Coverage Preserving Redundancy Elimination in Sensor Networks

Bogdan Cărbunar, Ananth Grama, Jan Vitek
Computer Sciences Department
Purdue University
West Lafayette, IN 47907, USA
Email: {carbunar,ayg,jv}@cs.purdue.edu

Octavian Cărbunar
IFIN-NIPNE
Turnul Magurele, Romania
Email: carbunar@ifin.nipne.ro

***Abstract*— In this paper we study the problem of detecting and eliminating redundancy in a sensor network with a view to improving energy efficiency, while preserving the network's coverage. We also examine the impact of redundancy elimination on the related problem of coverage-boundary detection. We reduce both problems to the computation of Voronoi diagrams, prove and achieve lower bounds on the solution of these problems, and present efficient distributed algorithms for computing and maintaining solutions in cases of sensor failures or insertion of new sensors. We prove the correctness and termination properties of our distributed algorithms, and analytically characterize the time complexity and the traffic generated by our algorithms. Our simulations show that the traffic generated per sensor insertion or removal (failure) experiences a dramatic decrease with increase in sensor density, (up to 300% when the number of sensors deployed in the same $1000 \times 1000m^2$ area increases from 150 to 800), and with increase in radio transmission range (up to 200% when the sensor's transmission range increases from 70m to 200m).**

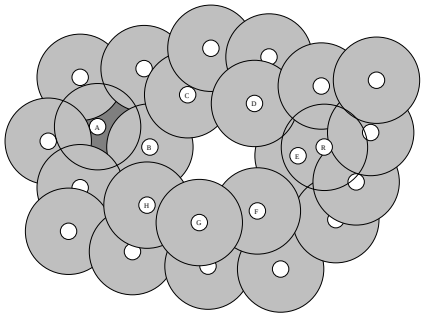


Fig. 1. Example of sensor network coverage – disks represent the coverage of the sensors situated at their center. Lightly shaded disks represent the coverage area of sensors situated on the boundary of the network. The dark areas belong to sensors that are not on the coverage boundary (e.g., sensor A). Sensor R is an example of a redundant sensor, since its coverage area is completely subsumed by other sensors.

I. Introduction

The need for distributed sensing and monitoring of remote or inaccessible areas, along with the integration of sensed information provide overarching motivations for sensor networks. The emphasis on low-cost, dense sensing comes with significant constraints on battery power, communication bandwidth, and compute power. For this reason, considerable work has focused on efficient techniques for extending the network's lifetime. This paper focuses on the problem of detecting and eliminating redundant sensors without affecting network coverage. We refer to this as the *coverage-preserving, energy-efficient redundancy elimination* problem. Fig. 1 illustrates an example of a redundant sensor, R, in a small network. The difficulty of this problem lies in the correct and efficient detection of redundant sensors, and the selection of the maximum number of redundant sensors that can be safely turned off simultaneously.

A related problem is one of detecting the boundary of the network's coverage area. As illustrated in Fig. 1, we define the *coverage-boundary* of a network to include not only the sensors situated on the outer periphery of the network, but also the ones that define "holes" in the coverage (sensors B, C, $\cdots$, H). Coverage boundary is important for identifying holes in coverage and for optimizing sensor placement. It is easy to see that a sensor is redundant if its removal does not impact the coverage boundary.

While these problems are intuitively stated, several constraints contribute to the complexity of these problems. First, sensors must solve these problems in a distributed, efficient, and scalable manner. Second, solutions to these problems must be adaptive in nature, *i.e.*, when old sensors fail or new ones are deployed, the new solution must be rapidly computed from the previous solution. Finally, since the RF interfaces have a limited transmission range, protocols must account for overheads of multi-hop routing. Since communication is generally much more energy consuming than local computation, the protocols themselves need to be energy efficient. Prior work in the area of energy-efficient coverage [1], [2], [3] provides solutions that are either less efficient, do not maintain the initial network coverage, or detect only a subset of the redundant sensors (see Section VII).

This paper makes the following specific contributions: in Section III, we derive necessary and sufficient conditions for a sensor to be redundant, and present an efficient distributed algorithm for the coverage-preserving, energy-efficient redundancy elimination problem. In Section IV, we present necessary and sufficient conditions for a sensor to be on the coverage boundary and show a lower bound of $\Omega(n \log n)$ for any (serial) algorithm for the problem. We propose a distributed algorithm for computing the coverage-boundary, whose serial counterpart has $\Omega(n \log n)$ complexity. Both algo-

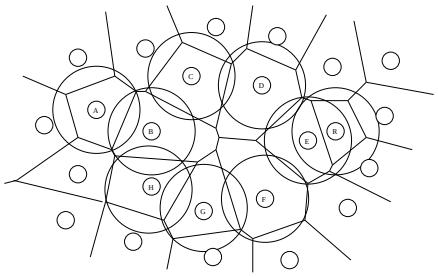


Fig. 2. Voronoi diagram of the sensors in Fig. 1 – circles represent the coverage disks of the labeled sensors. Note that only in the case of sensors A and R, the coverage area completely covers the corresponding Voronoi cell.

rithms are based on the distributed and adaptive construction of Voronoi diagrams. In Section V, we present efficient and scalable distributed algorithms for recomputing local Voronoi information in the presence of sensor failures and deployment of new sensors. We show that our algorithms are efficient and prove their correctness and stability. In Section VI, we experimentally characterize the performance of our algorithms. Section VII discusses related work, and conclusions are drawn in Section VIII.

II. Brief Overview of Voronoi Tessellations

Given a set $\mathtt{S}$ of $\mathtt{n}$ sites $\mathtt{s}_1, \mathtt{s}_2, \cdots, \mathtt{s}_\mathtt{n}$ in the plane, the Voronoi diagram is defined as the subdivision of the plane into $\mathtt{n}$ cells, one for each site, with the property that any point in the cell corresponding to a site is closer to that site than to any other site. Formally, the cell corresponding to site $\mathtt{s}_\mathtt{i}$ is defined as

$$\mathtt{cell}_{\mathtt{vd}}(\mathtt{s}_\mathtt{i}) = \bigcap_{\mathtt{j}=1,\mathtt{j}\neq\mathtt{i}}^{\mathtt{n}} \{\mathtt{x} | \mathtt{dist}(\mathtt{s}_\mathtt{i}, \mathtt{x}) \leq \mathtt{dist}(\mathtt{s}_\mathtt{j}, \mathtt{x})\}$$

We use the notation $\mathtt{dist}(\mathtt{p}, \mathtt{q})$ to denote the Euclidean distance between two points p and q. Two Voronoi cells share a *Voronoi edge*, and three Voronoi cells intersect at a *Voronoi vertex*. We call a site a *generator* or *neighbor* of another site if the Voronoi cells of the two sites share an edge. We will use the two names interchangeably. Fig. 2 illustrates the Voronoi diagram of the network in Fig. 1.

