CERIAS Tech Report 2005-40

COVERAGE PROBLEMS IN WIRELESS SENSOR AND RFID SYSTEMS

by Bogdan Carbunar

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

COVERAGE PROBLEMS IN WIRELESS SENSOR AND RFID SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Bogdan Cărbunar

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2005

ACKNOWLEDGMENTS

I have been very fortunate to have Jan Vitek, Ananth Grama and Cristina Niţă-Rotaru as my advisors. I thank Prof. Jan Vitek for his constant support and advice, sometimes against hope and reason. I thank Prof. Ananth Grama for his help and advice in producing the material gathered in this thesis. I thank Prof. Cristina Niţă-Rotaru for providing a way to safety in troubled times. I thank Prof. Dan C. Marinescu for making the Ph.D. experience possible.

I greatly appreciate the collaboration of, in chronological order, Octavian Cărbunar, Marco Tulio de Oliveira Valente, Ioannis Ioannidis, Radu Sion, Murali Krishna Ramanathan and Mehmet Koyuturk. Part of this work, presented in Chapter 3 and Chapter 4, has been done in collaboration with Murali Krishna Ramanathan.

I would like to thank my parents, Mariana and Octavian Cărbunar, my grandmother, Lucretia Variu and grandfather Florian Cărbunar, for teaching me when they didn't know they were doing it. I thank my wife, Mihaela, for enabling my sanity during the Ph.D. process.

In addition, I thank, in alphabetical order, Lotzi Boloni for his movie nights and love of Florida. Ma Di for his quiet way of sharing a lab. Gergana Markova for great gifts, being a good listener and friendship. Mayur Naik for his great approach to movies, human interaction and cooking. Radu Sion for being an exceptional friend, room-mate, late night philosopher, cook, entertainer, mechanic and problem-solver.

TABLE OF CONTENTS

Page

LIST OF FIGURES v

ABSTRACT ix

1 Introduction 1

2 Coverage and Redundancy Detection in Sensor Networks 8

2.1 Overview of Voronoi Tessellations 11

2.2 Energy-Efficient Coverage 13

2.2.1 Distributed Detection of Redundant Sensors 17

2.2.2 Blind Points 18

2.2.3 Management of Redundant Sensors 19

2.3 Planar Coverage Boundary 20

2.3.1 Coverage-Boundary for Heterogeneous Sensor Networks 24

2.3.2 Local Computation of Boundary Sensors 24

2.4 Application of Voronoi Overlays to Distributed Hash Tables 25

2.5 Distributed Computation and Maintenance of Voronoi Cells 27

2.5.1 Initial Distributed Computation of Voronoi Generators 27

2.5.2 Deployment of New Sensors 28

2.5.3 Sensor Failures 33

2.5.4 Maintenance of Voronoi Generators for Dynamic Sensor Networks 37

2.5.5 Routing to Voronoi Generators 40

2.6 Simulation Results 42

2.6.1 Static Sensor Networks 43

2.6.2 Dynamic Sensor Networks 46

2.7 Related Work 49

2.7.1 Coverage-Boundary 49

Page

2.7.2 Coverage-Preserving Redundancy Elimination 51
2.8 Conclusions 52
3 Reader Collision Avoidance in RFID Systems 53
3.1 The Reader Collision Problem 55
3.2 Reader Collision Avoidance (RCA) Algorithm 56
3.3 Analysis 60
3.4 Simulation and Results 62
3.5 Related Work 67
3.6 Conclusions 69
4 Redundant-Reader Elimination in Wireless RFID Systems 70
4.1 Network Model 72
4.2 The Redundant-Reader Problem 72
4.3 The Redundant Reader Elimination Algorithm 75
4.3.1 Extensions 77
4.4 Analysis 79
4.5 Simulation Results 79
4.6 Conclusions 83
5 Conclusions 84
LIST OF REFERENCES 87
VITA 91

LIST OF FIGURES

Figure Page

1.1 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. 2

1.2 Example of redundant sensors. Sensors S_1 and S_2 are both redundant. However, their simultaneous deactivation leaves a blind spot, shown in dashed curves. 3

1.3 Redundant reader example: readers R_1, R_2, R_3 and R_4 are redundant since the tags covered by each is covered by at least one other reader. This redundancy information would not be detected by a sensor redundancy detection algorithm, since none of the coverage areas the readers are subsumed by the others. 4

1.4 Example of reader collision problem. Concurrent operation of RFID readers R_1 and R_2 may prevent the correct identification of RFID tag T_1. The identifiers of tags are shown on the right side of the figure. 6

2.1 Voronoi diagram of the sensors in Fig. 1.1. The circles represent the coverage disks of the labeled sensors. Note that only for A and R their coverage area completely covers their Voronoi cell. In the next section we will show that this is not a coincidence. 11

2.2 Multiplicative weighted Voronoi diagram (MWVD) of 11 sensors. Each sensor is represented by a point, a weight and a light gray circle denoting the boundary of its sensing area. 13

2.3 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. . . 15

2.4 Proof of Lemma 3.3.1. 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. 16

Figure Page

2.5 (a) Example of a sensor network with blind points. Sensors $a, .., g$ are all redundant. However, if all of them are turned off simultaneously, the areas colored white are left uncovered. (b) Redundant graph of the network in (a) – the numbers associated with the nodes represent their degree in the redundant graph. The circled nodes represent winners in the first round, and the crossed nodes represent their direct neighbors, losers. Sensor 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. 18

2.6 Example for the proof of the lower bound for the coverage-boundary problem. We show the transformation from two sets $S_1 = \{8, 3\}$ and $S_2 = \{3, 12\}$ to an instance of the coverage-boundary problem in the Cartesian plane. The lowest horizontal line shows the coordinates on the x axis of the main circles. The main circles are represented using thicker arcs. Note that the circumcircle of the main circle corresponding to value 3 is completely covered by the secondary circles on L_1, L_2 and L_3, whereas the circumcircles of the main circles for 8 and 12 are not completely covered by the secondary circles. . . . 21

2.7 Proof of Theorem 3.4.2. (a) Example of the case where line $s_i x$ intersects $v_1 v_2$ inside s_i's coverage disk. (b) The intersection point is outside s_i's coverage disk. 23

2.8 (a) Example of GHT: nodes n_1 and n_2 look for the same key, whose associated object is located on node n_{f1}. Even though nodes n_1 and n_2 are connected, they find different home nodes and corresponding perimeter rings when looking for the same key. Node n_1 finds home node n_{f1} and home perimeter n_{f1}, p_1, n_1, whereas node n_2 finds home node n_{f2} with corresponding home perimeter n_{f2}, p_2, n_2. (b) The network in (a), augmented with a Voronoi overlay. Devices n_1 and n_2 both search the same key whose pos_{key} belongs to n_{f1}'s cell. Device n_1 clearly finds n_{f1}. Device n_{f2}, upon receiving the request from n_2, forwards it toward n_{f1}, through nodes $n_2, 1..4$, and n_1. This is because in n_1's local view node n_{f1} is the closest to pos_{key}. 25

2.9 Example of new sensor deployment: the gray circle, n_d, 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. 29

2.10 Illustration of the correctness proof for the join algorithm. 32

2.11 Example of network where sensors f_1, f_2 and f_3 fail simultaneously. The round arrows show the local decisions made by intermediate sensors, and the straight arrows show the trajectory of the DISC messages. 34

2.12 Example of host movements and associated topological events. 38

Figure Page

2.13 DMV: the dynamic and distributed algorithm for maintaining the Voronoi generators of individual sensors. 39

2.14 Proof of Theorem 3.6.1 – divide the area in $r/3 \times r/3$ bins. Each bin contains at least one sensor. The hashed bin contains sensor A. 41

2.15 Plots showing the evolution of the number of boundary sensors, redundant sensors and sensors that can be turned off. 42

2.16 Plots showing the total number of messages required for the deployment of new sensors. 44

2.17 DMV vs. flooding algorithm: evolution of 20, 30, and 40 sensors moving with a speed of 3 units/s in a square of size 200. 46

2.18 Performance of DMV: evolution of 30 sensors at speeds of 1, 5, and 10 units/s. 47

2.19 Counter-example to the use of a simple Gabriel-graph or RNG-graph based algorithms: points represent devices, large circles centered around points represent devices' coverage areas. The dotted circle represents A's coverage circumcircle. The direct links show device A's Gabriel and RNG neighbors respectively. 50

3.1 The generic reader and tag behavior. $getRandom(v_1, v_2)$ returns a random integer value between v_1 and v_2 and $bCast(packet)$ is used to broadcast $packet$. 58

3.2 Performance of RCA when the number of epochs per query grows from 1 to $2 \log\ \psi$, for a total of 500 readers and 4000 tags. The number of time frames per epoch is $2 \log\ \psi$. 62

3.3 The performance of our solution when the number of time frames per epoch increases from 1 to 38, for a total of 500 readers and 4000 tags. The number of time epochs per query is $\log\ \psi$. 63

3.4 Testing the scalability of our solution when the number of tags increases from 1000 to 8000, while the number of readers is 1000. 64

3.5 Testing the scalability of our solution when the number of readers increases from 10 to 500, for 4000 tags. 65

3.6 Testing the performance of our solution when the interrogation zone of the readers increases from 40m to 100m, for 500 readers and 4000 tags. We performed the experiments such that even when the interrogation zone is 40m the readers completely cover the tags. 67

Figure Page

4.1 Redundant reader example: readers R_1, R_2, R_3 and R_4 are redundant since the tags covered by each is covered by at least one other reader. This redundancy information would not be detected by a sensor redundancy detection algorithm, since the coverage areas of any of the readers are not subsumed by the others. The optimal solution requires only R_2 to be active, while the other readers may be turned off. 71

4.2 Set of points covered by a circle of radius R, shown with an interrupted perimeter. There is a circle of radius R going through points A and B and covering all the other points. Shrink this circle until it first touches one more point. The resulting circle, has radius less than or equal to R. 73

4.3 The generic RFID reader and writable tag behavior for detecting redundant readers. 76

4.4 Number of redundant readers discovered by RRE and Greedy when the number of tags randomly deployed increases from 1000 to 8000. The number of readers is constant, 500, throughout this experiment. 80

4.5 Difficulty of consistently breaking ties. 80

4.6 Number of redundant readers discovered by RRE and Greedy when the number of readers randomly deployed increases from 50 to 1000, for a total of 4000 tags. 81

4.7 Number of redundant readers discovered by RRE and Greedy when the interrogation radius of readers increases from 40 to 100m. The number of readers is 500 and the number of tags is 4000, for the entire duration of the experiment. 81

ABSTRACT

Cărbunar, Bogdan. Ph.D., Purdue University, May, 2005. Coverage Problems in Wireless Sensor and RFID Systems. Major Professors: Jan Vitek and Ananth Grama and Cristina Niţă-Rotaru.

