth, and a detailed proof that the protocol is stabilizing is also given. In addition, a final component of the heuristic that was left in [1] for future work is developed. Before presenting the protocol, the concepts of sensor barriers, breach-free sensor barriers, and stabilizing protocols are overviewed below.

A wireless sensor network consists 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 [2].

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 [3] [4] [5] [6]. Partial coverage, on the other hand, has regions within the area of interest that are not covered by the sensors [7] [8] [9].

One form of partial coverage that received significant attention due to its application to intrusion detection is barrier coverage [10] [11] [12] [13] [14] [15] [16] [17]. 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. Fig. 1(a) highlights a subset of sensors that provide barrier coverage to a rectangular area such as a corridor in a building. The users are located at one end of the corridor (called the *bottom* of the area) and possible intruders may arrive via the opposite end (called the *top* of the area).

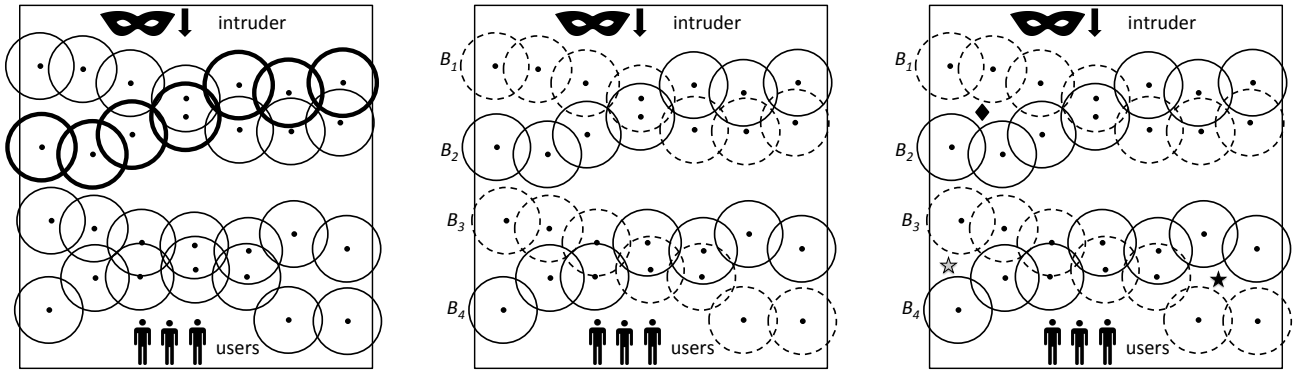
In the specific case of intrusion detection, providing full coverage is not an efficient use of the sensor resources, and leads to a reduced network lifetime. Instead, multiple sensor barriers can be constructed, as illustrated in Fig. 1(b). Only one barrier needs to be active at any moment in time; 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 area of interest, finding the largest number of sensor barriers is solvable in polynomial-time [12].

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 [18] [19]. 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 [18] [19].

An additional heuristic that outperforms those of [18] [19] was presented in [20]. This heuristic, as well as those in [18] [19], are centralized.

In [1], the centralized heuristic from [20] is transformed into a distributed protocol, where the sensor nodes organize themselves into breach-free barriers. In addition to being distributed, the solution is *self-stabilizing* [21] [22] [23] [24], i.e., starting from any arbitrary 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.

In this paper, the distributed solution of [1] is presented in greater detail and in a manner that is easier to follow. In



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

Figure 1. Sensor barriers.

addition, a detailed proof is given that the solution is indeed stabilizing. Finally, a feature of the protocol that in [1] was left for future work is explored and developed. Namely, the barriers organize themselves into a sleep-wakeup schedule without centralized support.

The paper is organized as follows. Section II reviews the concept of a barrier breach, and the centralized heuristic for breach-free barriers. In Section III, the basic mechanisms necessary to obtain a distributed version of the heuristic are discussed. Notation for the specification is given in Section IV, followed by the specification itself in Section V. A quick overview of the proof is given in Section VI. The detailed proof is given in the appendix. The specification of the component that allows the barriers to organize themselves into a sleep-wakeup schedule is given in Section VII, followed by the conclusion and future work in Section VIII.

II. Related Work and Background on Barrier Breaches

A. Motivation

The problem of barrier breaches can be seen through the example in Fig. 1(b). The figure shows four different sensor barriers, with each barrier displayed with different line types.

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, the sensors are divided into four barriers, B_1 through B_4. Each of these barriers divides the area into two horizontal sections. If the barriers are used 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. Obviously, while transitioning from barrier B_i to barrier B_{i+1}, there has to be a small amount of time during which both barriers are active. Otherwise, an intruder can reach the users at the moment barrier B_i is deactivated.

Although advantageous in terms of network lifetime, there is a potential drawback to this approach. Consider Fig. 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 B_3 is activated 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

The original definition of a barrier breach was given in [18], as follows.

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.

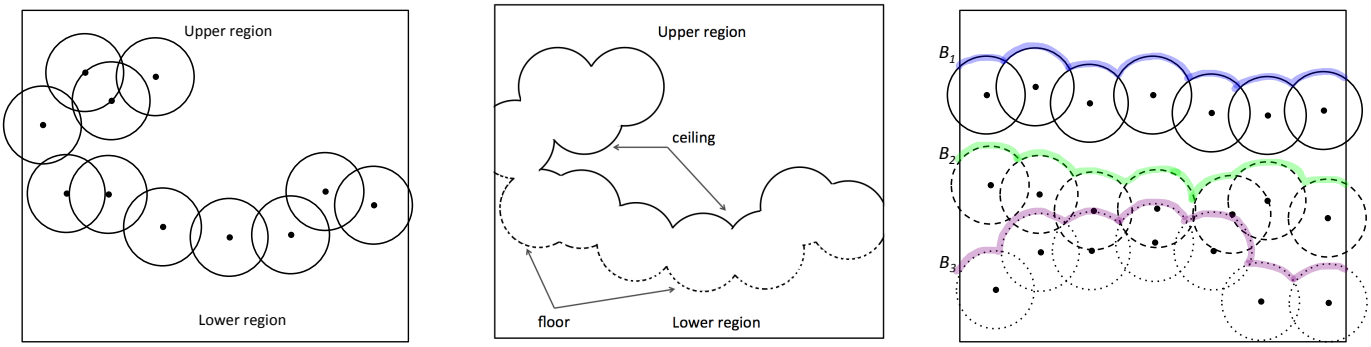
■

Before presenting our heuristic from [20], some background definitions given in [20] are reviewed.

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 Fig. 2(a). The ceiling and floor of this barrier are depicted in Fig. 2(b), where the ceiling is depicted with a solid line and the floor with a dashed line.