The Delaunay triangulation of a set $\mathtt{S}$ of sites is defined as the unique triangulation of $\mathtt{S}$ such that no point in $\mathtt{S}$ is inside the circumcircle of any triangle of the triangulation. The Delaunay triangulation is the dual of the Voronoi diagram of $\mathtt{S}$, in the sense that two sites are part of the same Delaunay triangle iff they are Voronoi neighbors. We formally define the Delaunay distance:

Definition 2.1: The Delaunay distance between two sites is the minimum number of hops on the Delaunay triangulation between the two sites.

For example, in Fig. 2, sensors A and R are at Delaunay distance 4.

III. Energy-Efficient Coverage

In this section we formalize the coverage-preserving energy-efficient redundancy elimination problem, and provide a solution based on Voronoi tessellations. Throughout this paper, we assume that all the sensors in the network have the same sensing range. We also make the assumption that each sensor knows its two dimensional location. This is a reasonable assumption since, in the absence of this information, the coverage-boundary and the redundancy information cannot be uniquely or correctly determined (i.e., from topological information alone). Extensions of these schemes to non-identical sensing ranges are possible using *multiplicative weighted Voronoi diagrams*. We demonstrate this in the context of the coverage-boundary problem in [4].

We now formally describe the coverage of a sensor.

Definition 3.1: The *coverage of a sensor* $\mathtt{s}$ with planar coordinates $(\mathtt{x}, \mathtt{y})$ and sensing range $\mathtt{r}$ is a disk with center $(\mathtt{x}, \mathtt{y})$ and radius $\mathtt{r}$. We call this disk the *coverage or sensing disk*, and call its border the *coverage or sensing circumcircle*, denoted by $\mathcal{C}(\mathtt{s})$. We say that a point $\mathtt{p}$ is covered by a sensor $\mathtt{s}$ if $\mathtt{dist}(\mathtt{s}, \mathtt{p}) \leq \mathtt{r}$.

The coverage of a network follows naturally as the union of the coverage disks of all the sensors in the network. Formally,

Definition 3.2: The *coverage of a network* is the area $\mathcal{A}$ with the property that for any point $\mathtt{p} \in \mathcal{A}$, there exists at least one sensor $\mathtt{s}$ in the network such that $\mathtt{p}$ is covered by the coverage disk of $\mathtt{s}$.

We naturally define a redundant sensor in the following manner:

Definition 3.3: A sensor is redundant if its sensing disk is *completely* covered by other sensors.

Before introducing the main result, we define a few additional terms:

Definition 3.4: The 2-Voronoi diagram of a sensor $\mathtt{s}$ is the Voronoi diagram of the Voronoi generators of $\mathtt{s}$, when $\mathtt{s}$ is excluded. The 2-Voronoi Vertices (2-VV) of a sensor $\mathtt{s}$ are the Voronoi vertices of the 2-Voronoi diagram of $\mathtt{s}$. A 2-Voronoi Intersection Point (2-VIP) of $\mathtt{s}$ is the intersection between an edge of the 2-Voronoi diagram and the coverage circumcircle of $\mathtt{s}$. A 2-Voronoi edge (2-VE) of $\mathtt{s}$ is either a Voronoi edge between 2-VVs of $\mathtt{s}$, or a Voronoi edge between a 2-VV and a 2-VIP of $\mathtt{s}$.

Fig. 4 illustrates an example of a redundant sensor. In the example, sensor $\mathtt{s}_1$ has five 2-VEs, one between two 2-VVs, $2-\mathtt{V}_1$ and $2-\mathtt{V}_2$, and the rest between a 2-VV and a 2-VIP of $\mathtt{s}_1$. The next lemma shows an important relationship between the Voronoi neighbors of a sensor and their coverage disks.

Lemma 3.1: The Voronoi generators of a sensor $\mathtt{s}$, $\mathtt{G}_\mathtt{s}$, are the ones closest to it. Therefore, there is no other sensor that covers part of $\mathtt{s}$'s coverage disk that is not already covered by the sensors in $\mathtt{G}_\mathtt{s}$.

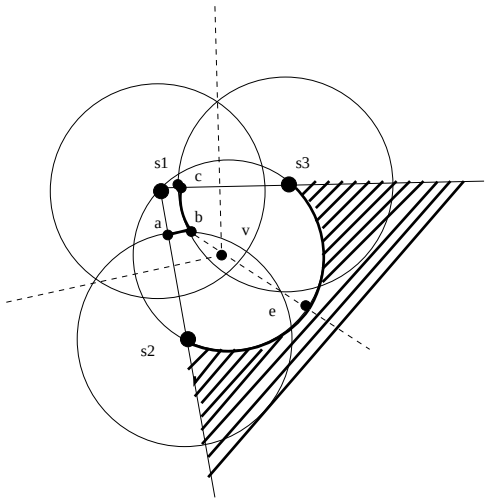


Fig. 3. Proof of Lemma 3.1. If sensors s_1, s_2 and s_3 are mutual Voronoi generators, another sensor can only be placed in the hashed area. Otherwise, that sensor would be a Voronoi generator of s_1.

Proof: Let us consider the case where sensor s_1 has generators s_2 and s_3 whose coverage areas intersect each other and also the coverage area of s_1, see Fig 3. We assume that all the sensors have the same coverage range, r. Let v be the Voronoi vertex generated by the three sensors. Fig. 3 also shows the circumcircle of sensors s_1, s_2 and s_3, centered at v, containing no other sensor. Let e be the intersection point between this circle and the Voronoi edge generated by s_2 and s_3. The only area where another sensor, that is not a Voronoi generator of s_1, can be placed, is the hashed area. Observe that $dist(b, e) = dist(b, v) + dist(v, e)$, so due to the triangle inequality, $dist(b, e) = dist(b, v) + dist(s_3, v) > dist(s_3, b) = r$. Also, $dist(s_2, a) = dist(s_2, b) = r$. It is easy to prove then that the distance between any point on the arc $\widehat{ab}$ and a point on the arc $\widehat{s_2 e}$ is larger than or equal to r. The arcs are emphasized in Fig. 3. Similarly, the distance between any point on the arc $\widehat{bc}$ and any point on the arc $\widehat{s_3 e}$ is larger than or equal to r. Hence, any sensor placed in the hashed area covers less of s_1's coverage area than s_2 and s_3. The cases where the coverage areas of s_2 and s_3 do not intersect, or do not interesect the coverage area of s_1 can be similarly proved. ■

The following theorem, the main result of this section, translates the problem of finding a redundant sensor to a local examination of the sensor's Voronoi neighbors.

Theorem 3.1: A sensor s is redundant if and only if all the 2-VVs and 2-VIPs of s are covered by the Voronoi generators of s.

Proof: **If** a sensor s is redundant, then all its 2-VVs and 2-VIPs are covered by the Voronoi generators of s. This is illustrated in Fig. 4, that shows an example of a redundant sensor, s_1. Since the coverage area of s_1 is completely covered by other sensors, using Lemma 3.1, we infer that it is completely covered by the Voronoi generators of s_1.