The rapid self-configuration, ease of deployment and small cost of components, coupled with the tremendous potential in areas of environmental and structural monitoring, supply chain automation, identification of products at check-out points, access control and security, motivate the popularity of wireless sensor networks, the recent interest generated by wireless Radio Frequency Identification (RFID) systems and their envisioned integration. While the autonomous operation and random deployment of components are the principal causes of the low set up cost of these systems, they also become the source of fundamental problems. This thesis studies the problem of extending the network lifetime in the context of sensor and RFID systems by defining and detecting redundant components whose simultaneous deactivation maintains the initial network coverage. For wireless sensor networks, we reduce the problem to the computation of Voronoi diagrams. Moreover, we examine the impact of redundancy elimination on the related problem of coverage-boundary detection. We present efficient distributed algorithms for computing and maintaining solutions for the redundant sensor elimination problem and coverage boundary problem in cases of sensor failures or insertion of new sensors. We prove the safety and liveness properties of our algorithms, and characterize their time complexity and traffic generated. Using detailed simulations, we also quantify the impact of system parameters such as sensor density, transmission range and failure rates on network traffic.

In the context of wireless RFID systems, we provide an efficient solution to a fundamental problem generated by reader collisions occurring at tags. We prove that an optimal solution for the redundant-reader problem is NP-hard and propose a randomized approx-

imation algorithm. We conduct elaborate experiments on realistic topologies in order to evaluate the accuracy, message overhead and efficacy of the protocols. Our simulations show that by repeating each query $\log m$ times and using $2 \log m$ time units for each query, where m is the total number of RFID readers, each reader can discover more than 99% of the covered RFID tags. Moreover, even without the existence of a centralized entity, we discover consistently more than half of the redundant readers of a greedy algorithm using centralized information.

1. INTRODUCTION

Due to technological advances and the emergence of new application areas, radio networks have considerably evolved in the past decade. Instances of radio networks include wireless sensor networks, Radio Frequency Identification (RFID) systems, mobile ad-hoc networks and cellular networks. Wireless sensor networks and RFID systems owe their popularity to the small size coupled with the modest price of components, making them extremely appealing for random, mass deployment.

While wired sensor networks have been for years part of technological processes the recent need for monitoring of remote or inaccessible areas, along with the integration of sensed information into a variety of physical processes provide overarching motivations for random deployment of wireless sensor networks. Example of applications include ecological, agricultural and military monitoring.

RFID systems consist of uniquely identified RFID transponders (tags) and RFID transceivers (readers), capable of identifying RFID tags placed in their vicinity. Applications of RFID systems include supply chain management, cold chain management, identification of products at check-out points, access control and security. The miniaturization of RFID readers, coupled with wireless capabilities, expands the applicability of RFID systems. Similar to wireless sensor networks, such systems can be dynamically deployed instead of being statically installed.

Unlike sensor networks, wireless RFID systems have the ability to decouple sensing from communication. RFID tags interfaced with external sensors, such as temperature and shock sensors [1], allow wireless RFID systems to be extended with new sensing capabilities. Moreover, the existing compatibility between recent RFID readers (SkyeRead M1-Mini [2]) and MICA2DOT motes makes the integration of wireless sensor networks and wireless RFID systems possible. Such a hybrid wireless sensor and RFID infrastruc-

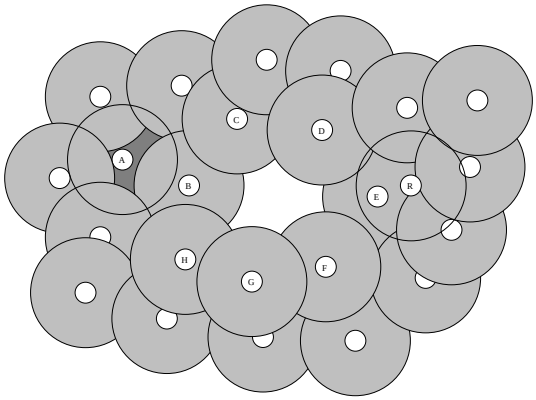


Fig. 1.1. Example of sensor network coverage – disks represent the coverage of the sensors situated at their center. Lightly shaded disks represent the coverage area of sensors situated on the boundary of the network. The dark areas belong to sensors that are not on the coverage boundary (e.g., sensor A). Sensor R is an example of a redundant sensor, since its coverage area is completely subsumed by other sensors.

ture can combine the ease of deployment with affordable identification and monitoring of objects.

The emphasis on low-cost, wireless sensing devices comes with significant constraints. Mica Motes have a lifetime of a few weeks when operating at full power, or years when operating at 2% duty cycle. SkyeRead M1-Mini [2] has an active life of a couple of weeks. Extending the lifetime of the sensor or reader network is a matter of paramount importance. At the same time, the relatively small sensing and interrogation zones of these devices requires dense deployment if reliable sensing of target areas and accurate detection of RFID tags is to be achieved .

Thesis Contributions The thesis is divided into two parts. The first part focuses on computing the coverage of wireless sensor networks, as it is an essential step in devising efficient battery-saving techniques. We look at two important coverage problems. The first problem consists of computing the boundary of the coverage of a sensor network. As illustrated in Fig. 1.1, we define the *coverage-boundary* of a network to include not only

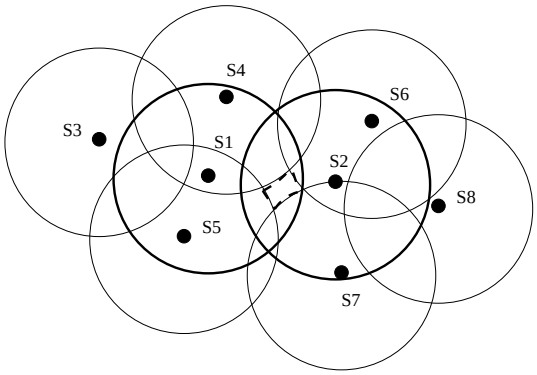


Fig. 1.2. Example of redundant sensors. Sensors S_1 and S_2 are both redundant. However, their simultaneous deactivation leaves a blind spot, shown in dashed curves.

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). More precisely, a sensor is on the coverage-boundary of the network, if the perimeter of its sensing disk is not entirely covered by other sensors in the network. Determining the coverage-boundary is important for identifying gaps in coverage and for optimizing sensor placement (by deploying new sensors).

The second problem consists of eliminating redundant sensors without affecting network coverage. This is called the coverage-preserving, energy-efficient redundancy elimination problem. We say that a sensor is redundant, if its sensing disk is *completely* covered by the sensing disks of other sensors in the network. Fig. 1.2 shows an example of redundant sensors, $\mathtt{S}_1$ and $\mathtt{S}_2$. Once all the redundant sensors are detected, we must choose a subset of redundant sensors, whose simultaneous deactivation will not leave holes in coverage (blind spots [3]). For example, in Fig. 1.2, the simultaneous deactivation of both redundant sensors will leave a blind spot. Note that the notion of coverage-boundary and redundancy are related, since the removal of a redundant sensor will not impact the coverage boundary of the network.

The difficulty of both problems stems from the inability of a centralized entity of gathering location information from all the sensors and notifying the sensors of the result. RF communications between sensors require much more energy than local computations and

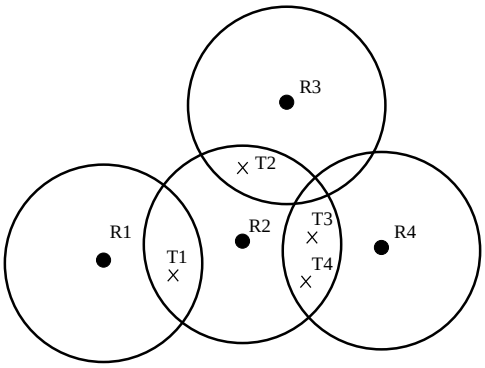


Fig. 1.3. Redundant reader example: readers R_1, R_2, R_3 and R_4 are redundant since the tags covered by each is covered by at least one other reader. This redundancy information would not be detected by a sensor redundancy detection algorithm, since none of the coverage areas the readers are subsumed by the others.

are the main cause of battery consumption. Thus, even if a spanning tree of the sensor network has already been built, the number of messages, ever increasing in size as they approach the root, make centralized approaches prohibitively expensive. Moreover, any solution has to be adaptive to topological changes, due to sensor failures and deployment of new sensors. A centralized approach would require that even a local change affecting only a few sensors be propagated towards the collection point.

A distributed algorithm can take advantage of the locality of the solution. A sensor can detect its presence on the coverage-boundary or its redundancy, using only location of sensors whose sensing disks intersect its own. However, in the worst case, such sensors can be farther apart than their transmission range, requiring intermediate sensors to relay this information.

For both the coverage-boundary detection problem and the coverage-preserving, energy-efficient redundancy elimination problem, we propose a solution based on the Voronoi diagram of the sensor network. We prove an equivalence relationship between the coverage of the perimeter of the sensing disk of a sensor by other sensor and the coverage of the sensor's Voronoi cell by its own sensing disk. We also prove an equivalence relationship

between the redundancy of a sensor and the coverage of the Voronoi structure generated by the Voronoi neighbors of the sensor. Thus, both solutions can be locally computed using only the geographic location of Voronoi neighbors of sensors . We provide a distributed solution, that adapts to failures and deployment of new sensors. We also propose a simple distributed algorithm for maintaining the local Voronoi information when sensors move. Moreover, we use the idea of locally maintaining the Voronoi cells of devices, to provide a simple distributed hash table implementation in general purpose ad-hoc networks.

The later part of the thesis studies related problems in RFID systems. An essential problem consists in the accurate detection of tags by readers, in the presence of interference. There are two main types of interference in RFID systems. The first type, tag collision, occurs at readers that receive simultaneous replies from multiple tags, preventing readers from correctly decoding the messages. Two solutions have been proposed to overcome this problem, one randomized, where tags are required to delay their answers to reader queries a random time interval and one based on a sequential traversal of the name space of tags (see [4] for more details). The second type of interference, reader collision, occurs when two RFID readers whose interrogation zones intersect at a tag, attempt to read the tag simultaneously. Fig. 1.4 shows an example of two RFID readers $\mathtt{R}_1$ and $\mathtt{R}_2$ simultaneously querying their corresponding tags. A query containing string "0" of $\mathtt{R}_1$ occurring concurrently with any query of $\mathtt{R}_2$ will interfere at $\mathtt{T}_1$, hiding its presence from $\mathtt{R}_1$. Note that simultaneous "1" queries of $\mathtt{R}_1$ and $\mathtt{R}_2$ will not interfere, allowing the correct reception of answers from $\mathtt{T}_2$ and $\mathtt{T}_3$.

Furthermore, we study the redundancy detection problem in the context of RFID systems. Unlike redundant sensors, whose circular sensing areas are completely subsumed by the coverage of other sensors, we define RFID readers to be redundant if their covered RFID tags are also covered by other RFID readers. Thus, redundant readers are defined in terms of discrete sets of points instead of contiguous areas. Fig. 1.3 shows an example where all RFID readers are redundant. Similar to the redundancy elimination problem presented above, detecting redundant readers is not enough, since the simple deactivation of all redundant readers may leave tags uncovered. Thus, an algorithm for finding the

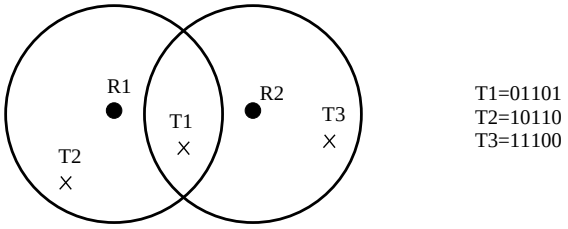


Fig. 1.4. Example of reader collision problem. Concurrent operation of RFID readers R_1 and R_2 may prevent the correct identification of RFID tag T_1. The identifiers of tags are shown on the right side of the figure.

maximum number of redundant readers that can be safely turned off simultaneously needs to be devised.