(a) Sensor set. (b) Ceiling and floor. (c) Barriers obtained.

Figure 2. Ordered-ceilings method.

Using these definitions, a condition can be obtained that guarantees that a breach is not present [20].

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. ■

This can then be used to ensure that an intruder does not reach the users, as follows. A schedule, i.e., a sequence, of barriers $(B_1, , B_2, \ldots, B_n)$ is said to be *non-penetrable* if there is no sequence of moves that an intruder can make to reach the users without being detected by any of the barriers during the lifetime of the schedule.

Theorem 1: **(Non-Penetrable)** A schedule $(B_1, B_2, \ldots, B_n)$ 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 [20]. ■

Consider for example Fig. 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 Fig. 1). For example, assume that more sensor nodes are added to form a new barrier (that is, from the left border of the area to its right border) 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. Disjoint Paths Heuristics

As mentioned above, several heuristics have been developed to obtain breach-free barriers. The heuristics presented in [18] [19] [25] are based on using a variant of maximum network flow to find the largest number of node-disjoint (i.e., sensor-disjoint) paths (i.e., barriers) that begin on the left side of the area and terminate on the right side of the area.

In [20], a heuristic known as the *ordered ceilings heuristic* is proposed, and it is shown to outperform the heuristics in [18] [19] [25]. The only exception is when the number of sensors per unit area is unreasonably high, in which case the heuristic of [25] outperforms the ordered ceilings heuristic.

D. Ordered Ceilings Heuristic

The ordered ceilings heuristic, which is the focus of this paper, is a centralized method that is based on the following observation that follows from the above theorem.

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. ■

The 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.. Fig. 2(c) shows a sample sensor network and the three barriers resulting from the heuristic.

III. Distributed Implementation

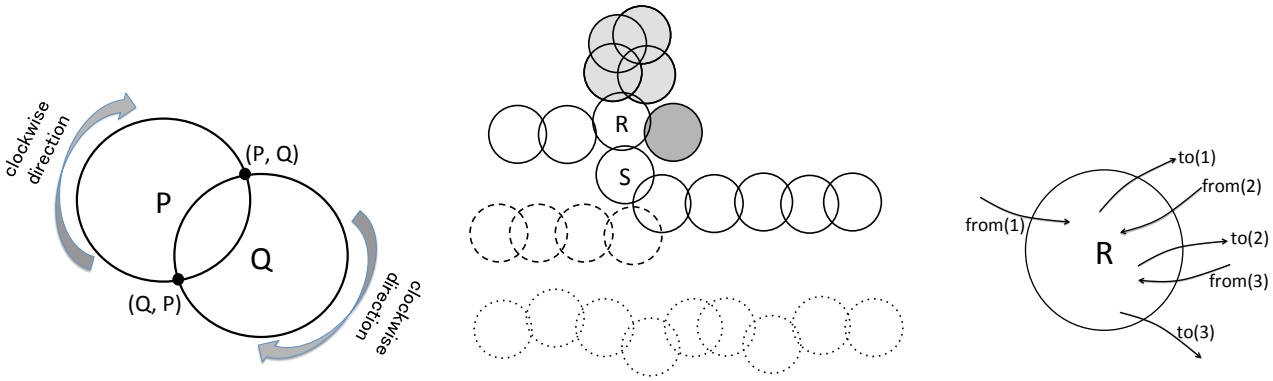
In this section, the method used to transform the centralized heuristic into a distributed protocol is presented. Assumptions about the network model are presented first, followed by the steps to perform this transformation.

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. The sensing area of each node is assumed to form 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 Fig. 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, it is assumed that nodes whose sensing range overlap are able to communicate wirelessly with each other, i.e., the transmission range is greater than twice the sensing range.

The batteries used by the sensors are assumed to be rechargeable, by means such as solar cells or by a station transmitting microwaves, and thus the network can run continuously. However, being actively sensing depletes the battery of the sensor. Sensors must therefore have a period of rest to recharge. However, self-stabilizing systems are assumed to run continuously, otherwise, they would not have time to recover from a transient fault.

Thus, by the above reasons, it is assumed 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 barrier n, the first barrier has had enough time to recharge to the level to be reactivated, and the schedule continues.



(a) Sensing intersection. (b) Detours (c) Pointers to neighbors.

Figure 3. Neighbor relationships.

There is of course a period of vulnerability when switching from barrier n to barrier 1, since an intruder that moved closer to barrier n could reach the users once the barrier switch is performed. It is expected that the users are aware of the time at which this vulnerability occurs, and they will take additional protection measures during this time.

Finally, sensor nodes, whether actively sensing or not, must wake up at specified intervals and exchange messages with their neighbors to maintain or correct their state.

B. Method

Consider Fig. 3(a). Any two sensor areas that overlap each other will intersect at only two points. These points are viewed as "edges" (P, Q) and (Q, P). These edges are directed according to clockwise order, as indicated in the figure. Hence, the top intersection point corresponds to edge (P, Q) (from P to Q), while the bottom intersection point 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 its 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 Fig. 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 Fig. 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 Fig. 2(a), but not in Fig. 2(c). This is illustrated more clearly in Fig. 3(b). Consider the barrier drawn with solid lines. When the barrier construction reaches sensor R, the next sensor in the barrier is directly above it. As the barrier continues to be built, the barrier returns back to R. These nodes constitute a *detour*, and are drawn filled with gray. The next node is to the right of R, which immediately returns back to R. This is another detour, but it consists of a single node. The barrier then proceeds along sensor S. Thus, 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 Fig. 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, such as in the case of the barrier drawn with dotted lines. In particular, assume that in a network there exists a set of m barriers with no overlapping sensor regions. For example, Fig. 3(b) has two barriers that do not overlap: the one drawn with solid lines and the one drawn with dotted lines. Then, the ordered ceilings heuristic is guaranteed to find at least m breach-free barriers.