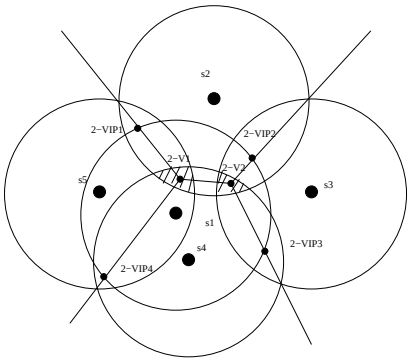


Fig. 4. Example redundant sensor, s_1. Points $2-V_1$ and $2-V_2$ are 2-VVs of s_1, and $2-VIP_{1..4}$ are 2-Voronoi Intersection Points of s_1. Note that $2-V_{1,2}$ and $2-VIP_{1..4}$ are all covered by at least two of the Voronoi neighbors of s_1.

Also, since the coverage area of a sensor is a circle, any three generators of s_1 that are mutual generators when s_1 is excluded, will cover a common area. Fig. 4 shows the common areas of generators s_2, s_4, and s_5, and s_2, s_3, and s_4, respectively, as the hashed areas. The common area of such three-generators contains the Voronoi vertex generated by them. This Voronoi vertex is a 2-VV of s_1, and is therefore covered by three Voronoi generators of s_1. In a similar fashion it can be proved that each 2-VIP of s_1 is covered by at least two of the Voronoi generators of s_1.

Only if all the 2-VVs and 2-VIPs of a sensor are covered by the sensor's Voronoi generators, the sensor is redundant. Fig. 4 illustrates the proof. The 2-VVs, 2-VEs, and 2-VIPs of s_1 define a partition of the coverage area of sensor s_1, consisting of four regions. Each region of the partition is associated with a Voronoi generator of s_1. Since the 2-VVs and 2-VIPs of s_1 are covered by the Voronoi generators of s_1, following the definition of Voronoi diagrams, see Section II, each Voronoi generator of s_1 covers the 2-Voronoi vertices, 2-VIPs, and 2-VEs that it generates. Thus, the region of the partition associated with a Voronoi generator of s_1 is completely covered by that generator, making s_1 redundant. ■

A. Distributed detection of redundant sensors

If each sensor knows the position of its Voronoi generators, it can easily detect its redundancy. Under specific assumptions on the network (bounded degree nodes), the expected number of Voronoi generators is constant, and the computation of the associated 2-Voronoi diagram takes constant time. The sensor then checks in constant time if its 2-VVs and 2-VIPs are covered by the Voronoi neighbor that generated them. This verification takes constant time since the number of 2-VVs and 2-VIPs of a sensor is $O(g)$, where g is the number of Voronoi generators of the sensor.

B. Blind Points

If two redundant sensors, that are also Voronoi neighbors, both decide to turn off, an area between them may be left

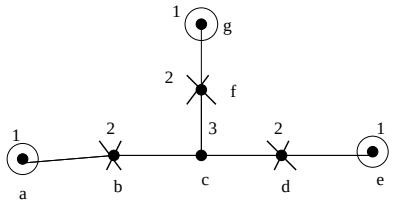


Fig. 5. Example of a redundant graph of a network – the numbers associated with the nodes represent their degree. The circled nodes represent winners in the first round, and the crossed nodes represent their direct neighbors, losers. Node c is not a loser in the first round since none of its neighbors is a winner, but it is a winner in the second round.

uncovered. Such an area is called a blind point [1]. We need to find the maximum number of redundant sensors that can be turned off without leaving blind points.

One solution to this problem, proposed by Tian and Georganas [1], uses a random back-off scheme. We propose a different solution, based on a slight modification of a distributed approximation of the maximal independent set (MIS) problem [5]. Let $\mathtt{G_R} = (\mathtt{V_R}, \mathtt{E_R})$ be the redundant graph of the network, where $\mathtt{V_R}$ is the set of redundant sensors. There is an edge $\mathtt{e} \in \mathtt{E_R}$ between two redundant sensors if and only if they are Voronoi neighbors. Then, the blind point problem is equivalent to one of finding the maximum independent set of the redundant graph, $\mathtt{G_R}$.

Our algorithm, similar to the one proposed by Luby [5], proceeds in rounds. In each round, a redundant sensor sends to its redundant Voronoi generators, a message containing its identity and the number of its redundant Voronoi generators. When a redundant sensor receives such a message from all its redundant Voronoi generators, it compares its value with the values received. A sensor that has the smallest value is a winner. A winner sends to all its redundant Voronoi generators, a message stating that it is a winner. A redundant sensor that receives such a message from one of its redundant Voronoi generators becomes a loser. At the end of each round, the winners are turned off, and together with the losers, do not participate in the following round. Luby [5] proves that a variant of this algorithm terminates and the expected number of rounds is $\mathtt{O}(\log \mathtt{n})$.

Fig. 5 shows an example of a redundant graph of a network. After the first round, sensors a, e, and g are winners and are turned off, and sensors b, d, and f are losers. In the second round, c is the unique participant, and a winner.

C. Management of Redundant Sensors

A winner in the above protocol can be safely turned off, since none of its redundant Voronoi neighbors is turned off. This is a special case of a sensor failure, and in Section V-B we provide an algorithm for correctly updating the local Voronoi information of sensors affected by sensor failures. The algorithm can be easily adapted to this situation, the only difference being that the affected sensors can be notified of the "failure", and do not have to discover it themselves.

A sensor that has been turned off, is periodically reawakened in order to check the presence of its Voronoi generators. If one (or more) of them has failed, the reawakened sensor recomputes its redundancy information. This is a special case of a new sensor joining the network, and the protocol is similar to the one described in Section 6.

IV. Planar Coverage Boundary Problem

A problem closely related to the coverage-preserving, energy-efficient redundancy elimination problem is one of finding planar coverage boundary. It is easy to see that a sensor is redundant iff its removal does not alter the boundary of the network. In this section we formally define the coverage-boundary problem and provide an efficient distributed solution. Using Definition 3.1, we say that a sensor is on the boundary of the coverage of the network iff the circumcircle of its sensing disk is not entirely covered by the coverage disks of all the other sensors in the network. Formally,

Definition 4.1: A sensor $\mathtt{s}$ is said to be *on the boundary* of the coverage of a network if there exists a point $\mathtt{p}$ on $\mathcal{C}(\mathtt{s})$ such that $\mathtt{p}$ is not covered by the coverage disk of any other sensor in the network.

With this definition in place, we define the *coverage-boundary* problem as the problem of finding all the sensors that are on the boundary of the network's coverage. Theorem 4.1 establishes a lower bound for any solution of the coverage-boundary problem. Its proof, based on a linear-time transformation from the set equality problem, can be found in [4].

Theorem 4.1: The coverage-boundary problem has a $\Omega(\mathtt{n} \log \mathtt{n})$ lower bound.

Our solution of the problem is based on the following theorem, whose proof can be found in [4].

Theorem 4.2: A sensor $\mathtt{s}$ is on the boundary of the network if and only if the Voronoi cell of $\mathtt{s}$ is not completely covered by its sensing range.

The coverage-boundary sensors enjoy a special relationship with the redundant sensors. More precisely, a redundant sensor is *not* a boundary sensor, since by Definition 3.3, its circumcircle is also completely covered by other sensors. The following theorem, describes another important property of redundant sensors.