Both avoiding reader collisions and determining the subset of redundant readers that can be simultaneously deactivated, are complicated by the lack of collision detection mechanisms, the potential inability of RFID readers to relay packets generated by other readers and the severe resource constraints of RFID tags. In the traditional ad-hoc networks, collisions occurring at remote devices due to simultaneous wireless transmissions of devices that are not in each other's range, are known as the *hidden terminal problem*. The classic virtual carrier sensing solution cannot be easily adapted to RFID systems since Request-To-Send (RTS) messages have a single destination whose identity is known, whereas RFID readers have to avoid collisions at multiple tags and also to detect the identities of those tags. Moreover, existing solutions based on a central coordinator, allocating different time slots or frequencies to interfering RFID readers, considerably reduce the applicability and flexibility of the RFID system.

Since the main functionality of RFID readers is to detect RFID tags placed in their vicinity, they may not be able to relay packets issued by other readers and form an ad-hoc reader network. Thus, the solutions for both problems need to be not only distributed but also localized. RFID readers have to accurately detect tags, discover their redundancy and safely become inactive, using only information obtained from the tags placed in their vicinity.

For the reader collision avoidance problem we present a randomized, distributed and localized algorithm, RCA. In this algorithm readers repeat their queries at random intervals in order to avoid, with high probability, collisions with other simultaneously transmitting readers. For the redundant-reader problem, we prove that determining the smallest subset of RFID readers required to cover all the RFID tags is an NP-hard problem. We propose a distributed and localized approximation algorithm for this problem. This algorithm is targeted for writable RFID tags, that is, tags able to store information when issued a write command from RFID readers in their vicinity. Our algorithm, RRE, uses as input the output of RCA. Moreover, its subsequent steps borrow from RCA the idea of repeating commands at random intervals to avoid collisions with high probability.

Thesis Structure. Chapter 2 details the coverage-boundary and the coverage-preserving, energy-efficient redundancy elimination problems and presents our distributed solution. Chapter 3 looks at the reader collision problem and proposes a randomized, distributed and localized solution. Chapter 4 introduces the redundant-reader elimination problem and describes a randomized, distributed and localized approximation algorithm. Finally, Chapter 5 presents our conclusions.

2. COVERAGE AND REDUNDANCY DETECTION IN SENSOR NETWORKS

With the proliferation of wireless ad-hoc networks, increasing emphasis is being placed on algorithmic and software infrastructure for providing services to sensors. At the core of this algorithmic infrastructure lie a number of complex problems such as determining the *coverage*, *boundary*, and *triangulation* area, placement and location of services, etc. Two constraints that contribute to the complexity of these problems are: (i) devices must solve these problems in a distributed, efficient, and scalable manner; and (ii) solutions to these problems must be adaptive in nature, *i.e.*, after movement of sensors a the new solution may be rapidly recomputed Moreover, any proposed algorithm must work in concert with ad-hoc routing techniques. In this chapter we study two problems – the *coverage-boundary* problem and service placement and location in ad-hoc networks using *distributed hash tables*.

The coverage of a wireless sensor may be approximated by a disk of a prescribed radius, called the sensing range, which may be different from the transmission range of the sensor. We call this disk the coverage or sensing disk, and the boundary of the disk as the coverage or sensing circumcircle. The coverage of a network is the region covered by at least one sensor in the network. Coverage can be computed by taking the union of individual coverage areas of all sensors in the network. The coverage-boundary problem corresponds to the identification of all boundary nodes in a network. In this chapter we prove that sensors whose perimeter of the sensing disk is completely covered by the sensing disks of other sensors in the network, are not on the coverage-boundary of the network.

Several applications can benefit from the knowledge of the coverage-boundary. In a disaster recovery scenario, the coverage-boundary not only specifies the limits of the

network, but since the coverage of a network is ultimately determined by the location of boundary nodes, these nodes can be suitably re-located to optimize coverage. For sensor networks, usually deployed for monitoring purposes in applications ranging from agricultural crop monitoring to military surveillance, coverage-boundary information provides a measure for the quality of monitoring and can be used to determine the best position for newly deployed sensors.

In a services infrastructure built on an ad-hoc network, it is often necessary to make specific services and data available to boundary nodes. Such data may include maps and traffic information for emergency personnel, logistic information for command and control, and control vectors in distributed control. The time-criticality of most of these applications coupled with the potentially large number of nodes in the network requires a scalable and efficient solution to the coverage-boundary problem.

This chapter also addresses 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. 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.

The last problem studied in this chapter consists of the implementation of distributed hash tables in ad hoc networks. Distributed hash tables provide a simple solution to the problem of locating services. While these have been frequently used in peer-to-peer networks, they can also be efficiently applied to ad-hoc wireless networks. In CAN [5], Chord [6], Pastry [7], and Tapestry [8], a key is associated with every object. Each peer stores a certain range of keys (and possibly, references to actual locations of objects). In the context of a wireless ad-hoc network each object can be mapped to a geographic location and the node closest to this location is responsible for storing the key [9]. A node requesting an object generates a query that is hashed (routed) to the node closest to the corresponding geographic location. Supporting this routing primitive is the key to solving the problem.

Our solutions to the above problems are based on the distributed and adaptive construction of Voronoi diagrams. For the coverage-boundary problem we prove a tight relationship between the presence of a sensor on the boundary of the network and the coverage of its own Voronoi cell. For the coverage-preserving energy-efficient elimination redundancy elimination problem, we prove an equivalence relationship between a redundant sensor and the coverage of the Voronoi structure of generated by its Voronoi neighbors. For the distributed hash table problem we map services or objects to geographic locations and store the object on the sensor whose Voronoi cell contains that point.

Prior work [10, 11] on the distributed computation of Voronoi diagrams or its dual – Delaunay triangulation, offers mostly approximate solutions. More precisely, a sensor computes the Voronoi or Delaunay information only using sensors that are not farther than a given range. These approximate solutions are not adequate for selected applications, (see Section 2.7), since they can provide erroneous answers in the case where the range considered is smaller than twice the sensing range of sensors.

The challenge of computing Voronoi tessellations lies in developing distributed and adaptive formulations that are both stable, *i.e.*, that converge to the actual solution in a dynamic environment, and efficient in the context of ad-hoc networks. We assume, here, that device mobility can be bounded and in cases of fast moving devices it is possible to increase the frequency of updates. Moreover, solutions to these problems must be adaptive to sensor failures and deployment of new sensors. The new solution must be rapidly computed from the previous solution. Finally, since RF interfaces have a limited transmission range, protocols must account for overheads of multi-hop routing. Since communication consumes more energy than computation, the protocols themselves need to be energy efficient.

Chapter Organization In Section 2.2, 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 2.3, we present necessary and sufficient conditions for a sensor to be on the coverage boundary and show

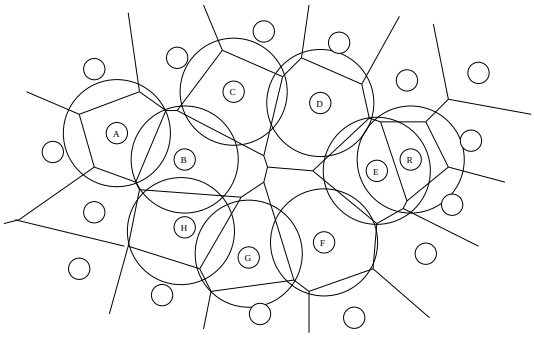


Fig. 2.1. Voronoi diagram of the sensors in Fig. 1.1. The circles represent the coverage disks of the labeled sensors. Note that only for A and R their coverage area completely covers their Voronoi cell. In the next section we will show that this is not a coincidence.

a lower bound of $\Omega(\mathtt{n}\log\mathtt{n})$ for any (serial) algorithm for the problem. We present a distributed algorithm for computing the coverage-boundary, whose serial counterpart has $\Omega(\mathtt{n}\log\mathtt{n})$ complexity. Both algorithms are based on the distributed and adaptive construction of Voronoi diagrams. In Section 2.4 we show an interesting application of Voronoi diagrams to the distributed hash table problem. In Section 2.5, 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 2.6, we experimentally characterize the performance of our algorithms. Conclusions are drawn in Section 2.8.

2.1 Overview of Voronoi Tessellations

Given a set $\mathtt{S}$ of $\mathtt{n}$ sites $\mathtt{s}_1, \mathtt{s}_2, .., \mathtt{s}_\mathtt{n}$ in a plane, their 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 Voronoi 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 meet along a *Voronoi edge*, and three Voronoi cells meet 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 use these two terms interchangeably. Fig. 2.1 illustrates the Voronoi diagram of the network in Fig. 1.1.

A 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 vertices of the same Delaunay triangle iff they are Voronoi neighbors. We formally define the Delaunay distance as follows:

Definition 2.1.1 *The Delaunay distance between two sites is the minimum number of hops between the two sites in the Delaunay triangulation.*

For example, in Fig. 2.1, sensors $\mathtt{A}$ and $\mathtt{R}$ are at Delaunay distance 4.

Multiplicative Weighted Voronoi Diagrams

An MWVD is defined in a manner similar to a Voronoi diagram, with the addition of weights at each of the $\mathtt{n}$ sites. In the definition of the classical Voronoi diagram, the sites have equal weights. The MWVD replaces the Euclidean distance used by the Voronoi diagram with a new distance $\mathtt{d_{mv}}$ defined by

$$\mathtt{d_{mv}}(\mathtt{s_i}, \mathtt{x}) = \frac{\mathtt{dist}(\mathtt{s_i}, \mathtt{x})}{\mathtt{w_i}} \tag{2.1}$$

In this definition, $\mathtt{s_i}$ corresponds to one of the $\mathtt{n}$ sites in the plane, $\mathtt{w_i}$ is a weight associated with it, $\mathtt{dist}$ is the Euclidean distance function, and $\mathtt{x}$ corresponds to any point in the plane. The MWVD cell of each site is formally defined as:

$$\mathtt{cell_{mwvd}}(\mathtt{s_i}) = \bigcap_{\mathtt{j}=1,\mathtt{j}\neq\mathtt{i}}^{\mathtt{n}} \{\mathtt{x} | \mathtt{d_{mv}}(\mathtt{s_i}, \mathtt{x}) \leq \mathtt{d_{mv}}(\mathtt{s_j}, \mathtt{x})\}$$

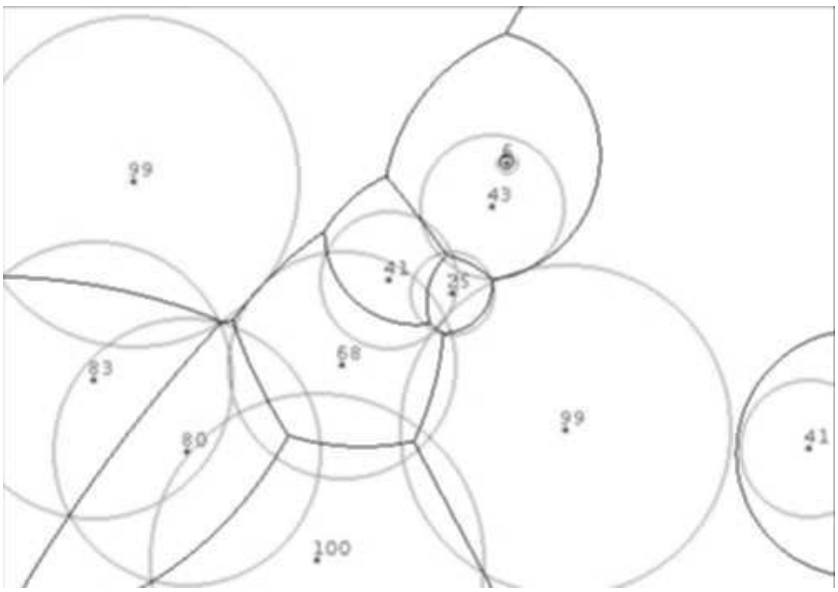


Fig. 2.2. Multiplicative weighted Voronoi diagram (MWVD) of 11 sensors. Each sensor is represented by a point, a weight and a light gray circle denoting the boundary of its sensing area.