C. Variables and Neighbor Relationships

To implement the above scheme, the main variables (pointers) of a sensor node R are shown in Fig. 3(c). Both variables *from* and *to* are parallel sequences of neighbors. If the node has no detours, then $from(1)$ is the previous neighbor in its barrier, and $to(1)$ is the next node in the barrier. However, assume that node R has two detours, which is the case in Fig. 3(c). In this case, $to(1)$ is the next node after R in the first detour, and $from(2)$ is the neighbor from which the first detour returns. Similarly, $to(2)$ and $from(3)$ are the next node and the returning node for the second detour. Finally, $to(3)$ is the neighbor that follows R in the barrier, and this neighbor is not involved in a detour at R. Hence, in a stable state where all barriers are fixed, $|from| = |to|$, 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(1)* 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

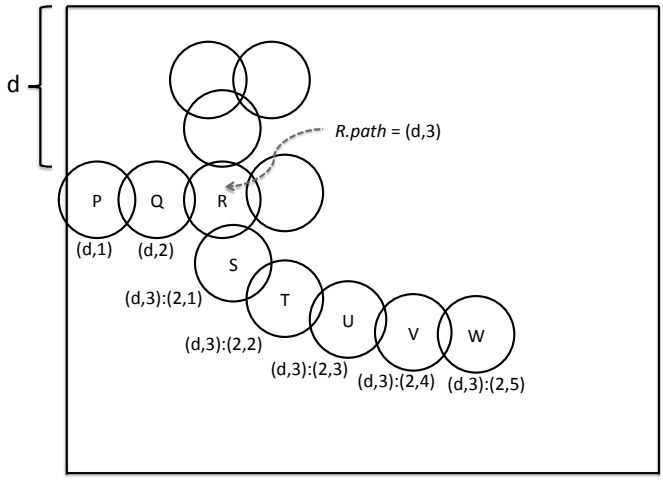


Figure 4. Path example.

at it because they occur before R in the barrier or 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. For efficiency, each node instead maintains 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 $R.to(3) = S$, 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.

Consider as an example Fig. 4, where a barrier is being constructed from left to right (only part of the barrier is drawn). The sensor node on the left border is P, and it intersects the left border at a distance d from the top. Hence, its path is $(d, 1)$. Because P has no detours, node Q has a path equal to $(d, 2)$, i.e., two hops from the left border. Similarly, because node Q has no detours, node R has a path equal to $(d, 3)$. However, R does have two detours, and thus the path of S is $(d, 3) : (2, 1)$. The pair $(2, 1)$ indicates that two detours were skipped at the previous node, i.e., at R, and that S is one hop away from the node where the detours were skipped. Because neither S nor any remaining node have detours, the paths of the remaining nodes simply consist of increasing the hop count of the last term in the path. E.g., the path of W is $(d, 3) : (2, 5)$, because W is five hops away from the node where the last detour occurred, i.e., from R.

The hop count of a path is denoted by $HC(path)$. It is simply the sum of the hop counts of each term in the path. For example, node $HC(W.path) = 3+5 = 8$ and $HC(T.path) = 3 + 2 = 5$. Note that HC corresponds to the number of hops along the barrier if the nodes involved in a detour are not counted.

Given the paths of two nodes, R and S, $R \prec S$ denotes that 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

The notation used to specify the protocol originates from [23] [24], and is typical for specifying stabilizing systems. 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, a shared memory model is used, i.e., each node is able to read the variables of its neighbors. This can be relaxed to a message-passing model, which is discussed in the conclusion and future work section. Parameters are discussed further below.

Every action in a process is of the form:

$$\text{<guard>} \rightarrow \text{<statement>}.$$

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 the following parameter definition is given,

par g : 1 .. 2

then the following action

$x = g \quad \rightarrow \quad x := x + g$

is a shorthand notation for the following two actions.

$x = 1 \quad \rightarrow \quad x := x + 1$

$x = 2 \quad \rightarrow \quad x := x + 2$

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. All executions of a protocol are assumed to be weakly fair, that is, an action whose guard is continuously true must be eventually executed.

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 [23] [24].

To distinguish between variables of different nodes, the variable name is prefixed with the node name. 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

The specification of a stabilizing protocol that organizes sensors into breach-free barriers is given below. The sensor barrier of a node can be obtained by following its pointer variables, i.e., the left node is indicated by variable *from(1)* 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 presented in Section VII.

To simplify the presentation of the code, the actions for sensor nodes whose sensing region overlaps the borders, i.e., when nodes are a potential endpoint of a barrier, are not presented. Instead, it is assumed 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) whose intersection point with P is the point where P intersects the left border. Furthermore, the path that S advertises to P is of the form $(d, 0)$, where d is the depth of the point of (S, P). 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 complete specification of an arbitrary sensor node u is given in Fig. 5. The specification is broken down into smaller segments below, and the intuition behind each of them is presented.

The inputs and variables of a sensor node u are as follows. The actions are described further below.

node u
inp G : **set of node id's** {sensing neighbors}
L : **natural number** {max. barrier length}
var $from$: **sequence of element of** G;
to : **sequence of element of** G;
$path$: **sequence of** $(N^+, 1 \dots L)$;
par g : **element of** G {any neighbor of u}
i : $1 \dots |G|$

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

{action 1: new or improved $from(1)$}
$from(1) \neq g \wedge u = g.to(i) \wedge HC(g.path) < L \wedge$
$extend\text{-}one\text{-}hop(g.path, i) \prec path \wedge \quad \rightarrow$
$from := \{g\}; \quad to := \emptyset;$
$path := extend\text{-}one\text{-}hop(g.path, i);$

[] {action 2: new or improved $to(i)$}
$|from| \geq i \wedge HC(path) < L \wedge g \neq to(i) \wedge$
$clockwise(from(1, i), to(1, i-1) : g) \wedge$
$|to| \geq i \Rightarrow between(from(i), g, to(i)) \wedge$
$(extend\text{-}one\text{-}hop(path, i) \prec g.path \vee$
$path \in extend\text{-}multiple\text{-}hop(g.path)) \quad \rightarrow$
$to := to(1, i-1);$
$from := from(1, i);$
if $u \notin g.from$ **then**
$to := to : g;$

[] {action 3: new or improved $from(i+1)$}
$|to| \geq i \wedge u \in g.to \wedge g \neq from(i+1) \wedge$
$from\text{-}consistent(g, u, i) \wedge$
$clockwise(from(1, i) : g, to(1, i)) \wedge$
$|from| > i \Rightarrow between(to(i), g, from(i+1)) \quad \rightarrow$
$from := from(1, i) : g;$
$to := to(1, i);$

[] {action 4: sanity of to, $from$, and $path$}
$\neg(|to| \leq |from| \leq |to| + 1) \vee HC(path) > L \vee$
$(|from| = 0 \wedge path \neq \emptyset) \vee \neg clockwise(from, to)$
$\rightarrow$
$from := \emptyset; \; to := \emptyset; \; path = \emptyset;$

[] {action 5: sanity of $from(i)$}
$|from| \geq i \; \wedge \neg(u \in from(i).to \wedge$
$from\text{-}consistent(from(i), u, i)) \quad \rightarrow$
$from := from(1, i-1);$
$to := to(1, i-1);$

[] {action 6: sanity of $to(i)$ and neighbor's path}
$|to| \geq i \wedge g = to(i) \wedge \neg(path \prec g.path \wedge$
$(|from| > i \Rightarrow to(i).from(1) = u)) \quad \rightarrow$
$from := from(1, i);$
$to := to(1, i-1);$
end

Figure 5. Specification of an arbitrary sensor node u.