Theorem 4.3: If a redundant sensor is turned off, none of the remaining sensors that are not on the coverage-boundary will be transformed into a coverage-boundary sensor.

A. Local Computation of Boundary Sensors

Each sensor, knowing only the position information of its Voronoi generators, can discover its presence on the sensor network coverage-boundary in the following manner: generate the Voronoi diagram of only itself and its Voronoi generators; this takes constant time, since the expected number of Voronoi generators is constant. Check if its distance to each of the Voronoi vertices generated is less than the sensing range. This takes constant time, since the number of Voronoi vertices

generated is $O(g)$, where g is the expected number of Voronoi generators per sensor. It is easy to see that the serial version of this algorithm has $O(n \log n)$ complexity, where n is the total number of sensors.

V. Distributed Computation and Maintenance of Voronoi Cells

The resource and scalability constraints of an ad-hoc network make the existence of a centralized entity, that would compute the global Voronoi tesselation, an unreasonable assumption. Instead, every sensor must keep enough data to allow for the local computation of the desired information. Our goal is to permit each sensor to correctly determine its Voronoi cell and the identity of its Voronoi neighbors. This information is sufficient to autonomously decide its own presence on the boundary, according to Theorem 4.2, or its redundancy, according to Theorem 3.1.

Starting with each sensor knowing only its own location, and using a heartbeat algorithm to exchange local information between neighboring sensors, each sensor learns of the network in k steps, where k is the diameter of the network. Each sensor can then compute its set of Voronoi generators from the information collected, and also information about the intermediate hops necessary to route to them. It then discards the information about all the other sensors.

A. Deployment of New Sensors

The deployment of new sensors, potentially changes the network's coverage-boundary and the set of redundant sensors. In both cases, this boils down to updating the local Voronoi information of the affected sensors. Fig. 6 illustrates an example where the new sensor ns joins an existing network. Only sensors whose Voronoi cell is affected have to be notified about the new sensor. It is well known that the circumcircle of a Delaunay triangle contains no other site (sensor). We call such a circle a Delaunay circle. Hence, only sensors that generate a Delaunay triangle whose Delaunay circumcircle contains the new sensor, must be notified. We say that such a triangle is *in conflict* with the new sensor, and the sensors that generate the triangle are called *notifiables.* All the sensors in Fig. 6 are notifiable with regard to ns.

Before proceeding with the description of the join algorithm, we present two useful lemmas, direct consequences of Lemmas 4.3 and 4.4 from [6].

Lemma 5.1: Given a randomly deployed sensor network of size n, the expected number of sensors affected by the random deployment of a new sensor is $O(\log n)$.

Lemma 5.2: The expected number of Voronoi neighbors of a sensor in a randomly deployed sensor network is constant.

a) The Join Algorithm.: When a new sensor is deployed, it starts by periodically sending a beacon message. Its presence can be detected by another sensor only if the radio transmission ranges of the two sensors exceeds the distance between them. Since more then one sensor can detect the new sensor, we choose the one whose Voronoi cell contains the new sensor to be the *introducer*. The goal of the introducer is to notify other affected sensors about the new sensor. In Fig. 6(a), the introducer of ns is s_1. If the new sensor is connected to the network, according to the definition of a Voronoi cell (see Section II), our choice guarantees that the introducer will detect the new sensor.

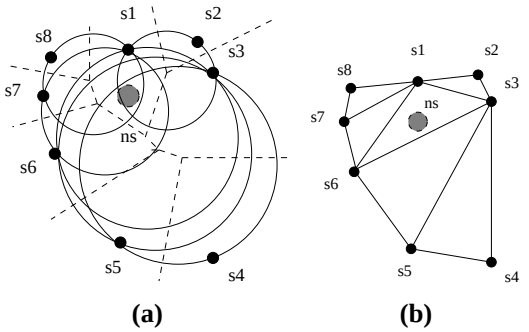


Fig. 6. Example of new sensor deployment: the gray circle, ns, represents the new sensor. (a) The Voronoi diagram of the old sensors. The circles denote circumcircles of Delaunay triangles. (b) The Delaunay triangulation of the old sensors. Sensor s_1 sends a message only to sensors s_2 and s_8. When sensor s_3 receives the notification from sensor s_2, it first constructs its list of notifiables, consisting of sensors s_2, s_4, s_5, s_6, and s_1. It then colors sensor s_2 gray, since sensor s_2 is the one that sent it the notification. It then traverses the list of notifiables. In the first traversal it colors sensor s_1 gray. We note that this is natural, since we already know that sensor s_2 is closer to sensor s_1 than sensor s_3. In the next traversal, sensor s_3 colors sensor s_6 gray, since $\mathtt{dist}(s_3, s_6) > \mathtt{dist}(s_1, s_6)$. The next traversal colors sensor s_5 gray and the last colors sensor s_4 gray. When sensor s_6 receives the notification from sensor s_7, that received it from sensor s_8, it constructs its list of notifiables, s_7, s_1, s_3, s_5. It marks sensor s_7 gray and propagates the gray color to sensors s_1 and s_3. It then marks sensor s_5 black and notifies it. The notification similarly reaches sensor s_4.

The introducer initiates the notification process, and during the process, each notified sensor uses only local information to decide its behavior. Each sensor has a list of incident Delaunay triangles, and the introducer, s_{intro}, traverses its list of Delaunay triangles, identifies all those that are in conflict with the new sensor, and isolates the notifiables among its Voronoi neighbors. s_{intro} then sorts the notifiables based on its Euclidean distance to them. Each sensor uses three colors to differentiate the notifiables in its local view. Initially, all the notifiable Voronoi neighbors are $white$. Upon completion of the coloring algorithm, they will be either $gray$ or $black$, and the sensor, in this case s_{intro}, will only contact the black ones.

The coloring algorithm proceeds as follows. First, s_{intro} takes its closest white notifiable, marks it black, and removes it from the white list. Then, it traverses the white list in clockwise order starting from the newly removed notifiable. If a white notifiable, w, is part of a local Delaunay triangle that has a gray or black vertex, gb, such that $\mathtt{dist}(s_{intro}, w) > \mathtt{dist}(gb, w)$, it marks w gray and removes it from the white list. If, at the end of the traversal, a notifiable has been marked gray, the traversal is repeated, until no more white notifiables are marked gray. Then, if the white list is not empty, the closest white notifiable is again removed and marked black and the above process is repeated. In the

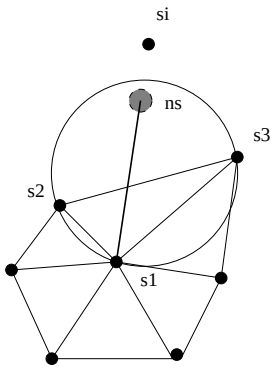


Fig. 7. Illustration of the correctness proof for the join algorithm. ns is the added sensor, s_i is the introducer, s_2 and s_3 are notifiables at Delaunay distance $l-1$ from s_i, and s_1 is a notifiable at Delaunay distance l from s_i.

example from Fig. 6, sensor s_1 first colors sensor s_2 black and in the subsequent traversal colors sensor s_3 gray. It then colors sensor s_8 black, and in the first traversal colors sensor s_7 gray. A second traversal colors sensor s_6 gray.