This definition changes the bisectors of the Voronoi diagram into arcs (or circles). By making each sensor be a site and by replacing the weight $\mathtt{w_i}$ of a site with the sensing range, $\mathtt{r_i}$, of the corresponding sensor, equation 2.1 becomes: $\mathtt{d_{mv}(s_i, x) = dist(s_i, x)/r_i}$ Fig. 2.2 shows an example of a MWVD. The dark gray arcs form the MWVD cells. Note that the sensor with weight 25 is not on the boundary since its sensing circumcircle is completely covered by the sensing circles of neighboring sensors. Also, its sensing disk completely covers its MWVD cell. The sensor with weight 43 is on the boundary since its sensing circumcircle is not completely covered by the other sensing disks. Also, its MWVD cell is not completely covered by its sensing disk.

2.2 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

chapter, we assume that all the sensors in the network have the same sensing range. Extensions of these schemes to non-identical sensing ranges are possible using *multiplicative weighted Voronoi diagrams*.

We also make the assumption that each sensor knows its location. This is reasonable 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). Moreover, several algorithms have been proposed for locating devices in ad-hoc networks [12–14]. Examples of applications using location information include geographic routing [15–18] and management of location information [19, 20].

Definition 2.2.1 *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 is the union of the coverage disks of all the sensors in the network. Formally,

Definition 2.2.2 *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* $\mathtt{s}$.

The definition of a redundant sensor follows naturally:

Definition 2.2.3 *A sensor is said to be redundant if its sensing area is* completely *covered by other sensors.*

We define the 2-Voronoi diagram of a sensor in the following manner:

Definition 2.2.4 *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*

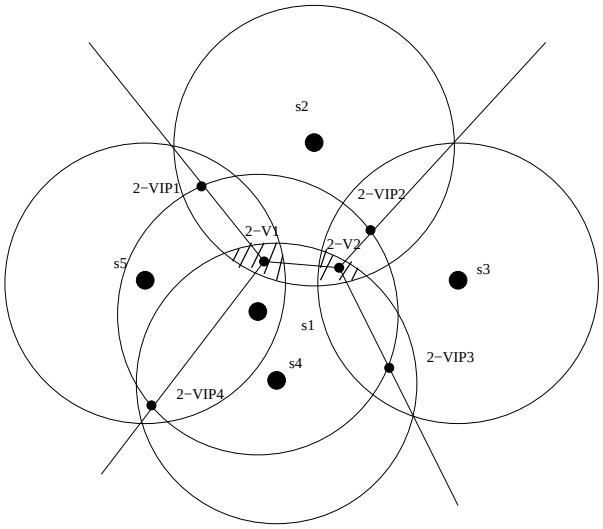


Fig. 2.3. Example redundant sensor, s_1. Points $2-V_1$ and $2-V_2$ are 2-VVs of s_1, and $2-VIP_{1..4}$ are 2-Voronoi Intersection Points of s_1. Note that $2-V_{1,2}$ and $2-VIP_{1..4}$ are all covered by at least two of the Voronoi neighbors of s_1.

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. 2.3 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 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 2.2.1 *A sensor* $\mathtt{s}$ *is redundant if and only if all the 2-VVs and 2-VIPs of* $\mathtt{s}$ *are covered by the Voronoi generators of* $\mathtt{s}$.

In order to prove the theorem, we prove first the following lemma.

Lemma 2.2.1 *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* $\mathtt{G}_\mathtt{s}$.

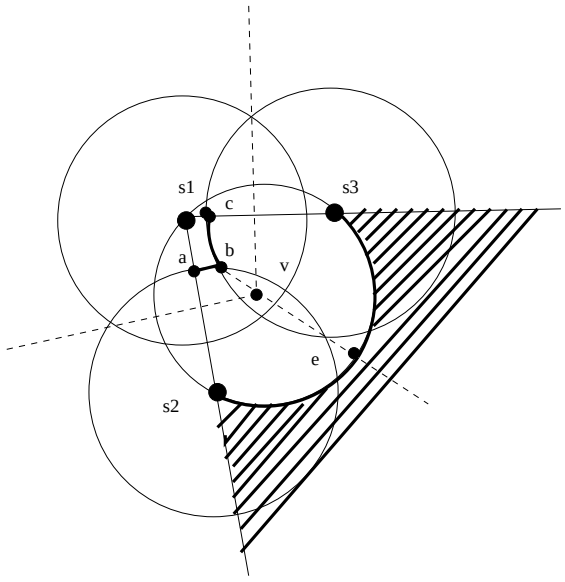


Fig. 2.4. Proof of Lemma 3.3.1. If sensors s_1, s_2 and s_3 are mutual Voronoi generators, another sensor can only be placed in the hashed area. Otherwise, that sensor would be a Voronoi generator of s_1.

Proof Let us consider the case in which sensor $\mathtt{s}_1$ has generators $\mathtt{s}_2$, and $\mathtt{s}_3$ whose coverage areas intersect each other and also the coverage area of $\mathtt{s}_1$ (Fig 2.4). We assume that all the sensors have the same coverage range, $\mathtt{r}$ (recall that this assumption can be relaxed with the use of MWVDs as opposed to Voronoi diagrams). Let $\mathtt{v}$ be the Voronoi vertex generated by the three sensors. Fig. 2.4 also shows the circumcircle of sensors $\mathtt{s}_1$, $\mathtt{s}_2$ and $\mathtt{s}_3$, centered at $\mathtt{v}$, containing no other sensor. Let $\mathtt{e}$ be the intersection point between this circle and the Voronoi edge generated by $\mathtt{s}_2$ and $\mathtt{s}_3$. The only area where another sensor, that is not a Voronoi generator of $\mathtt{s}_1$, can be placed, is the hashed area. Observe that $\mathtt{dist}(\mathtt{b},\mathtt{e}) = \mathtt{dist}(\mathtt{b},\mathtt{v}) + \mathtt{dist}(\mathtt{v},\mathtt{e})$. Therefore, due to triangle inequality, $\mathtt{dist}(\mathtt{b},\mathtt{e}) = \mathtt{dist}(\mathtt{b},\mathtt{v}) + \mathtt{dist}(\mathtt{s}_3,\mathtt{v}) > \mathtt{dist}(\mathtt{s}_3,\mathtt{b}) = \mathtt{r}$. Also, $\mathtt{dist}(\mathtt{s}_2,\mathtt{a}) = \mathtt{dist}(\mathtt{s}_2,\mathtt{b}) = \mathtt{r}$. It is easy to prove then that the distance between any point on the arc $\widehat{\mathtt{ab}}$ and a point on the arc $\widehat{\mathtt{s}_2\mathtt{e}}$ is greater than or equal to $\mathtt{r}$. The arcs are emphasized in Fig. 2.4. Similarly, the distance between any point on the arc $\widehat{\mathtt{bc}}$ and any point on the arc $\widehat{\mathtt{s}_3\mathtt{e}}$ is greater than or equal to $\mathtt{r}$. 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 intersect the coverage area of s_1 can be similarly proved. ■

We can now prove Theorem 2.2.1.

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. 2.3, which shows an example of a redundant sensor, s_1. Since the coverage area of s_1 is completely covered by other sensors, using Lemma 2.2.1, we infer that it is completely covered by the Voronoi generators of s_1. Furthermore, 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. 2.3 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. 2.3 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 (Section 2.1), 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. ■

2.2.1 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

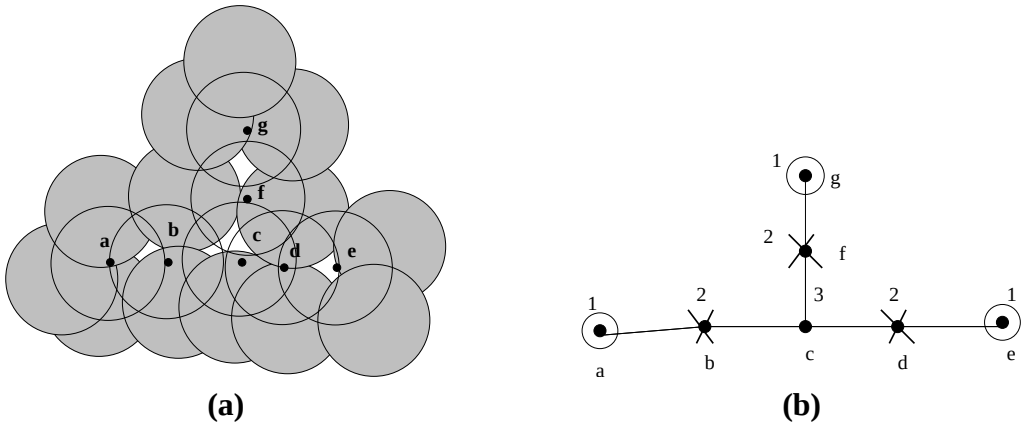


Fig. 2.5. (a) Example of a sensor network with blind points. Sensors $a, .., g$ are all redundant. However, if all of them are turned off simultaneously, the areas colored white are left uncovered. (b) Redundant graph of the network in (a) – the numbers associated with the nodes represent their degree in the redundant graph. The circled nodes represent winners in the first round, and the crossed nodes represent their direct neighbors, losers. Sensor 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.

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 each of its 2-VVs and 2-VIPs is covered by all the Voronoi neighbors that generated it. This verification takes constant time since the number of 2-VVs and 2-VIPs of a sensor is on the order of the number of Voronoi neighbors of the sensor.

2.2.2 Blind Points

If two redundant sensors that are also Voronoi neighbors, decide to turn off simultaneously, an area between them may be left uncovered. Such an area is called a blind point [3]. Fig. 2.5(a) shows an example of blind points created when *all* the redundant sensors simultaneously turn off. We need to find the maximum number of redundant sensors that can be turned off without generating blind points.