The node has two inputs. Input G is the set of neighboring sensor nodes. It is assumed that 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 $from$ and to of each process are as described earlier. Variable *path* is the abbreviated path of the node. Also, $from(i, j)$, where $i < j$, denotes the subsequence of $from$ starting at $from(i)$ and ending at $from(j)$. The subsequence $to(i, j)$ is defined in the same way.

Each node has six actions. Due to the semantics, the order in which they are written is irrelevant for their execution. Thus, actions are presented below in the order that is the easiest to describe.

The first action obtains a value for $from(1)$, or replaces it by a better value.

$$\begin{array}{l} from(1) \neq g \wedge u = g.to(i) \wedge HC(g.path) < L \wedge \\ \textit{extend-one-hop}(g.path, i) \prec path \wedge \quad \rightarrow \\ \qquad from := \{g\}; \quad to := \emptyset; \\ \qquad path := \textit{extend-one-hop}(g.path, i); \end{array}$$

The above action checks that if a neighbor g is pointing at u (i.e., $g.to(i) = u$ for some i), and the path of g being offered to u is better than u's current path, then u chooses g as its predecessor. Note that by changing the value of $from(1)$ all other values of $from$ and to may be invalid, since in effect the node is changing from one barrier to another. Hence, to is set to empty, and the path of u is obtained form that of g. This is obtained from function

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

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

If $to(i)$ has a value, then the following action attempts to improve it, i.e., find a neighbor that is closer clockwise than $to(i)$. If $to(i)$ does not have a value, then the action attempts to find a neighbor to point to with $to(i)$.

$$\begin{array}{l} |from| \geq i \wedge HC(path) < L \wedge g \neq to(i) \wedge \\ \textit{clockwise}(from(1, i), to(1, i-1) : g) \wedge \\ |to| \geq i \Rightarrow \textit{between}(from(i), g, to(i)) \wedge \\ (\textit{extend-one-hop}(path, i) \prec g.path \vee \\ \quad path \in \textit{extend-multiple-hop}(g.path)) \quad \rightarrow \\ \qquad to := to(1, i-1); \\ \qquad from := from(1, i); \\ \qquad \textbf{if } u \notin g.from \textbf{ then} \\ \qquad\qquad to := to : g; \end{array}$$

Although the guard of the above action seems complex, it is just a series of simple tests, one per line.

The first one ensures that $from(i)$ is defined, since otherwise $to(i)$ cannot exist, and it ensures that the hop count of the path of u can be extended (i.e., it is less than L).

Also, $from$ and to should remain in clockwise order after replacing $to(i)$ by g.

Furthermore, if $to(i)$ is already defined (i.e., if $|to| \geq i$), then the new value g has to be closer in clockwise order than the current value of $to(i)$, i.e., g has to be in between $from(i)$ and $to(i)$.

Finally, u points to g under two conditions: either u's path will improve the current path of g (and thus g will choose u as its predecessor in the barrier) or u is part of a detour that started at g and u is the last node in this detour. This is expressed using the functions $\textit{extend-one-hop}$, defined earlier, and $\textit{extend-multiple-hops}(path)$, which is defined below.

Intuitively, $\textit{extend-multiple-hops}(path)$ is the set of all possible path values that can be obtained by extending the given $path$ by any number of hops. More formally, let

$$path = (x_1, y_1) : (x_2, y_2) : \ldots : (x_{n-1}, y_{n-1}) : (x_n, y_n)$$

Also, let $path' \in \textit{extend-multiple-hops}(path)$. Thus, $path'$ must have one of two forms. The first case is when

$$path' = (x_1, y_1) : (x_2, y_2) : \ldots : (x_{n-1}, y_{n-1}) : (x_n, z)$$

where $z > y_n$. This indicates that $path'$ extends $path$ by $z - y_n$ hops and there are no detours along these hops. The second case is when

$$\begin{array}{rcl} path' & = & (x_1, y_1) : (x_2, y_2) : \ldots : (x_{n-1}, y_{n-1}) : (x_n, z) : \\ & & (a_1, b_1) : (a_2, b_2) : \ldots : (a_m, b_m) \end{array}$$

where $z \geq y_n$ and $m \geq 1$. This indicates that $path'$ encounters m nodes with detours after u.

Function $\textit{extend-multiple-hops}(path, i)$ denotes the more specific case where the first hop is extended via $to(i)$. Above, it corresponds to $z = y_n \wedge m \geq 1 \wedge a_1 = i$.

Note that if the value of $to(i)$ changes, then all subsequent values of to and $from$ are no longer valid, and they are thus removed by the command of the action. Also, if g is chosen to become $to(i)$, then g cannot already be pointing at u with $g.from$. This is necessary for the safety properties presented in the detailed proof.

The following action obtains a new value of $from(i+1)$, or attempts to improve if it already exists. Note that this does not apply to $from(1)$. Thus, $from(i+1)$ is the neighbor that completes the return of detour i.

$$\begin{array}{l} |to| \geq i \wedge u \in g.to \wedge g \neq from(i+1) \wedge \\ \textit{from-consistent}(g, u, i) \wedge \\ \textit{clockwise}(from(1, i) : g, to(1, i)) \wedge \\ |from| > i \Rightarrow \textit{between}(to(i), g, from(i+1)) \quad \rightarrow \\ \qquad from := from(1, i) : g; \\ \qquad to := to(1, i); \end{array}$$

The above action checks that if g is the neighbor to become $from(i + 1)$, then the values of u and g are consistent. This is done with function $\textit{from-consistent}(g, u, i)$ explained further below. It also checks that the correct clockwise order is maintained. Finally, if $from(i + 1)$ is already defined, i.e., if $|from| > i$, then g is between $to(i)$ and the current $from(i+1)$, i.e., it occurs earlier in the clockwise order than the current $from(i + 1)$.

Similar to above, all values of to and $from$ after $from(i+1)$ are no longer valid, and they are thus removed by the command of the action.

Function $\textit{from-consistent}(g, u, i)$, where g is u's neighbor, and $g = from(i)$, is defined as follows. If $i = 1$, then g is the node previous to u in the barrier, and thus it must be that

$$u.path = \textit{extend-one-hop}(g.path, 1).$$

On the other hand, if $i > 1$, then the path of g is actually an extension of that of u, and thus

$$g.path = \textit{extend-multiple-hops}(u.path, i).$$

Thus,

$$\begin{aligned} &from\text{-}consistent(g,u,i) = \\ &\quad (i = 1 \wedge (u.path = extend\text{-}one\text{-}hop(g.path,1))) \\ &\qquad\qquad \vee \\ &\quad (i > 1 \wedge (g.path = extend\text{-}multiple\text{-}hops(u.path,i))). \end{aligned}$$

The last three actions are *sanity* actions. That is, they check that the local state of the node is correct and also consistent with respect to that of its neighbors. Otherwise, the state of the node is reset to an appropriate value.