On completion of the coloring phase, s_{intro} sends a notification to each black notifiable. For a notifiable b, the message contains the identity and position of the new sensor, the identity of the notificator, in this case s_{intro}, and the identities of all the gray notifiables in the local view of s_{intro}, that b needs to notify. Each sensor that receives a notification concerning a new sensor, drops it if it has already received the same notification from another Voronoi neighbor. However, if this is the first time that it receives a notification about this new sensor, it creates its own white list of notifiable Voronoi neighbors. It then removes from this list all the gray sensors contained in the notification, and marks them black, since it has to notify them. It also removes from the white list the notificator and marks it gray, since it should not notify it. It then repeats the coloring phase described above, and sends to each black notifiable a notification with the same structure as the one that itself has received. Figure 6 illustrates an example of the notification process.

The local coloring and propagation of gray is meant to reduce the number of redundant messages. Sensor s_3 will not notify sensors s_6, s_5 and s_4, since sensor s_8 will notify sensor s_7, that in turn will notify sensor s_6, and so on. Of course this method will not eliminate all redundant notifications, but by locally reducing the distance between a notifier and a notifiable, this method has the advantage of reducing the number of actual messages that need to be sent for a notification. This is because the chance of two Voronoi neighbors of being in each other's radio transmission range increases when their Euclidean distance decreases.

We present several properties of the join algorithm.

Correctness 1: Upon completion of the algorithm, all the notifiable sensors have been notified.

Proof: By induction on the Delaunay distance between a notifiable and the introducer. Fig. 7 illustrates the proof, where ns represents the new sensor. The basis case is simple, since all notifiables at distance 1 from the introducer are clearly notified. For the induction step, we assume that all notifiables at distance $l-1$ are notified. Let s_1 be a notifiable at Delaunay distance l from the introducer. Being a notifiable, s_1 has at least a triangle in conflict with the new sensor, ns. Since in terms of the Euclidean distance s_2 and s_3 are s_1's closest Voronoi neighbors to ns, the triangle $\Delta s_1 s_2 s_3$ is such a conflict triangle. Sensor s_1 is at Delaunay distance l from the introducer, s_i, therefore it has at least one Voronoi neighbor at Delaunay distance $l-1$ from s_i. Since s_i is the closest sensor to ns, no other Voronoi neighbor of s_1 is at a smaller Delaunay distance from s_i than s_2 and s_3. Hence, at least one of s_2 or s_3 are at Delaunay distance $l-1$ from s_i. If both are, since both are notifiable, they will both be notified. At least the one closer to s_1 will then notify it, since both will detect the triangle $\Delta s_1 s_2 s_3$ to be in conflict with ns. If only one of s_2 or s_3 is at distance $l-1$, then that one will notify s_1. ■

Termination 1: The algorithm will terminate, i.e., notifications will not be sent indefinitely.

Proof: Each notifiable sensor can receive notifications only from its Voronoi neighbors that are also notifiable. Only notifiable sensors receive notifications. A notifiable sensor propagates a notification only to notifiable Voronoi neighbors, and only when receiving for the first time a notification concerning a new sensor. Since the number of expected notifiables is bounded (Lemma 5.1), and each propagates a notification only once, the algorithm will terminate. ■

Complexity 1: The expected number of notifications for a join is $O(\log n)$, where n is the number of sensors already in the network.

Proof: For a new sensor, the expected total number of notifiables is $O(\log n)$, see Lemma 5.1, and each notifiable receives notifications only from its Voronoi neighbors that are also notifiables. Since the expected number of Voronoi neighbors of a sensor is constant, see Lemma 5.2, the expected number of notifications is $O(\log n)$. ■

B. Sensor Failures

Sensor failures, like deployment of new sensors, also require the modification of local Voronoi neighbors of affected sensors. The following lemma identifies the sensors affected by a single failure, and limits the size of their set of candidate replacement Voronoi generators.

Lemma 5.3: A single sensor failure affects the local Voronoi information belonging only to the Voronoi neighbors of the failed sensor. Furthermore, the set of candidates for the position of new Voronoi neighbors of each affected sensor consists solely of the Voronoi neighbors of the failed sensor.

Proof: The direct Voronoi neighbors of the failing sensor are clearly affected, since one of their Voronoi neighbors disappears. To see why other sensors are not affected, consider the following: let f_2 be a failed sensor and s_6 be a sensor

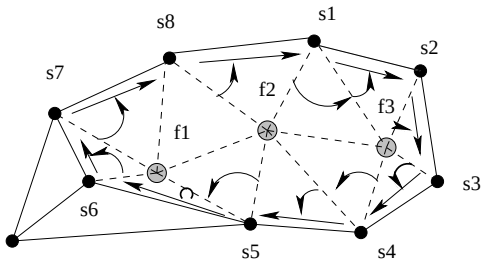


Fig. 8. Example of network in which sensors $\mathtt{f}_1$, $\mathtt{f}_2$ and $\mathtt{f}_3$ fail simultaneously. The round arrows illustrate the local decisions made by intermediate sensors, and the straight arrows illustrate the trajectory of the DISC messages. Let's say that sensor $\mathtt{s}_7$ detects that $\mathtt{f}_1$ has failed. Sensor $\mathtt{s}_7$ creates a DISC message containing $\mathtt{f}_1$ in the failed list, and sends it to sensor $\mathtt{s}_8$. Before sending to $\mathtt{f}_2$, sensor $\mathtt{s}_8$ detects that $\mathtt{f}_2$ has also failed, and adds $\mathtt{f}_2$ to the failed list. Node $\mathtt{s}_8$ then forwards the DISC message to sensor $\mathtt{s}_1$. Similarly, sensor $\mathtt{s}_1$ adds $\mathtt{f}_3$ to the failed list and forwards the message to sensor $\mathtt{s}_2$. When sensor $\mathtt{s}_4$ receives this DISC message, it detects that $\mathtt{f}_2$ and $\mathtt{f}_3$ are already in the failed list, so it forwards the message to sensor $\mathtt{s}_5$, without modifying the failed list. All the intermediate sensors add their identities and positions to the affected list, so when sensor $\mathtt{s}_7$ receives the DISC message that it initiated, it is able to correctly recompute its Voronoi generators and incident Delaunay triangles.

that is not a Voronoi neighbor of $\mathtt{f}_2$ (see Fig. 8). Since its Voronoi neighbors are intact, by definition, the Delaunay circles generated by sensor $\mathtt{s}_6$ contain no other sensors inside. Hence sensor $\mathtt{s}_6$ will not acquire new Voronoi neighbors. The same argument can be used to prove that the affected sensors will have to consider as candidates for new Voronoi neighbors, only the set of affected sensors. For example, sensor $\mathtt{f}_3$ must only consider sensors $\mathtt{s}_1, \mathtt{s}_4, \mathtt{s}_5, \mathtt{f}_1, \mathtt{s}_8$ to replace the position left by $\mathtt{f}_2$. It need not consider sensor $\mathtt{s}_6$, since $\mathtt{s}_6$'s Voronoi neighbors are not changed by $\mathtt{f}_2$'s failure. ■

b) Departure Algorithm.: We now present the actions taken when one or more sensors fail, possibly simultaneously. Each sensor S periodically sends a beacon to its Voronoi generators. For each Voronoi generator, S has a timeout value that is set whenever a beacon is received from the respective neighbor. The timeout value is an upper bound on the period of the beacon plus the round-trip time for that generator. This value varies since a sensor may have multiple intermediate sensors on the path to its Voronoi neighbors. If the timeout of a generator F expires before the respective beacon is received, S declares F failed and initiates a protocol to discover its new generators. For this, it creates a message of type DISC, containing the identity of S, a sequence number maintained by S and two lists. The sequence number is incremented each time a failure is detected by S. The first list, called the *failed list*, contains identities of sensors that are assumed to have failed, and initially contains only F. The second one, called the *affected list*, is empty, but eventually collects the identities of the sensors affected by the failure of the sensors in the failed list. Node S sends the DISC message to its first Voronoi neighbor in counterclockwise order from F.