One solution to this problem, proposed by Tian and Georganas [3], uses a random back-off scheme. We propose an alternate solution, based on a slight modification of a distributed approximation of the maximal independent set (MIS) problem [21]. Let $G_R = (V_R, E_R)$ be the redundancy graph of the network, where V_R is the set of redundant sensors. There is an edge $e \in E_R$ 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 redundancy graph, G_R.

Our selection algorithm, similar to the one proposed by Luby [21], 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 [21] proves that a variant of this algorithm terminates and the expected number of rounds is $O(\log n)$.

Fig. 2.5(b) shows the redundancy graph of the sensor network from Fig. 2.5(a), and shows a trace of the selection algorithm. 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.

2.2.3 Management of Redundant Sensors

A winner in the above protocol can be safely turned off, since none of its redundant Voronoi neighbors are turned off. This is a special case of a sensor failure, and in Section 2.5.3 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, periodically wakes up in order to check the presence of its Voronoi generators. If one (or more) of them fail, 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 2.9.

2.3 Planar Coverage Boundary

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 2.2.3, 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.

Definition 2.3.1 *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. However, if two sensors have the same position, we do not consider that any point on the circumcircle of one of the sensors is covered by the disk of the other sensor.*

With these definitions in place, we formally describe the *coverage-boundary* problem:

The Coverage-Boundary Problem Given a set of $\mathtt{n}$ sensors in the plane, each with a sensing range $\mathtt{r}$, find all the sensors that are on the boundary of the coverage.

The following theorem establishes a lower bound for any solution of the coverage-boundary problem.

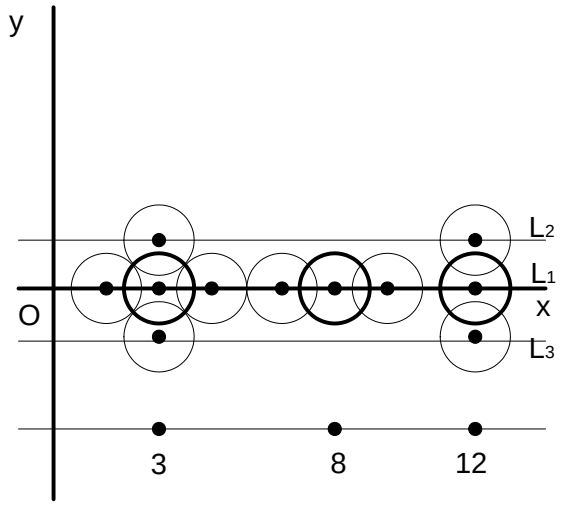


Fig. 2.6. Example for the proof of the lower bound for the coverage-boundary problem. We show the transformation from two sets $S_1 = \{8, 3\}$ and $S_2 = \{3, 12\}$ to an instance of the coverage-boundary problem in the Cartesian plane. The lowest horizontal line shows the coordinates on the x axis of the main circles. The main circles are represented using thicker arcs. Note that the circumcircle of the main circle corresponding to value 3 is completely covered by the secondary circles on L_1, L_2 and L_3, whereas the circumcircles of the main circles for 8 and 12 are not completely covered by the secondary circles.

Theorem 2.3.1 *The coverage-boundary problem has a* $\Omega(n \log n)$ *lower bound.*

Proof The proof is based on a linear-time transformation from the set equality problem to the coverage boundary problem. The set equality problem is stated as follows: Given two sets S_1 and S_2 of real numbers, both of size n, determine if the two sets are equal. The problem is known to have a $\Omega(n \log n)$ lower bound.

The transformation works in the following manner. Consider three horizontal lines in the Cartesian plane. The first line, L_1, is the x axis, L_2 and L_3 are lines parallel to L_1 going through the points $(0, \sqrt{2})$ and $(0, -\sqrt{2})$, respectively (Figure 2.6). For each element e_1 from the set S_1, insert three circles with their centers situated on L_1, in the following manner. The main circle has the center at $(e_1, 0)$ and radius 1, and the two secondary circles have centers at $(e_1 - \sqrt{2}, 0)$ and $(e_1 + \sqrt{2}, 0)$, respectively, both with a radius of

1. Similarly, for each element e_2 from the set S_2 add three circles in the plane. The main circle is again on L_1, centered at $(e_2, 0)$ and radius one. The secondary circles have their centers on L_2 and L_3, $(e_2, \sqrt{2})$ and $(e_2, -\sqrt{2})$, respectively, both with radius 1.

If two elements in the sets S_1 and S_2 are equal, the circumcircle of the main circles generated by the elements is completely covered by the disks of the secondary circles. Note that the two main circles generated by these elements do not cover each other, as stated in Definition 2.3.1. It is easy to see that the two sets S_1 and S_2 are equal if and only if the circumcircles of the main circles generated in the Cartesian plane are completely covered by the disks of the secondary circles. That is, S_1 and S_2 are equal iff the result of solving the coverage-boundary problem finds *all* the main circles as not being on the coverage-boundary. The transformation takes time $O(n)$ since for every element in the two sets, a constant number of circles (three) are added in the Cartesian plane. This proves that the coverage-boundary problem has a $\Omega(n \log n)$ lower bound. ■

Our solution for the coverage-boundary problem is based on the following theorem.

Theorem 2.3.2 *A sensor* s *is on the boundary of the network if and only if the Voronoi cell of* s *is not completely covered by its sensing range.*

Proof **If** sensor s_i is on the boundary, the coverage disk of s_i does not entirely cover the Voronoi cell of s_i. We only consider the case where s_i's Voronoi cell is bounded, since otherwise its cell is clearly not covered. Following Definition 2.3.1, let us take a point x on s_i's coverage circumcircle, such that x is not covered by the disk of any other sensor (Fig. 2.7(b)). Since s_i's cell is bounded, $\overline{d_i x}$ intersects one of the Voronoi edges of s_i's Voronoi cell. Let that Voronoi edge be $\overline{v_1 v_2}$, generated by s_i and s_j, and the intersection point be y. Point y cannot be inside s_i's coverage disk, since x would be covered by the coverage disk of sensor s_j, contradicting Definition 2.3.1 (Fig. 2.7(a)). Point y is then outside s_i's coverage disk. Since y belongs to s_i's Voronoi cell, there exists a point in s_i's Voronoi cell not covered by s_i's coverage disk.

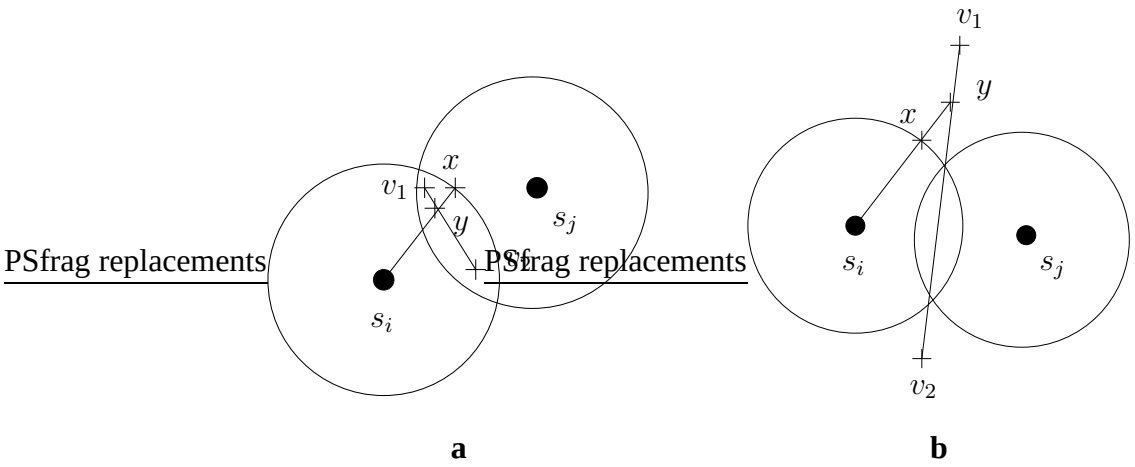


Fig. 2.7. Proof of Theorem 3.4.2. (a) Example of the case where line $s_i x$ intersects $v_1 v_2$ inside s_i's coverage disk. (b) The intersection point is outside s_i's coverage disk.

Only if the coverage disk of sensor $\mathtt{s_i}$ does not entirely cover its Voronoi cell, then $\mathtt{s_i}$ is on the boundary of the network. To prove this, let us consider a point $\mathtt{y}$ situated on a Voronoi edge belonging to $\mathtt{s_i}$'s cell, such that $\mathtt{y}$ is not covered by $\mathtt{s_i}$'s sensing disk (Fig. 2.7(b)). Let $\mathtt{x}$ be the intersection of $\overline{\mathtt{d_i y}}$ and $\mathcal{C}(\mathtt{s_i})$. Since the Voronoi cell of sensor $\mathtt{s_i}$ is convex, $\mathtt{x}$ is inside $\mathtt{s_i}$'s Voronoi cell. Then $\mathtt{r} = \mathtt{dist}(\mathtt{s_i}, \mathtt{x}) < \mathtt{dist}(\mathtt{s_j}, \mathtt{x}), \forall \mathtt{j} \neq \mathtt{i}$, hence $\mathtt{x}$ is not covered by any other sensor. According to Definition 2.3.1, $\mathtt{s_i}$ is on the boundary. ■

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 2.3.1, its circumcircle is also completely covered by other sensors. The following theorem describes another important property of redundant sensors.

Theorem 2.3.3 *The temporary inactivation of a redundant sensor will not switch the boundary state of a sensor from non coverage-boundary to coverage-boundary.*

Proof The sensors that are affected by the inactivation of a redundant sensor, $\mathtt{r}$, are those whose coverage area intersects the coverage area of $\mathtt{r}$. According to Definition 2.3.1, a

sensor is not on the coverage-boundary of the network if the circumcircle of its coverage area is completely covered by other sensors. Let us say that sensor $\mathtt{s}$ is affected by the inactivation of sensor $\mathtt{r}$, and $\mathtt{s}$ is not on the coverage-boundary of the network before $\mathtt{r}$ is turned off. The coverage circumcircle of $\mathtt{s}$ is then divided into two arcs: one that is covered by $\mathtt{r}$ and one that is not. The one that is not covered by $\mathtt{r}$ is clearly covered by other sensors. Since $\mathtt{r}$ is redundant and the coverage areas of sensors are circles (convex regions) the arc of $\mathtt{s}$'s coverage circumcircle that is covered by $\mathtt{r}$ is also covered by other sensors. ■

2.3.1 Coverage-Boundary for Heterogeneous Sensor Networks

Since wireless sensor networks may contain sensors with different sensing ranges, we investigate an extension of our results in such cases. To see why Theorem 2.3.2 still holds for the MWVD cells, consider the case in which the sensing range of sensor $\mathtt{s_i}$ does not entirely cover its MWVD cell, and the cell is bounded. If there is a Voronoi arc that is not covered, take a point $\mathtt{y}$ on the uncovered part of the arc. Let $\mathtt{x}$ be the intersection between $\overline{\mathtt{s_i y}}$ and $\mathcal{C}(\mathtt{s_i})$. Since $\mathtt{s_i}$'s Voronoi cell is convex, $\mathtt{x}$ is inside $\mathtt{s_i}$'s Voronoi cell. Then, by the definition of the Voronoi cell, $\mathtt{dist}(\mathtt{d_i}, \mathtt{x})/\mathtt{r_i} < \mathtt{dist}(\mathtt{d_j}, \mathtt{x})/\mathtt{r_j},\ \forall \mathtt{j} \neq \mathtt{i}$. $\mathtt{dist}(\mathtt{d_i}, \mathtt{x}) = \mathtt{r_i}$, hence $\mathtt{dist}(\mathtt{d_j}, \mathtt{x}) > \mathtt{r_j}$, and $\mathtt{x}$ is not covered by the disk of any other sensor.