The first of the three sanity actions is as follows.

$$\begin{aligned} &\neg(|to| \leq |from| \leq |to|+1) \vee HC(path) > L \vee \\ &(|from| = 0 \wedge path \neq \emptyset) \vee \neg clockwise(from, to) \\ &\rightarrow \qquad from := \emptyset;\ to := \emptyset;\ path = \emptyset; \end{aligned}$$

The above action ensures that the lengths of the $from$ and to variables are consistent, and also that they correspond to points that are clockwise around the sensing circle of the node. Also, it ensures that the path has a length of at most L, and there cannot be a path if there is no $from(1)$ node. If any of these is not true, the local variables are reset to an empty value.

The next action ensures that if $from(i)$ has a value, then the local information is consistent with that of neighbor $from(i)$.

$$\begin{aligned} &|from| \geq i \ \wedge \neg(u \in from(i).to \wedge \\ &from\text{-}consistent(from(i), u, i)) \quad \rightarrow \\ &\qquad from := from(1, i-1); \\ &\qquad to := to(1, i-1); \end{aligned}$$

The above action first checks that $from(i)$ is defined, and if so, it checks if node $from(i)$ is pointing towards u with, and if so, that the path at node u is consistent with that of its neighbor $from(i)$.

In the last action, the value of $to(i)$ and the path of the neighbor it points to are coordinated. The neighbor must have a path that is worse than that of u. In addition, if a detour has completed, then the first node of the detour must point back at u.

$$\begin{aligned} &|to| \geq i \wedge g = to(i) \wedge \neg(path \prec g.path \wedge \\ &(|from| > i \Rightarrow to(i).from(1) = u)) \quad \rightarrow \\ &\qquad from := from(1, i); \\ &\qquad to := to(1, i-1); \end{aligned}$$

VI. Correctness Overview

For terseness, the detailed proof of correctness of the protocol is deferred to the appendix.

The fairness in the execution model allows the actions of nodes to not be executed for an arbitrary (but finite) amount of time. In practice, all nodes operate at about the same speed, and thus, the proofs are based in the commonly used notion of an execution round.

An *execution round* starting at a state s_0 in an execution sequence $s_0, s_1, \ldots,$ is the minimum prefix of this execution sequence such that every action in every node either is disabled at each state in the round, or is enabled at some state in the round and is either executed or disabled at a later state in the round.

The proof follows the following overall steps. First, due to the sanity actions, from any arbitrary initial state, within $O(1)$ rounds the following will hold and continue to hold at every node.

$$\begin{aligned} &(|to|+1 \geq |from| \geq |to| \geq 0) \ \wedge \\ &clockwise(from, to) \wedge HC(path) \leq L \end{aligned}$$

I.e., variables satisfy what is depicted in Fig. 3(c).

Because the initial state is arbitrary, the path stored at each node may not be consistent with that of its neighbors. The next step is to show that it will. To this end, an ordered pair of neighboring nodes (g, u) is said to be *i-joined,* $i \geq 1$, if

$$u \in g.to \wedge g = u.from(i).$$

It must be shown that within $O(1)$ rounds, for all i and for all i-joined pairs (g, u), $from\text{-}consistent(g, u, i)$ will hold and continue to hold.

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 by following the $from(1)$ variables there is a node that can be reached twice. 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.

Within $O(1)$ rounds, it can be shown that there is no sequence of nodes $(u_0, u_1, \ldots, u_n)$ such that $u_0 = u_n$ and $i\text{-}joined(u_j, u_{j+1}, 1)$, for each j, $0 \leq j < n$. That is, following the $from(1)$ values from one node to another does not lead to a loop.

The following step is to show that all nodes only contain abbreviated paths that have as their first entry a non-fictitious entry point along the left border. The path with a fictitious first entry and with the smallest hop count will not match the path of its $from$ neighbor, and thus will reset its values. Thus, within $O(L)$ rounds all path values 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 $O(L)$ rounds. The remaining barriers will be constructed similarly in top-down order. If the total number of barriers is B, then within $O(BL)$ rounds all the barriers will be constructed, and the values of the variables of each node will cease to change.

VII. 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. The steps required to do so are overviewed in this section. The simple proofs of correctness are left to the reader.

First, each node needs to know if the construction of its barrier has completed. To do so, a boolean variable, $built$, is added to indicate if this is the case. The following two actions need to be added to each node u.

$$right\text{-}border \quad \rightarrow \quad built := (|from| \geq 1)$$

$$\neg right\text{-}border \quad \rightarrow \quad built := (|to| \geq 1 \wedge to(|to|).built)$$

Above, $right\text{-}border$ is true if node u overlaps the right border of the area. The fact that the barrier has been built propagates from the node on the right border back to the node on the left border. If u is on the right border, then its barrier is complete provided it is part of a barrier, i.e., if $from(1)$ is defined. If it is not on the right border, then the last element of array to points to the next node along the barrier (other elements of to point to detour nodes). Thus, if this last element thinks the barrier is complete then node u also will consider the barrier complete.

Next, if its barrier is built, each node must determine the order of its barrier in the schedule. The topmost border is first in the schedule, followed by the next border down. For this purpose, four additional variables are added to each node u:

$order$: an positive integer, indicating the order of u's barrier in the schedule.
$depth$: the depth of the barrier of u (i.e., distance from the top of the area.
$prev$: the depth of the barrier that is previous (right above) the barrier of u.
src : the source of the information for the previous barrier. This is the neighbor of u from whom u learned about the previous barrier. This could be either a neighbor on the same barrier as u that intersects the previous barrier, or u has a neighbor that directly intersects the previous barrier. If there is no previous barrier, $src = u$.

The depth of the barrier of u is determined by the location of the left-most node of the barrier, i.e., the node whose sensor range overlaps the left border. Recall that this information is present in the first item of the path variable. Hence, $depth$ is simply defined as the first value in $path$.

There are two tasks. The first task is to improve the choice for the previous barrier. That is, there could be a barrier that is in between node u's barrier and what u believes should be the previous barrier. The second task is to ensure that the values of the four variables are consistent.

For the first task, the following action is added to each node u.

$$\begin{aligned}(depth > g.depth > prev) \wedge built \wedge g.built \quad \rightarrow \\ order := g.order + 1; \\ prev := g.depth; \\ src := g\end{aligned}$$

This action chooses g as the source of the information if its depth is between the depth of the former previous barrier and the depth of the barrier of u.