A sensor N that receives a DISC message, is a Voronoi neighbor of F. Consequently, it adds its identifier to the affected list of the received DISC message. Node N then looks at the last item in the failed list, let that be called G. Node N finds its first counterclockwise Voronoi generator starting from G, let this be called H. If sensor H is locally considered by sensor N to be failed, and is not already in the failed list of the received DISC message, sensor N adds H to the failed list and repeats this process for the next counterclockwise Voronoi neighbor starting from H. The modified DISC message is forwarded to the first generator of N, in counterclockwise order from G, that has not failed.

This procedure, similar to walking in a labyrinth with the left hand touching the wall, finds the smallest perimeter enclosing a cluster of simultaneously failed sensors, and reaches *all* the sensors affected by the cluster's failure. When the initiator S receives the DISC message that it has initiated, it recomputes its Voronoi diagram and dual Delaunay triangulation using its local information and the information in the affected list received with the DISC message. Fig. 8 illustrates an example of a network where three sensors, $\mathtt{f}_1, \mathtt{f}_2, \mathtt{f}_3$, fail simultaneously.

Each generator of a failed sensor must detect the absence of the failed sensor and generate a DISC message. The total number of messages sent is therefore proportional to the square of the number of Voronoi neighbors of the failed sensors. The number of messages could be linear, if the ring of affected sensors would have a leader. The leader could be chosen by each sensor, during its lifetime, to be its closest Voronoi generator. This sensor, called monitor, would then watch over its target's presence. In case of failure detection, the monitor would send two circular DISC messages, the first one collecting the information of all the affected sensors, and the second one propagating this list to all of them. However, the simultaneous failure of a sensor and its monitor would break the protocol, since not all the affected sensors would be notified.

We now prove several properties of the departure algorithm. Let $\mathtt{G}_\mathtt{F} = (\mathtt{V}_\mathtt{F}, \mathtt{E}_\mathtt{F})$ be the failure graph, where $\mathtt{V}_\mathtt{F}$ is the set of failed sensors, and $\mathtt{e} \in \mathtt{E}_\mathtt{F}$ is an edge between two sensors $\mathtt{f}_1$ and $\mathtt{f}_2$ in $\mathtt{V}_\mathtt{F}$ if $\mathtt{f}_1$ and $\mathtt{f}_2$ are Voronoi neighbors. For every connected component C in $\mathtt{G}_\mathtt{F}$, let $\mathtt{A}_\mathtt{C}$ be the set of Voronoi neighbors of the sensors in C, that are not themselves in C. That is, $\mathtt{A}_\mathtt{C}$ is the set of sensors affected by the failure of the sensors in the connected component C. In Fig 8, f_1, f_2 and f_3 form a connected component C of the failure graph, and $\mathtt{A}_\mathtt{C} = \{\mathtt{s}_1, .., \mathtt{s}_8\}$. With these definitions, a generalization of Lemma 5.3 can be easily proved.

Lemma 5.4: A connected component of the failure graph, C, affects only the Voronoi diagram of its Voronoi neighbors, $\mathtt{A}_\mathtt{C}$. Furthermore, the set of candidates for the new Voronoi generators for each affected sensor in $\mathtt{A}_\mathtt{C}$ is contained in $\mathtt{A}_\mathtt{C}$.

We can now prove the correctness of our protocol.

Correctness 2: On completion of the departure algorithm, the sensors affected by the failures will correctly compute their new Voronoi neighbors.

Proof: A sensor S that is affected by the failure of a sensor F, part of a connected component C of the failure

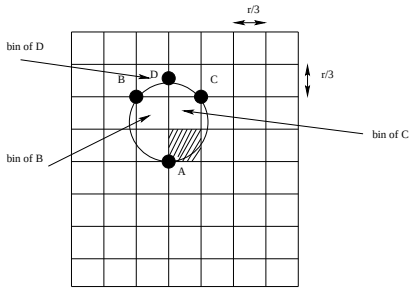


Fig. 9. Proof of Theorem 5.1 – divide the area in $\mathtt{r}/3 \times \mathtt{r}/3$ bins. Each bin contains at least one sensor. The hashed bin contains sensor A.

graph G_F, will send a DISC message that will traverse all the sensors in A_C. The message will collect information about all the traversed sensors. Using Lemma 5.4, we conclude that when S receives the DISC message that it initiated, it will have all the information necessary for computing the new Voronoi neighbors. ■

Termination 2: The departure algorithm terminates.

Proof: The number of sensors that send DISC messages is equal to the number of sensors affected by the sensor failures. Each DISC message will traverse only sensors that are affected by failures. ■

Complexity 2: The total number of messages necessary to recompute the Voronoi neighbors of the sensors in the network, due to $|V_F|$ failures is $O(|V_F|^2)$.

Proof: The number of messages is upper bounded by the square of the number of sensors affected by the failures. The number of sensors affected by $|V_F|$ failures is $O(|V_F|)$, using Lemma 5.2. ■

C. Routing to Voronoi Generators

We have assumed, until now, that the notification of the sensors affected by failures or new sensor deployments is done using direct messages between Voronoi neighbors. In other words, we have assumed that routing is done along the edges of the Delaunay triangulation. However, two Voronoi neighbors may not be within each other's range, requiring a routing protocol. LAR [7], DREAM [8] and GPSR [9] are examples of location based routing protocols that can be used for routing between non-adjacent Voronoi neighbors. However, in the following theorem, based on [10], we provide a lower bound on the radio transmission range of sensors, that ensures that sensors that are Voronoi neighbors are almost surely within each other's range.

Theorem 5.1: Let $\mathtt{n}$ be the number of sensors randomly placed in a square of area $\mathtt{S}$, each sensor having a radio transmission range $\mathtt{r}$. Then there exists a constant $\mathtt{c} > 9$ such that if $\mathtt{r} \geq \sqrt{\frac{\mathtt{c}\mathtt{S}\log \mathtt{n}}{\mathtt{n}}}$, then any two sensors $\mathtt{s}_1$ and $\mathtt{s}_2$ that are Voronoi generators are almost surely within each other's transmission range (i.e., $\mathtt{Pr}(\mathtt{dist}(\mathtt{s}_1, \mathtt{s}_2) \leq \mathtt{r}) \rightarrow 1$ when $\mathtt{n} \rightarrow \infty$).