2.3.2 Local Computation of Boundary Sensors

Based only on the position information of its Voronoi neighbors and of itself, any sensor can decide its presence on the network coverage-boundary in the following manner: generate the Voronoi diagram of a set of sites consisting only of itself and its Voronoi neighbors; this step takes constant time, since the expected number of Voronoi generators is constant. Then, check if its distance to each of the Voronoi vertices generated is less than the sensing range. This step takes constant time, since the number of Voronoi vertices generated is on the order of the number of generating sites.

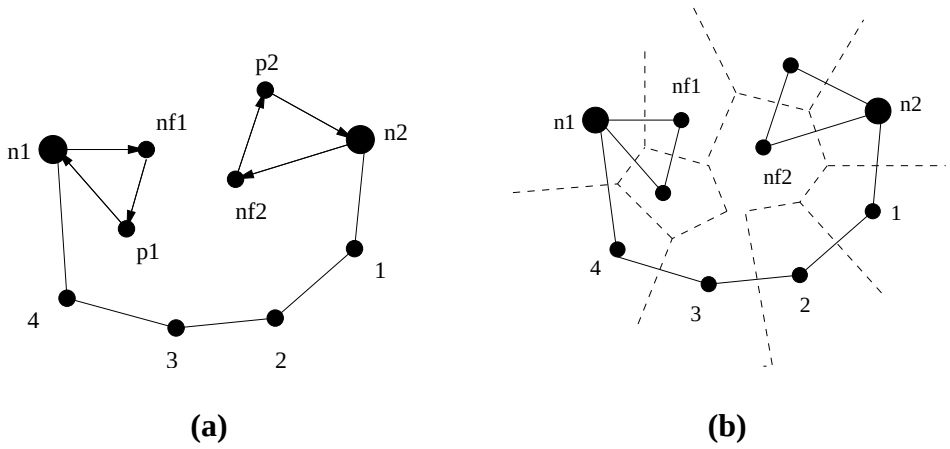


Fig. 2.8. (a) Example of GHT: nodes n_1 and n_2 look for the same key, whose associated object is located on node n_{f1}. Even though nodes n_1 and n_2 are connected, they find different home nodes and corresponding perimeter rings when looking for the same key. Node n_1 finds home node n_{f1} and home perimeter n_{f1}, p_1, n_1, whereas node n_2 finds home node n_{f2} with corresponding home perimeter n_{f2}, p_2, n_2. (b) The network in (a), augmented with a Voronoi overlay. Devices n_1 and n_2 both search the same key whose pos_{key} belongs to n_{f1}'s cell. Device n_1 clearly finds n_{f1}. Device n_{f2}, upon receiving the request from n_2, forwards it toward n_{f1}, through nodes $n_2, 1..4$, and n_1. This is because in n_1's local view node n_{f1} is the closest to pos_{key}.

2.4 Application of Voronoi Overlays to Distributed Hash Tables

In this section we address the use of Voronoi overlays to the problem of resource sharing in ad-hoc networks. The problem of distributed resource sharing, introduced originally in the context of peer-to-peer networks, stores shared resources (or references to these resources) in the network to facilitate fast access. An elegant adaptation of this problem to ad-hoc networks is provided in [9]. GHT [9] uses the notion of location to implement a distributed hash table on ad-hoc networks. Each object has an associated key that is mapped to geographic coordinates. We denote the corresponding coordinates as $\texttt{pos}_{\texttt{key}}$. The node that is closest to this location is the one that stores the object (or a reference to it). GHT uses GPSR [17] to find the closest node called the $home\ node$. Since GPSR

performs perimeter routing, when the message carrying the key reaches the home node, it enters perimeter mode. This happens because no other node is closer to the key. The packet traverses the perimeter around the home node, called the $home\ perimeter$, and replicates the object on all the nodes on the perimeter. One of the drawbacks of this technique is that two nodes trying to retrieve an object, given the same key, may find two different home nodes and associated home perimeters, due to restricted connectivity. This is illustrated in Fig. 2.8(a).

We provide an alternate solution for the distributed hash table problem, based also on mapping keys to geographic locations, but using the Voronoi overlay of the network's nodes. Specifically, we place an key/object binding on the node whose Voronoi cell contains the position $\texttt{pos}_{\texttt{key}}$ corresponding to the key. We can uniquely map each key/object binding to a single node, since every node has a unique Voronoi cell associated with it, and every point inside the cell of a node has the property of being physically closer to that node than to any other node in the network.

Whenever a node needs to install or retrieve a key/object binding, it converts the key into a geographic position $\texttt{pos}_{\texttt{key}}$. If $\texttt{pos}_{\texttt{key}}$ is inside its Voronoi cell, the node stores the key/object binding. If $\texttt{pos}_{\texttt{key}}$ is outside its Voronoi cell, the key/object binding must be appropriately routed to its destination node. Several forwarding methods can be used, the simplest being to greedily forward the request to the Voronoi generator that is closest to $\texttt{pos}_{\texttt{key}}$. Another method is to forward the request to the Voronoi generator whose cell is intersected by the line segment having the initiator of the request and $\texttt{pos}_{\texttt{key}}$ as endpoints. Bose and Morin [22] have proved that both methods reach the intended destination. Fig. 2.8(b) illustrates an example of our approach.

Discussion. Note that the coverage-boundary nodes have larger Voronoi cells than the other nodes, a direct consequence of Theorem 2.3.2. A uniform distribution of the keys leads to load imbalance, with nodes on the coverage-boundary storing more information than internal nodes. A simple solution to this problem is to use Multiplicative Weighted Voronoi Diagrams. A coverage-boundary node will participate with a smaller weight, effectively generating a smaller Voronoi cell. We can extend this idea to provide better load

balancing, by making the weight of each node a function of its resources, i.e. available storage capacity and battery power. A resource rich node will have a larger Voronoi cell (Figure 2.2), and will store more information.

2.5 Distributed Computation and Maintenance of Voronoi Cells

The resource and scalability constraints of a sensor network make the existence of a centralized entity, that would compute the global Voronoi overlay, 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 2.3.2, or its redundancy, according to Theorem 2.2.1.

2.5.1 Initial Distributed Computation of Voronoi Generators

Initially, each sensor knows only its own location, and stores it in a local repository. Each step of the algorithm requires every sensor to send a message containing the information kept in its repository to all its network neighbors, i.e., sensors that lie within its transmission range. Upon receiving this message from a neighbor, a sensor adds the information received about new sensors to its local repository. It then recomputes its Voronoi cell with the information contained in the repository. At the end of every step, each sensor checks to see if the updates received brought it new information. The algorithm continues for a sensor until no new information is received from all its network neighbors. Upon termination, each sensor discards all the information from the local repository, with the exception of its Voronoi generators and its network neighborhood information.

After k steps, where k is the diameter of the network, every sensor learns of every other sensor in the network. Therefore the algorithm terminates in $O(k)$ time. At each step, every sensor broadcasts exactly one message, hence the total number of messages is $O(kn)$, where n is the number of sensors in the network. The number of steps in the distributed

algorithm is optimal. This is because two Voronoi neighbors may be separated in the network neighbor graph by k links. The construction for this is simply a set of sensors organized in a ring in which each sensor only sees two other sensors in the network. In this case, the diameter k is $n/2$ and two sensors that are Voronoi neighbors may be $n/2$ hops away.

2.5.2 Deployment of New Sensors

The deployment of new sensors has the potential to change the set of redundant sensors and the coverage-boundary of the network. In both cases, this problem can be reduced to updating the local Voronoi information of the affected sensors. Fig. 2.9 illustrates an example in which a new sensor ns joins an existing network. Only sensors whose Voronoi cell is affected by the presence of the new sensor have to be notified about the new sensor. It is well known that the circumcircle of a Delaunay triangle contains no other 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. 2.9 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 [23].

Lemma 2.5.1 *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 2.5.2 *The expected number of Voronoi neighbors of a sensor in a randomly deployed sensor network is constant.*

Join Algorithm. Whenever a new sensor is deployed, it sends a beacon message and a sensor that receives this beacon is said to *detect* the new sensor. Thus, the new sensor can be detected by another sensor only if the radio transmission ranges of the two sensors exceeds the distance between them. A sensor that detects the presence of a new sensor needs

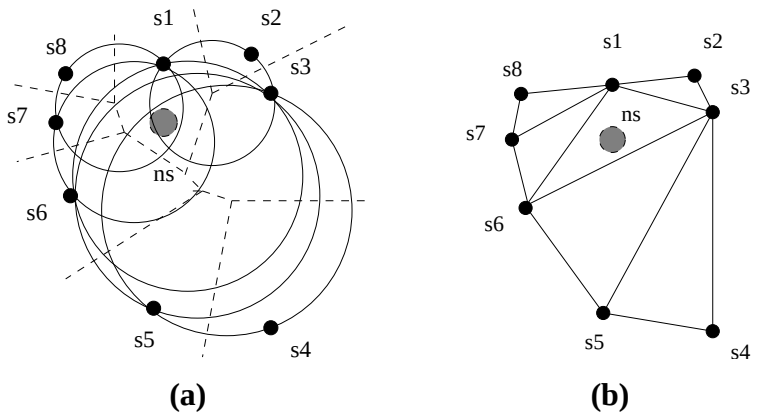


Fig. 2.9. Example of new sensor deployment: the gray circle, n_d, represents the new sensor. (a) The Voronoi diagram of the old sensors. The circles denote circumcircles of Delaunay triangles. (b) The Delaunay triangulation of the old sensors.

to notify other sensors affected by its presence. Since more than one sensor can detect the new sensor, multiple, redundant notifications might be sent in the network. To avoid this, we choose only one sensor, called *introducer* (S), to perform the notification. We choose S to be the sensor whose Voronoi cell contains the position of the newly deployed sensor. In Fig. 2.9(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 (Section 2.1), our choice guarantees that the introducer will detect the new sensor.

The introducer initiates the notification process, and during the process, each notified sensor uses only local information to determine its behavior. Each sensor has a list of incident Delaunay triangles, and the introducer, S, traverses its list of Delaunay triangles, identifies the ones in conflict with the new sensor, and isolates the notifiables among its Voronoi neighbors. Then, sensor S sorts its 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, will only contact

the black ones. A gray colored notifiable denotes a sensor that can be easier notified by one of the black notifiables, thus its notification should be postponed.