Next, the values of the four variables must be consistent. If they are inconsistent, then they are reset using the following commands.

$$\begin{aligned}src := u; \\ prev := 0; \\ order := 1;\end{aligned}$$

With these commands, u assumes that it is the first barrier. It will remain this way until it improves. Let us refer to those three commands as $reset\text{-}src$;

There are three different cases when $reset\text{-}src$ should be executed, depending on the value of the source.

- The first is when the source is u itself, but its values are inconsistent. Let us refer to this case as $bad\text{-}u$, and is defined as follows.
$$src = u \wedge \neg(built \wedge prev = 0 \wedge order = 1)$$
- The second is when the source is either the left or right node on the same barrier. Let us refer to this case as $bad\text{-}from\text{-}to$, and is defined as follows.
$src \in \{from(1), to(|to|)\} \wedge$
$\neg(src.src \neq u \wedge src.built \wedge built \wedge$
$src.order = order \wedge$
$(src.prev = prev < depth))$
- The last one is when the source is not on the same barrier as u, denoted by $bad\text{-}other$, and is defined as follows.
$src.depth \neq depth \wedge$
$\neg(src.built \wedge built \wedge src.src \neq u \wedge$
$(src.depth = prev < depth) \wedge$
$src.order + 1 = order)$

Thus, to each node u, the following action is added.

$$bad\text{-}u \vee bad\text{-}from\text{-}to \vee bad\text{-}other \quad \rightarrow \quad reset\text{-}src$$

VIII. Conclusion and Future Work

In this paper, a distributed and stabilizing version of the best-performing centralized heuristic for breach-free barriers is presented. Also, an additional feature is developed that allows the barriers to organize themselves into a sleep-wakeup schedule without centralized support.

The execution model used is based on shared memory. However, a message passing implementation is straightforward using the techniques described in [23] due to the low level atomicity of the actions, that is, each action refers to variables of only a single neighbor at a time.

The stabilization time of $O(B \cdot L)$ rounds is an upper bound on the worst-case behavior of the system when all variables have an arbitrary initial value. A more detailed analysis may reveal an even lower upper bound, such as $O(L)$. This investigation is left for future work. Also, on average, it is expected that the system will recover much faster than this from a few random faults. This could be analyzed via simulations, which are also deferred to future work.

Appendix

Some basic stabilization properties are presented first. For a property P to hold within $O(1)$ rounds, it must be shown that if P holds before each action, then it will continue to hold after the action is executed. In addition, there must be an action that once executed it makes P true. Hence, once P becomes true, it continues to be true.

Theorem 2: Let $S1$ be the following safety predicate.

$$\begin{aligned}(|to| + 1 \geq |from| \geq |to| \geq 0) \wedge \\ clockwise(from, to) \wedge \\ HC(path) \leq L\end{aligned}$$

Predicate $S1$ will hold and continue to hold after $O(1)$ rounds.

Proof:

Step 1:

Assuming that $S1$ holds before each action, it is shown next that it will continue to hold after the action is executed.

First action: If the action executes, $from = \{g\}$ and $to = \emptyset$, which trivially satisfies the last two conjuncts of $S1$. From the action's guard, $HC(g.path) < L$, and thus, after extending it by one hop, $HC(path) \leq L$ holds. Thus, $S1$ continues to hold.

Second action: If the action executes, then after the action $|to| + 1 \geq |from| \geq |to|$. The path is not affected, so $HC(path) \leq L$ continues to hold. Also, the guard of the action ensures that the new values of $from$ and to will be clockwise. Hence, $S1$ continues to hold.

Third action: If the action executes, then $|from| = i + 1 \wedge |to| = i$, which satisfies $S1$. Also, the guard of the action ensures that the values are clockwise. The path variable is not affected. Hence, $S1$ continues to hold.

Fourth action: The empty values assigned by the action trivially satisfy $S1$.

Fifth action: The fifth action only shortens $from$ and to, with $|from| = |to|$, and hence, all three conjuncts of $S1$ continue to hold after the action.

Sixth action: Similar to above, the action only shortens $from$ and to, and hence, $S1$ continues to hold.

Step 2:

There must be an action that forces $S1$ to become true if it does not hold before the action. This action is the fourth action. If any of the conjuncts of $S1$ does not hold, then the action is able to execute. Due to fairness, it will execute in one round, and thus force $S1$ to become true. ■

Theorem 3: Within $O(1)$ rounds,

$$(\forall i, v, w : i\text{-joined}(v, w) \Rightarrow from\text{-}consistent(v, w, i))$$

holds and continues to hold.

Proof: Let $S2$ be the above predicate. The same two steps as above will be used but now for $S2$. Note that if i-joined(v, w, i) is false for some specific values of i, v, and w, then it is not required to be proven that $from\text{-}persistent(v, w, i)$ holds. The only case where the implication can be falsified is in the case where the left-hand side is true and the right-hand-side is false. Thus, let us focus on when an action turns the left-hand side true (in which case it must also set the right hand side true), or when an action makes the right-hand-side false (in which case the left-hand side must also be set to false).

Step 1:

Assuming that $S2$ holds before each action, it will be shown that it will continue to show after the action is executed.

First action: This action eliminates all the join relations in which node u is included. However, it does establish a new one, 1-joined(g, u). In this case, the new value of $path$, i.e., extending by one hop the path of g, is what $S2$ demands. Thus, $S2$ holds after this action.

Second action: This action also removes join relations since it shortens to and $from$. It has the potential to add a new one due to giving a new value to $to(i)$. However, this is only done if g is not pointing back at u with $g.from$. Hence, no new join relation can be created, and thus $S2$ is preserved.

Third action: Similar to the second action, this action also removes join relations since it shortens to and $from$. It has the potential to add a new one due to giving a new value to $from(i+1)$, thus creating an i-joined(g, u) link. This new link satisfies $from\text{-}consistent(g, u, i)$ due to the guard. Hence, $S2$ holds.

Fourth, fifth, and sixth actions: Executing these actions can only eliminate join relations, and hence $S2$ continues to hold.

Step 2:

There must be an action that forces $S2$ to become true if it does not hold before the action. Consider any triple (v, w, i) such that $S2$ does not hold. This implies that the left-hand-side, i-joined(v, w) is true, while the right hand side, $from\text{-}persistent(v, w, i)$ is false. If within $O(1)$ rounds the left-hand-side becomes false, then there is no proof obligation. Assume otherwise. Then, the guard of action 5 is true with $w = u$ and $v = u.from(i)$. In this case, when the action executes, the pair (v, w) is no longer i-joined, and $S2$ holds and continues to hold for the triple (v, w, i). ■

Observation 2: Within $O(1)$ rounds, there is no sequence of nodes $(u_0, u_1, \ldots, u_n)$ such that $u_0 = u_n$ and $joined(u_j, u_{j+1}, 1)$, for each j, $0 \leq j < n$. That is, following the $from(1)$ values from one node to another does not lead to a loop. Furthermore, this continues to hold.