Proof: We divide the square of area $\mathtt{S}$ into square bins of size $\mathtt{r}/3$. There are $\frac{9\mathtt{S}}{\mathtt{r}^2}$ bins. Similar to [10], the probability that a bin is empty after throwing $\mathtt{n}$ balls(sensors) in the initial square is $(1 - \mathtt{S}\mathtt{r}^2/9)^{\mathtt{n}} \leq \mathtt{e}^{-\mathtt{n}\mathtt{S}\mathtt{r}^2/9} \leq \mathtt{e}^{-\mathtt{c}\mathtt{S}^2 \log \mathtt{n}/9} = \mathtt{n}^{-\mathtt{c}\mathtt{S}^2/9}$. The expected number of empty bins is then $\frac{9\mathtt{S}}{\mathtt{r}^2}\mathtt{n}^{-\mathtt{c}\mathtt{S}^2/9} = \frac{9}{\mathtt{c}\log \mathtt{n}}\mathtt{n}^{1-\mathtt{c}\mathtt{S}^2/9} \leq \mathtt{S}\mathtt{n}^{1-\mathtt{c}\mathtt{S}^2/9}$, if $\mathtt{c} \geq 9$. The expected number of empty bins then tends to 0, when $\mathtt{n}$ tends to infinity.

If every bin contains at least one sensor, it is easy to see that each sensor will be in the transmission range of all the sensors in the adjacent bins (the largest distance between two sensors in adjacent bins is $\mathtt{r}\sqrt{5}/3 \leq \mathtt{r}$) . To see how far two Voronoi neighbors can be, Fig. 9 shows the Delaunay circle with the largest diameter that a sensor can form with the sensors in two adjacent bins. The circle "bites" into the bins of sensors D and E. In order for D or E to be Voronoi generators of A, they need at least to be inside the Delaunay circle of A, B and C, which has a diameter of $5\mathtt{r}/6 < \mathtt{r}$. Thus, D and E are covered by A. This proves that a sensor tends asymptotically to be in the transmission range of all its Voronoi generators. ■

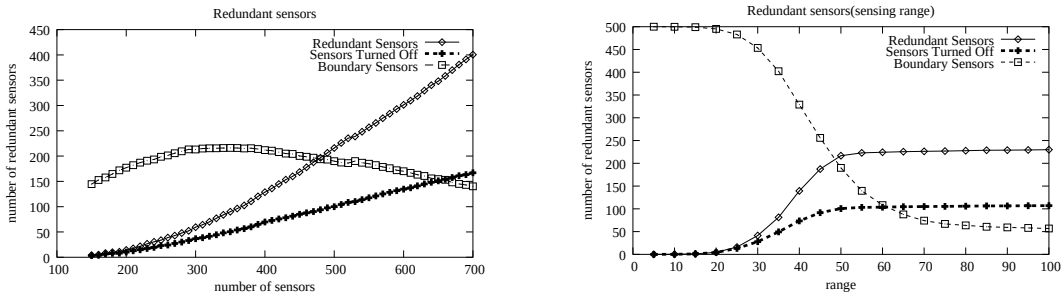
VI. Simulation Results

In this section we present the details of our simulations, performed using our Java based simulation testbed. All experiments are performed by randomly placing identical sensors in a square of size $1000 \times 1000\mathtt{m}^2$. In Section VI-A we investigate the dependencies between the number of redundant and coverage-boundary sensors and the total number of sensors deployed and their sensing range. In Section VI-B we measure the communication traffic generated by join and leave operations.

A. Coverage-Boundary and Redundant Sensors

We investigate the evolution of three metrics: the number of coverage-boundary sensors, the total number of locally detected redundant sensors, and the total number of sensors that can be simultaneously turned off. For all the experiments in this section, we randomly generate 10 different sensor network configurations, and present only the average results. In the first experiment we assume that all the sensors have the same *sensing range* of 50m. We measure the dependency between the three metrics and the number of sensors, by increasing the number of sensors deployed from 150 to 700.

Fig. 10(a) shows the results of this experiment. Initially, all the 150 sensors are on the boundary. If the number of coverage-boundary sensors experiences an initial increase, it quickly saturates, and then steadily decreases to 140 for 700 total sensors. This is because as the sensor density increases, the number of internal sensors also increases, but initially slower than the total number of deployed sensors. On the other hand, as expected, the total number of redundant sensors is always larger than the number of possible simultaneous turn-offs, as detected by Luby's [5] algorithm. However, the algorithm is scalable, since the number of possible simultaneous turn-offs grows linearly with the number of sensors.



(a) **(b)**

Fig. 10. Plots showing the evolution of the number of boundary sensors, redundant sensors and sensors that can be turned off.

The second experiment measures the evolution of the same metrics for sensor networks of 500 sensors, when the sensing range is increased from 5 to 100. Fig. 10(b) shows the results. When the range is smaller than 20, all the metrics have extreme values. However, for larger values, the number of boundary sensors decreases drastically to almost 10% of the total number of sensors. The number of simultaneous turn-offs however, saturates quickly at 20%, when the sensing range is around 50. Consequently, a relatively small sensing range is enough to detect most of the possible simultaneous sensor turn-offs.

B. Network Load

We investigate the traffic generated in the network by insertions of new sensors and failure of existing ones. The messages generated are necessary to recompute the local Voronoi diagrams, and with it, the coverage-boundary and the redundant sensor information. As before, we assume that all the sensors have similar capabilities, which in this section is expressed by identical radio transmission ranges. Note that the experiments in Section VI-A only consider the sensing range.

In the first experiment, we initially place between 150 and 800 sensors in the square of size $1000 \times 1000\mathrm{m}^2$. All the sensors have the same radio transmission range, of 115m. To investigate the performance of the join algorithm, for each value of the number of sensors initially deployed, we generate 10 random network configurations, and for each configuration we insert a new sensor at 150 random positions. Similarly, for sensor failures, for each of the 10 random networks we randomly select 150 sensors to fail. We present only the average values over 1500 measurements.

Fig. 11(a) shows the results of this experiment. The average number of messages transmitted for each join/leave operation, decreases abruptly with increase in sensor density, and saturates at around 400 sensors. Even though a higher sensor density implies a larger number of Voronoi generators per sensor, increasing the number of sensors that need to be notified, it also implies shorter distances between Voronoi neighbors, thus fewer actual messages. Therefore, denser networks simplify the task of locally updating the Voronoi and coverage information.

In the second experiment, we place 500 sensors in the same square and increase the transmission range from 50 to 200. Similar to the previous experiment, the values reported are averaged over 1500 measurements. Fig. 11(b) presents the results. The average number of messages required per update decreases quickly and saturates at a transmission range of around 120. Energy-wise, there exists a tradeoff between the transmission range employed and the total number of messages required per update. Since the energy required per transmission increases super-linearly with the distance, shorter transmission ranges are preferred.