The coloring algorithm proceeds as follows. First, S 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, G, such that $\mathtt{dist}(\mathtt{S},\mathtt{W}) > \mathtt{dist}(\mathtt{G},\mathtt{W})$, it marks W gray and removes it from the white list. Since W is closer to G than to S, it can be easier notified by G than by S. 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 example from Fig. 2.9, sensor $\mathtt{s}_1$ first colors sensor $\mathtt{s}_2$ black and in the subsequent traversal colors sensor $\mathtt{s}_3$ gray. It then colors sensor $\mathtt{s}_8$ black, and in the first traversal colors sensor $\mathtt{s}_7$ gray. A second traversal colors sensor $\mathtt{s}_6$ gray.

On completion of the coloring phase, S sends a notification to each black notifiable. For such a notifiable B, the message contains the identity and position of the new sensor, the identity of the notifier, in this case S, and the identities of all the gray notifiables in the local view of S, that B needs to notify. A sensor N that receives a new sensor notification, ignores the notification if it has already received it from another Voronoi neighbor. Otherwise, N creates its own white list of notifiable Voronoi neighbors. N 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 notifier (the sensor from whom it has received this notification) and marks it gray, since it should not notify it. N 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.

Example. Figure 2.9 illustrates an example of the notification process. Here, ns is a newly deployed sensor. As explained above, sensor $\mathtt{s}_1$ sends a message only to sensors $\mathtt{s}_2$ and $\mathtt{s}_8$. When sensor $\mathtt{s}_3$ receives the notification from sensor $\mathtt{s}_2$, it first constructs its list of notifiables, consisting of sensors $\mathtt{s}_2$, $\mathtt{s}_4$, $\mathtt{s}_5$, $\mathtt{s}_6$, and $\mathtt{s}_1$. It then colors sensor $\mathtt{s}_2$ 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 has 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 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, which in turn will notify sensor s_6, and so on. 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.

Whenever a new sensor joins the network, not only the sensors affected by its presence need to be notified, but also the new sensor has to be notified about the presence of the sensors that form its Voronoi neighborhood. This task is performed by the sensors affected by the new sensor, since they know its identity and location.

We now present several properties of the join algorithm.

Correctness. 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. 2.10 illustrates the proof, where $\mathtt{ns}$ represents the new sensor. The basis 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 $\mathtt{l}-1$ are notified. Let s_1 be a notifiable at Delaunay distance $\mathtt{l}$ from the introducer. Being a notifiable, s_1 has at least a triangle

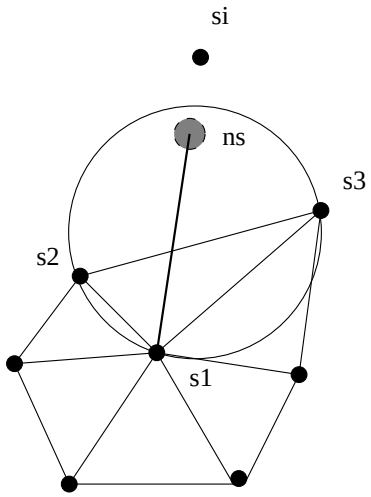


Fig. 2.10. Illustration of the correctness proof for the join algorithm.

in conflict with the new sensor, $\mathtt{ns}$. Since in terms of the Euclidean distance $\mathtt{s}_2$ and $\mathtt{s}_3$ are $\mathtt{s}_1$'s closest Voronoi neighbors to $\mathtt{ns}$, the triangle $\Delta\mathtt{s}_1\mathtt{s}_2\mathtt{s}_3$ is such a conflict triangle. Sensor $\mathtt{s}_1$ is at Delaunay distance $\mathtt{l}$ from the introducer, $\mathtt{s}_\mathtt{i}$, therefore it has at least one Voronoi neighbor at Delaunay distance $\mathtt{l}-1$ from $\mathtt{s}_\mathtt{i}$. Since $\mathtt{s}_\mathtt{i}$ is the closest sensor to $\mathtt{ns}$, no other Voronoi neighbor of $\mathtt{s}_1$ is at a smaller Delaunay distance from $\mathtt{s}_\mathtt{i}$ than $\mathtt{s}_2$ and $\mathtt{s}_3$. Hence, at least one of $\mathtt{s}_2$ or $\mathtt{s}_3$ are at Delaunay distance $\mathtt{l}-1$ from $\mathtt{s}_\mathtt{i}$. If both are, since both are notifiable, they will both be notified. At least the one closer to $\mathtt{s}_1$ will then notify it, since both will detect the triangle $\Delta\mathtt{s}_1\mathtt{s}_2\mathtt{s}_3$ to be in conflict with $\mathtt{ns}$. If only one of $\mathtt{s}_2$ or $\mathtt{s}_3$ is at distance $\mathtt{l}-1$, then that one will notify $\mathtt{s}_1$. ■

Termination. 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 2.5.1), and each propagates a notification only once, the algorithm will terminate. ■

Complexity. 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 $O(\log n)$ (Lemma 2.5.1), 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 (Lemma 2.5.2), the expected number of notifications is $O(\log n)$. ■

2.5.3 Sensor Failures

Sensor failures, like the 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 2.5.3 *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 that is not a Voronoi neighbor of f_2 (Fig. 2.11). Since f_2 is not a Voronoi neighbor of sensor s_6, by the definition of Voronoi diagrams, there are no sensors in the interior of any of the Delaunay circles generated by sensor s_6. Hence sensor 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 f_3 must only

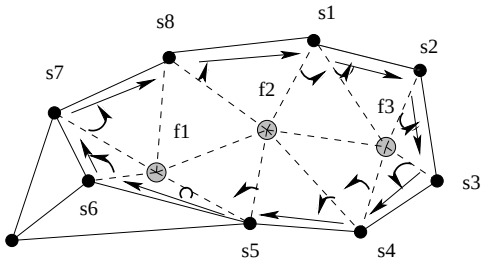


Fig. 2.11. Example of network where sensors f_1, f_2 and f_3 fail simultaneously. The round arrows show the local decisions made by intermediate sensors, and the straight arrows show the trajectory of the DISC messages.

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. ■

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. 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. Sensor 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. Sensor N then looks at the last item in the failed list, say G. Sensor N finds its first counterclockwise Voronoi generator starting from G, say 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 Delaunay triangulation using its local information and the information in the affected list received with the DISC message.

Example. Fig. 2.11 shows an example of a network in which three sensors, f_1, f_2, f_3, fail simultaneously. Let us say that s_7 detects that f_1 has failed. s_7 creates a DISC message containing f_1 in the failed list, and sends it to s_8. Before sending to f_2, s_8 detects that f_2 has also failed, and adds f_2 to the failed list. s_8 then forwards the DISC message to s_1. Similarly, s_1 adds f_3 to the failed list and forwards the message to s_2. When s_4 receives this DISC message, it detects that f_2 and f_3 are already in the failed list, so it forwards the message only to s_5. All the intermediate sensors add their identities and positions to the affected list, so when s_7 receives the DISC message that it has initiated, it is able to correctly recompute its Voronoi generators and incident Delaunay triangles.

Note. 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 on the order of 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 $\mathtt{G_F} = (\mathtt{V_F}, \mathtt{E_F})$ be the failure graph, where $\mathtt{V_F}$ is the set of failed sensors, and $\mathtt{e} \in \mathtt{E_F}$ is an edge between two sensors $\mathtt{f_1}$ and $\mathtt{f_2}$ in $\mathtt{V_F}$ if $\mathtt{f_1}$ and $\mathtt{f_2}$ are Voronoi neighbors. For every connected component $\mathtt{C}$ in $\mathtt{G_F}$, let $\mathtt{A_C}$ be the set of Voronoi neighbors of the sensors in $\mathtt{C}$, that are not themselves in $\mathtt{C}$. That is, $\mathtt{A_C}$ is the set of sensors affected by the failure of the sensors in the connected component $\mathtt{C}$. In Fig 2.11, $\mathtt{f_1}$, $\mathtt{f_2}$ and $\mathtt{f_3}$ form a connected component $\mathtt{C}$ of the failure graph, and $\mathtt{A_C} = \{\mathtt{s_1}, .., \mathtt{s_8}\}$. With these definitions, a generalization of Lemma 2.5.3 can be easily proved.

Lemma 2.5.4 *A connected component of the failure graph,* $\mathtt{C}$*, affects only the Voronoi diagram of its Voronoi neighbors,* $\mathtt{A_C}$*. Moreover, the set of candidates for the new Voronoi generators for each affected sensor in* $\mathtt{A_C}$ *is contained in* $\mathtt{A_C}$*.*

Then, the following properties can be proved.

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

Proof A sensor $\mathtt{S}$ that is affected by the failure of a sensor $\mathtt{F}$, part of a connected component $\mathtt{C}$ of the failure graph $\mathtt{G_F}$, will send a $\mathtt{DISC}$ message that will traverse all the sensors in $\mathtt{A_C}$. The message will collect information about all the traversed sensors. Using Lemma 2.5.4, we conclude that when $\mathtt{S}$ receives the $\mathtt{DISC}$ message that it initiated, it will have all the information necessary for computing the new Voronoi neighbors. ■

Termination. The departure algorithm terminates.

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

Complexity. 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 2.5.2. ■

2.5.4 Maintenance of Voronoi Generators for Dynamic Sensor Networks

While our main assumption, of sensor networks being static, is valid for most scenarios, certain cases arise where sensors are placed on moving objects. For example, in military applications, sensors can be associated with soldiers or vehicles. The main strength of such scenarios, namely the mobility of sensors, translates into the necessity of correctly updating the information stored by each sensor. In addition to old links breaking and new ones forming, the topology and shapes of the Voronoi cells change as sensors move. Fig. 2.12 illustrates a simple configuration of four sensors, their Voronoi diagram, and the circumcircle $\mathcal{C}(O_1)$ of sensors 0, 1, and 2, and $\mathcal{C}(O_2)$ of sensors 1, 2, and 3, respectively. As shown by Albers, Guibas, Mitchell and Roos [24], in this case, if sensor 0 moves, a topological event occurs when sensor 0 crosses circle $\mathcal{C}(O_2)$. Similarly, if sensor 3 moves, an event occurs when sensor 3 crosses $\mathcal{C}(O_1)$. Fig. 2.12(b) shows the result of sensors 0 and 3 moving simultaneously, as indicated by the arrows in Fig. 2.12(a). Even though none of the individual movements generates a global event, their orchestrated movement does, as sensors 0 and 3 become weak Voronoi neighbors. Two sensors are called weak Voronoi neighbors if their Voronoi cells share a Voronoi vertex, and not a Voronoi edge. This example illustrates that an independent movement of sensors can change the Voronoi topology, and information stored locally is not enough to detect these changes. This is true because a sensor is only aware of its new position, but not of the new position of other sensors. It also shows that the threshold above which a sensor must move in order to advertise its new position does not depend on changes in its local topology. The sensor *must* contact its generators every time it moves.