The above observation follows from the value of $path$ decreasing, with respect to $\prec$, at every hop when the $from(1)$ values are followed (as indicated by Theorem 3), and from the antisymmetry of $\prec$.

Observation 3: Within $O(1)$ rounds, any execution will reach a state where $S1 \wedge S2$ holds, and this continues to hold for all remaining states of the execution.

Recall that all variables can have an arbitrary value in the initial state. Throughout the rest of this section, let us assume a state as indicated in Observation 3 has already been reached. Next, it must be shown that all nodes will have a path variable whose initial position on the left border corresponds to that of a real node on the border, i.e., no fictitious initial nodes will exist in the path variables.

Lemma 2: After $O(L)$ rounds, for any node u, if the first value in its path variable is x, then there exists a node on the left border with depth x.

Proof: Consider any value y such that there is no sensor node on the left border with this depth, and there is at least one node whose path contains (y, h) for some h. It must be shown that all paths with a value of y must disappear.

First note that no new fictitious y can be introduced into the system, since any new path of a node is derived from that of its neighbors.

Let us denote a path value as a y-path if it begins with a depth of y. The first pair in a y-path is of the form (y, h), where h is the hop count. Let h_{min} be the smallest value of h in a y-path. It is argued next that h_{min} must increase.

Let u have (y, h_{min}) in its path. Because of Theorem 3, $from\text{-}consistent(u.from(1), u, 1)$ must be false and continue to be false. This is because this requires the path of

$u.from(1)$ to be (y, h') where $h' < h_{min}$. Thus, either u changes $from(1)$ using the first action, or action 5 will set $from$ to empty (note that by definition $from(1, 0)$ is the empty sequence). In the former, the neighbor of u must have a path that does not start with y, or has y and a hop count greater than or equal to h_{min}. Thus, u loses the y value or increases its hop count. In the latter, $|from| = 0$, in which case eventually u chooses a new $from(1)$ and the same argument applies for a higher hop count. It is also possible that action 4 sets the path to empty, in which case node u also loses its y value.

Thus, (y, h_{min}) will disappear. Since the hop count in (y, h) has a maximum value of L, then within $O(L)$ rounds all paths values starting with y will disappear. ■

The main result is presented next, i.e., that the barriers are actually constructed. The first step is showing that the top-most barrier gets constructed, and all the values of its nodes remain stable. The building of the remaining barriers follow simply by induction on the barriers.

Theorem 4: Within $O(L)$ rounds, the $from$ and to variables of the nodes in the top most barrier correctly follow the sequence of nodes in the barrier until the rightmost node is reached.

Proof: The proof is by induction over the nodes in the barrier. Let the barrier nodes be $u_0, u_1, \ldots, u_n$.

Base case: Consider node u_0, which intersects the left border. Let its depth be x. If any node has a path starting with $(x, 0)$, when action 5 executes in the node, the new path will not have $(x, 0)$.

The only node that can advertise $(x, 0)$ is the virtual node S. Thus, consider any node whose path begins with $(x, 1)$. If the node is not on the left border, then the path cannot be consistent with its $from(1)$ node (and if $from(1)$ is not defined then action 4 will set $path$ to nil). Thus, action 5 will set the path to nil. Thus, no node other than the left node can have $(x, 1)$ in its path.

Finally, consider the left-most node. It receives an advertised path of $(x, 0)$ from the virtual node S.

There are four cases to consider.

1) $u_0.from(1) \neq S$ and $u_0.path = (x, 1)$.
   As long as this is the case, the path is not consistent since no node other than S can advertise $(x, 0)$. Thus, action 5 remains enabled until the above case changes.
2) $u_0.from(1) = S$ and $u_0.path$ is worse than $(x, 1)$.
   This is not possible since it is not $from$-$consistent$, and violates Theorem 3.
3) $u_0.from(1) \neq S$ and $u_0.path$ is worse than $(x, 1)$.
   Action 1 remains enabled unless the case changes. Since no node can advertise $(x, 0)$ other than S, the only way this case may change is by executing the first action, which establishes the fourth case.
4) $u_0.from(1) = S$ and $u_0.path = (x, 1)$.
   In this case, no node can offer a better path than S, u_0 is consistent with S, and from Theorems 2 and 3 the sanity actions will not fire. Hence, this case remains true forever.

Thus, these three cases must eventually end up in case 4, which remains true forever, as desired. Note that this will be done in $O(1)$ rounds.

Inductive Step: Assume the Theorem holds for $u_0, u_1, \ldots, u_h$, show that it will hold for $u_0, u_1, \ldots, u_{h+1}$.

Let $to(i)$ be the pointer at u_h that is meant to point at u_{h+1} (all pointers less than i at u_h are fixed by the induction hypothesis). It must be shown that eventually $u_h.to(i) = u_{h+1}$, $u_{h+1}.from(i) = u_h$ for the appropriate i, and $u_{h+1}.path = extend\text{-}one\text{-}hop(u_h, i)$.

Note that because u_{h+1} follows u_h in the barrier, there cannot be any node, whether in the barrier or not, that is clockwise in between the last $from$ value at u_h and node u_{h+1}.

There are two cases to consider.

Case 1:

Node u_{h+1} *does not* appear earlier in the barrier. Hence, it should eventually be the case that $u_{h+1}.from(1)$ should point to u_h.

There are two subcases.

Sub-case (a): $u_h.to(i) = u_{h+1}$ already.

In this case $to(i)$ cannot change value. This is because, from Theorems 2 and 3, the sanity actions will not fire. Also, no other neighbor can be clockwise in between u_h and u_{h+1}, as mentioned above. Hence, action 2 cannot change $to(i)$.

Sub-case (b): $u_h.to(i) \neq u_{h+1}$.

From the induction hypothesis, the path at u_{h+1} is worse than that of u_h. Furthermore, as argued before there are no nodes clockwise in between these two nodes. Hence, the second action fires, which clears all the pointers equal or greater than i.

If $u_h \notin u_{h+1}.from$, then $u_h.to(i)$ is set to u_{h+1}, as desired. Note that this continues to hold because there cannot be a node clockwise in between these two nodes.