VII. Related Research

The problem of coverage of a set of entities has been studied in a variety of contexts. Tian and Georganas [1] present an algorithm for detecting sensors whose coverage area is completely covered by other sensors. A sensor turns itself off only when each sector of its coverage disk is covered by another sensor. Hence, this mechanism discovers only a subset of all the redundant sensors.

Zhang and Hou [2] provide a distributed algorithm for extending the network lifetime by turning off "redundant" sensors. Their mechanism for deciding a sensor to be redundant requires a sensor to divide its coverage area into small grids and then using a bitmap to indicate whether the center of each square of the grid is covered by some other sensor. Thus, this method provides a tradeoff for the grid granularity. For small values it becomes too complex, but for large values it may turn off sensors that are not redundant. Since only the neighboring sensors are probed for grid coverage, this work also finds only a subset of all the redundant sensors.

Ye et al. [3] present an algorithm that extends the network lifetime by maintaining a necessary set of working sensors and turning off redundant ones. A sensor is alternatively sleeping or active. When a sensor wakes up, if it has an active sensor inside its transmission range, it turns off again. Hence, their algorithm has no claim for maintaining the coverage of the network.

Slijepcevic and Potkonjak [11] introduce a centralized algorithm for finding the maximum number of disjoint subsets of sensors, where each subset completely covers the same area as the entire set of sensors. Meguerdichian et al. [12] define the coverage using as metrics the best covered and least covered paths between two sensors in the network. Haas [13] presents algorithms for optimizing coverage under constraints on message path length. Fang, Gao and Guibas [14] present

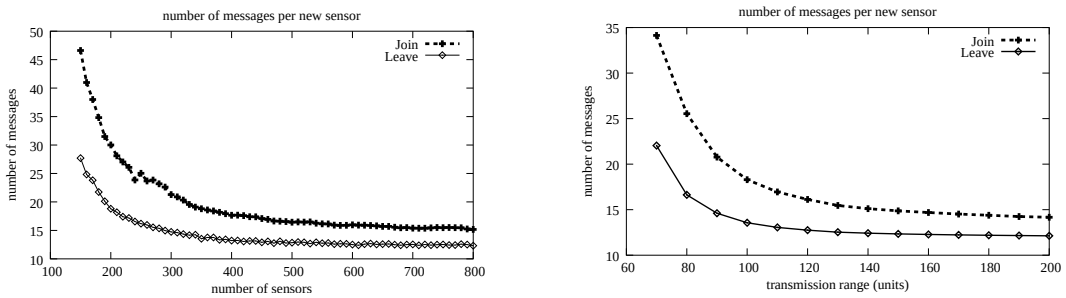


Fig. 11. Plots showing the total number of messages required for the deployment of new sensors.

an algorithm for routing around connectivity holes in a sensor network.

The problem of network coverage is related to the problem of frequency assignment in cellular networks [15], whose purpose is to assign a frequency for every base station in a centralized manner, such that no two base stations with the same frequency cover the same device. Our goal is quite different from the work mentioned above, namely to discover the coverage-boundary and to extend the lifetime of the network by eliminating redundant sensors. Moreover, our approach is distributed, that is, we do not assume global knowledge of the network.

The problem of sensor coverage has also received considerable attention in robotics [16]. Given a bounded environment the problem requires a robot equipped with a sensor to build a complete map of the environment without any initial knowledge. the robot to pass through specified points of the unknown region. notion of a *hierarchical generalized Voronoi graph* is used to incrementally construct the map using only line of sight data. For an extensive survey of coverage problems in sensor networks, please consult [4], [17].

Distributed computation of Voronoi diagrams is addressed by Stojmenovic [18] in the context of routing in ad-hoc networks and by Hu [19] in the context of topology control of ad-hoc networks. In their approach, a sensor builds a Voronoi diagram of itself and its neighbors. This computes only an approximation of the correct Voronoi cells of sensors, and therefore would produce incorrect information for the problems that we address.

VIII. Conclusions

In this paper, we have studied the problem of coverage-preserving, energy-efficient redundancy elimination for extending a network's lifetime, and the related coverage-boundary problem. We have reduced both problems to the computation of Voronoi diagrams and showed how to solve them using only local information. We have provided distributed and localized algorithms that allow sensors to update their view of the solution in cases of sensor failures and new sensor deployments. We have proved the correctness and termination properties of these algorithms. Our simulations show that the algorithms are efficient and scale well with the number of sensors.

References

[1] Di Tian and Nicolas D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41. ACM Press, 2002.

[2] Honghai Zhang and Jennifer Hou. Maintaining coverage and connectivity in large sensor networks. In *International Workshop on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless and Peer-to-Peer Networks*, Feb 2004.

[3] Fan Ye, Gary Zhong, Songwu Lu, and Lixia Zhang. Peas:a robust energy conserving protocol for long-lived sensor networks. In *23rd International Conference on Distributed Computing Systems (IEEE ICDCS)*, 2003.

[4] B. Carbunar, A. Grama, and J. Vitek. Coverage problems in sensor networks. Technical report, Purdue University, 2004. http://www.cs.purdue.edu/homes/carbunar/coverage.pdf.

[5] M Luby. A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 1–10. ACM Press, 1985.

[6] Olivier Devillers, Stefan Meiser, and Monique Teillaud. Fully dynamic delaunay triangulation in logarithmic expected time per operation. *Comput. Geom. Theory Appl.*, 2(2):55–80, 1992.

[7] Young-Bae Ko and Nitin H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of MobiCom*, pages 66–75, 1998.

[8] Stefano Basagni, Imrich Chlamtac, Violet R. Syrotiuk, and Barry A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of MobiCom*, 1998.

[9] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of MobiCom*, pages 243–254, 2000.

[10] S. Muthukrishnan and G. Pandurangan. The bin-covering technique for thresholding random geometric graph properties. Technical Report 2003-39, DIMACS, November 2003.

[11] Sasa Slijepcevic and Miodrag Potkonjak. Power efficient organization of wireless sensor networks. In *IEEE ICC*, 2001.

[12] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak. Exposure in wireless ad hoc sensor networks. In *Proceedings of MobiCom*, July 2001.

[13] Z.J. Haas. On the relaying capability of the reconfigurable wireless networks. In *IEEE 47th Vehicular Technology Conference*, volume 2, pages 1148–1152, May 1997.

[14] Q. Fang, J. Gao, and L. J. Guibas. Locating and bypassing routing holes in sensor networks. In *IEEE INFOCOM*, March 2004.

[15] G. Even, Z. Lotker, D. Ron, and S. Smorodinsky. Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. In *43rd Annual IEEE FOCS*, pages 691–700, 2002.

[16] Howie Choset. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[17] M. Cardei and J. Wu. *Coverage in Wireless Sensor Networks*. Handbook of Sensor Networks. CRC Press, 2004.

[18] I. Stojmenovic. Voronoi diagram and convex hull based geocasting and routing in wireless networks. Technical Report TR-99-11, University of Ottawa, December 1999.

[19] Limin Hu. Topology control for multihop packet radio networks. *IEEE Transactions on Communications*, 41(10):1474–1481, October 1993.