If $u_h \in u_{h+1}.from$, this prevents $to(i)$ to be set. However, from this point forward, no value $to(j)$, $j > i$, can point to u_{h+1}. This is because if $to(i)$, $to(i+1)$, etc., are set by action 2, then the nodes chosen appear clockwise after u_{h+1}. Hence, u_{h+1} cannot be set to any $to(j)$, $j > i$. Thus, eventually action 5 in u_{h+1} executes and removes the $from$ value pointing back at u_h. Then, nothing can stop action 2 at u_h to set $u_h.to(i) = u_{h+1}$, which is subcase (a).

Hence, since u_h continuously points at u_{h+1}, the first action of u_{h+1} will set $u_{h+1}.from(1) = u_h$, as desired.

Case 2:

Node u_{h+1} *does* appear earlier in the barrier. Hence, it must be shown that $u_h = u_{h+1}.from(i)$, where i is the next index at u_{h+1} that has not been used for earlier parts of the barrier. Note that $from(j)$ and $to(j)$, where $j < i$, are fixed at u_{h+1} due to the induction hypothesis. Similarly, to and $from$ up to $i - 1$ are fixed in u_h from the induction hypothesis.

The first step is to show that eventually $u_h.to(i) = u_{h+1}$ holds and continues to hold. The proof up to this point is the same as in Case 1, except that in Case 1 $extend\text{-}one\text{-}hop$ was used in action 2, while in Case 2 $extend\text{-}multiple\text{-}hop$ is used. Thus, it is assumed below that $u_{h+1}.from(i) = u_h$ holds and continues to hold.

Note that since both u_h and u_{h+1} are both already in the barrier (i.e. $u_{h+1} = u_j$ for some $j < h + 1$). Hence, from the induction hypothesis, their paths are fixed, and the path of u_h is an extension of the path of u_{h+1}.

Also, as argued above there is no neighbor that is clockwise in between u_h and u_{h+1}.

Hence, if $u_{h+1}.from(i) = u_h$, then this will continue to hold (sanity actions no longer fire, and from above, action 3 cannot choose a node better than u_h).

If on the other hand, $u_{h+1}.from(i) \neq u_h$, then also from the above, the guard of action 3 is enabled and continues to be enabled until it fires, setting $u_{h+1}.from(i) = u_h$.

End of inductive step.

Thus, within $O(L)$ rounds, the barrier up to the node at the right border will be constructed. Although discussed earlier, it is assumed that nodes at the right border are aware that they are located on the border. Thus, these nodes have only the first and fifth actions, they restrict $from$ to only one value, and they have no to variable. Thus, their path will be maintained consistent with its $from(1)$ neighbor, and they always choose the best possible neighbor for $from(1)$.

Thus, by induction, the theorem holds. ■

After the top barrier is constructed above, there might be some $from$ and to values that are dangling, i.e., that point to nodes not on the barrier. These have to disappear to ensure that the next barrier is constructed without interference from the first barrier.

Theorem 5: Let x be the depth of the top-most barrier. Within $O(L)$ rounds, there are no nodes that are not in the top barrier that contain a depth of x in their path. Also, the $from$ and to variables of nodes in the top barrier point exclusively to the appropriate nodes in the barrier.

Proof: Consider any node u that is on the barrier. Assume that some of its $from$ or to variables, other than those used for the barrier, have values. Let $to(i)$ be the last value in to pointing at a barrier node. Thus, $from(i+1)$ points to a non-barrier node, let us say, w.

If the path of w is consistent with u, then the value of $w.path$ should be a possible extension of u via detour i, i.e., $w.path \in extend\text{-}multiple\text{-}hop(u, i)$. If it is not, then the fifth action of u will remove all $from$ and to values after $to(i)$. If it is, then u cannot remove $from(i+1)$.

In this case, consider following the $from(1)$ variables (i.e., backwards), starting at w, and continuing as long as the next node's path is the extension by one hop of the previous node. Eventually, a node is reached whose path is not in $extend\text{-}multiple\text{-}hop(u, i)$, and thus is not a one-hop extension of the previous node. Thus, action 5 in this node will execute at the node and remove its pointers. This will also activate action 5 in the previous node, continuing in a cascade of nodes until w is reached. Then, in node u, action 5 will reset all pointers after $to(i)$.

Thus, at all nodes, eventually only those pointers used for the barrier have a value. ■

Since the nodes not in the top barrier cannot receive a path value corresponding to nodes in the top barrier, at this moment the next barrier can be built independently of the previous one, and thus, from induction, all barriers will be built.

Corollary 1: Within $O(B \cdot L)$ rounds, where B is the number of barriers in the ceilings heuristic, the $from$, to, and $path$ variables of all nodes are aligned to these barriers, and continue to be aligned unless a fault occurs in the system.

REFERENCES

[1] J. A. Cobb and C. T. Huang, "Fault-tolerant breach-free sensor barriers," in Proc. of the International Conference on Systems and Networks Communications (ICSNC), Nov. 2015, pp. 63–69.

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

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

[4] 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), Jun. 2004, pp. 121–132.

[5] M. Cardei, M. T. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in Proc. of the IEEE INFOCOM Conference, vol. 3, Mar. 2005, pp. 1976–1984.

[6] M. T. Thai, Y. Li, and F. Wang, "O(log n)-localized algorithms on the coverage problem in heterogeneous sensor networks," in Proc. of the IEEE Int'l Performance, Computing, and Communications Conference, (IPCCC), Apr. 2007, pp. 85–92.

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

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

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

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

[11] A. Saipulla, C. Westphal, B. Liu, and J. Wang, "Barrier coverage of line-based deployed wireless sensor networks," in Proc. of the IEEE INFOCOM Conference, Apr. 2009, pp. 127–135.

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

[13] 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), Aug. 2010, pp. 80–89.

[14] 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), Dec. 2014, pp. 24–29.

[15] B. Xu, Y. Zhu, D. Li, 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, pp. 179–188.

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

[17] 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 Int'l Conference on Wireless Algorithms, Systems, and Applications (WASA), Jun. 2014, pp. 368–377.

[18] D. Kim, J. Kim, D. Li, S. S. Kwon, and A. O. 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.

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

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

[21] M. Schneider, “Self-stabilization,” ACM Computing Surveys, vol. 25, no. 1, Mar. 1993, pp. 45–67.

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

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

[24] M. G. Gouda, “The triumph and tribulation of system stabilization,” in Proc. of the 9th International Workshop on Distributed Algorithms (WDAG). London, UK: Springer-Verlag, 1995, pp. 1–18.

[25] J. A. Cobb, “In defense of stint for dense breach-free sensor barriers,” in Proc. of the International Conference on Systems and Networks Communications (ICSNC), Aug. 2016, pp. 12–19.