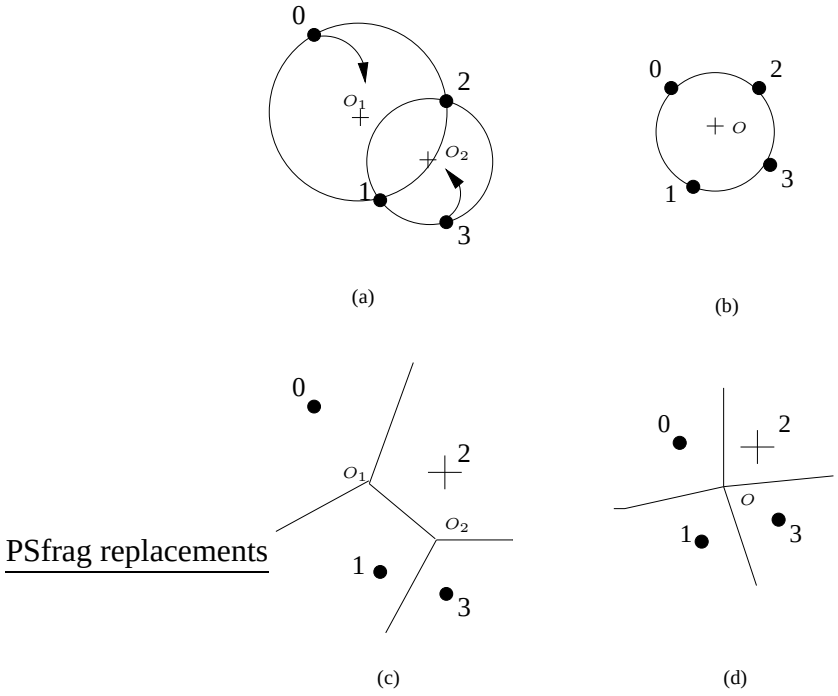


Fig. 2.12. Example of host movements and associated topological events.

We introduce an algorithm – **DMV** (Dynamic Maintenance of Voronoi), which maintains, in a distributed and efficient manner, the Voronoi cells of sensors as they move within specified, and arguably reasonable, bounds (Fig. 2.13). We remind that each sensor stores only information concerning its Voronoi neighbors. Any sensor that moves must send its new position to all of its old generators (its generators before the move) since their cells are directly affected by this move. This is performed in the `moveTo` method in Fig. 2.13, which is invoked every time a sensor moves. Since a sensor may move continuously and `moveTo` can be invoked only at discrete times, we choose to use a timer and call `moveTo` every time the timer expires. The arguments of `moveTo` represent the new x and y coordinates. New sensor information along with information regarding the old generators is created and sent to the old generators using the method `sendToNode`.

When a sensor receives an update concerning a generator, it recomputes the Voronoi diagram using its local data augmented with the information received. A sensor `s` needs to take further action upon detecting a change in the structure of its Voronoi neighbors.

```
1. Object implementation Node;
2.   myX, myY : integer;  #node position
3.   old_state, state, generators, updates : array[integer]ofNode;
4.   Operation moveTo(x : integer, y : integer)
5.     myX := x; myY := y;
6.     setOldGens();
7.     for i in 1..size(generators) do
8.       Nodegen := generators[i];
9.       sendToNode(CHANGE, gen, self);
10.    od
11.  end
12.
13.  Operation run()
14.    guard inQ.first.type = CHANGE do
15.      node = inQ.first.node;
16.      old_state := state;
17.      update(state, node);
18.      generators := Voronoi.build(state);
19.      remUselessInfo(state, generators);
21.      updates := diff(old_state, state);
21.      for i in 0..size(updates) do
22.        Updateup = updates[i];
23.        sendToNode(up.getV1(), up.getV2());
24.        sendToNode(up.getV2(), up.getV1());
25.      od
26.    od
27.  end
```

Fig. 2.13. DMV: the dynamic and distributed algorithm for maintaining the Voronoi generators of individual sensors.

That is, s performs an action concerning two sensors v_1 and v_2 if the following conditions are met: (i) v_1 and v_2 are both s's generators *before and after* the update, (ii) v_1 and v_2 *are not* Voronoi neighbors in the Voronoi diagram of s before the update, and (iii) v_1 and v_2 *are* Voronoi neighbors in the Voronoi diagram of s after processing the update. The action taken by s is then to propagate an update to v_1, notifying it of the possibility of v_2 becoming its Voronoi neighbor, and a similar update to v_2, notifying it about v_1. Fig. 2.12(c) and (d) shows the Voronoi cell of sensor 2 in the local view of sensor 2, before and after the movement. As can be seen in sensor 2's view, sensors 1 and 3 become

each other's generators, therefore sensor 2 notifies sensor 1 of the possibility of sensor 3 becoming its Voronoi neighbor. Similarly sensor 2 notifies sensor 3 about sensor 1.

In Algorithm 2.13 this step is performed in the method `processInfo` executed on behalf of the notified sensor. The method takes the identifier of the advertised service as an argument. In lines 16-18 the old generators of the receiver are saved, the local knowledge is updated with the received information, and the new Voronoi generators are recomputed. Since some of the old generators of the receiver might no longer be among the new ones, because of the information received, in line 19 we discard them using the method `remUselessInfo`. Line 20 generates all the differences between the structure of the old and new Voronoi generators, and in lines 21-25, all the detected changes are forwarded to the corresponding generators.

In order to reduce the number of update messages, we use the following optimizations. As can be seen in Line 6, a sensor packs information about its old generators along with its new position, before notifying its Voronoi neighbors of its movement. In this manner, if one sensor detects in its local view that a sensor v_1 is a new generator of sensor v_2, but v_2 is already in the list of old generators of v_1, then there is no need to notify v_1 of v_2, because v_1 already has this information. Moreover, every time a sensor receives an update, it recomputes its Voronoi generators and discards the old ones that are now isolated (Line 19), thereby sending updates only to sensors that are among its new generators. In addition, a sensor will not propagate an update about itself and one of its generators, since that generator can symmetrically detect this change.

2.5.5 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 transmission range, requiring a routing protocol. LAR [16], DREAM [18], and GPSR [17] are exam-

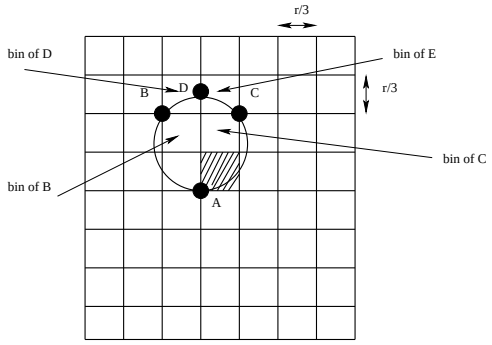


Fig. 2.14. Proof of Theorem 3.6.1 – divide the area in $r/3 \times r/3$ bins. Each bin contains at least one sensor. The hashed bin contains sensor A.

ples of location based routing protocols that can be used for routing between non-adjacent Voronoi neighbors. However, in the following theorem, based on [25], we provide a lower bound on the radio transmission range of sensors, that ensures that sensors that are Voronoi neighbors are likely to be within each other's transmission range.

Theorem 2.5.1 *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 [25], the probability that a bin is empty after throwing $\mathtt{n}$ balls(sensors) in the initial square is $(1 - \mathtt{r}^2/9\mathtt{S})^{\mathtt{n}} \leq \mathtt{e}^{-\mathtt{n}\mathtt{r}^2/9\mathtt{S}} \leq \mathtt{e}^{-\mathtt{c}\log \mathtt{n}/9} = \mathtt{n}^{-\mathtt{c}/9}$. The expected number of empty bins is then $\frac{9\mathtt{S}}{\mathtt{r}^2}\mathtt{n}^{-\mathtt{c}\mathtt{S}^2/9} = \frac{9}{\mathtt{c}\log \mathtt{n}}\mathtt{n}^{1-\mathtt{c}/9} \leq \mathtt{n}^{1-\mathtt{c}/9}$. This tends to 0 if $\mathtt{c} \geq 9$.

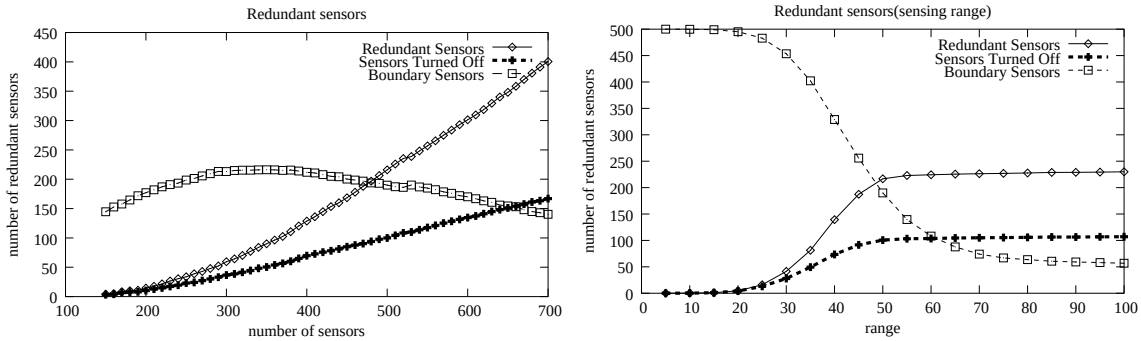
If every bin contains at least one sensor, it is easy to see that each sensor will be in the transmission range of all the sensors in the adjacent bins (the largest distance between two sensors in adjacent bins is $\mathtt{r}\sqrt{5}/3 \leq \mathtt{r}$) . To see how far two Voronoi neighbors can be, Fig. 2.14 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 $\mathtt{D}$ and $\mathtt{E}$. 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 $5r/6 < r$. 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. ■

Given the transmission range of the sensors, this theorem can be easily used to find the number of sensors that need to be randomly deployed in a given square area, in order to provide, with high probability, direct connectivity between Voronoi neighbors. For example, if 2000 sensors with a transmission range of 25m are randomly deployed in a square of size $100 \times 100m^2$, it is highly probable that any two Voronoi neighbors can communicate directly. This implies a sensor per $4m^2$.

2.6 Simulation Results

In this section, we support our analytical results with a detailed simulation study for quantifying the overhead and correctness of our algorithms. Section 2.6.1 presents the experimental results for static sensor networks and Section 2.6.2 evaluates DMV in the case of dynamic sensor networks.



(a) **(b)**

Fig. 2.15. Plots showing the evolution of the number of boundary sensors, redundant sensors and sensors that can be turned off.

2.6.1 Static Sensor Networks

All the experiments for static networks are performed by randomly placing identical sensors in a region of size $1000 \times 1000\text{m}^2$. In Section 2.6.1 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 2.6.1 we measure the communication traffic generated by join and leave operations.

Detecting 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.

Each time a new sensor $\texttt{ns}$ joins the network, new paths need to be generated between the new sensor and its new Voronoi generators. If the new sensor is not in the transmission range of one of its Voronoi generators, $\texttt{s}_1$, the path from $\texttt{ns}$ to $\texttt{s}_1$ is chosen to be the concatenation of the path between $\texttt{ns}$ and the sensor closest to it, with the path between its closest neighbor and $\texttt{s}_1$, since these paths already exist. Even though this is not necessarily the shortest path between $\texttt{ns}$ and $\texttt{s}_1$, our measurements showed that the communication overhead introduced by using this path is very small. Our experiments measure not only the number of messages required to notify all the sensors affected by the new sensor, but also the number of messages necessary for the affected sensors to contact the new sensor.

Fig. 2.15(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 boundary sensors out of a total of 700 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 [21] algorithm. However, the algorithm is scalable, since the number of possible simultaneous turn-offs grows linearly with the number of sensors.

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. 2.15(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.

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 informa-

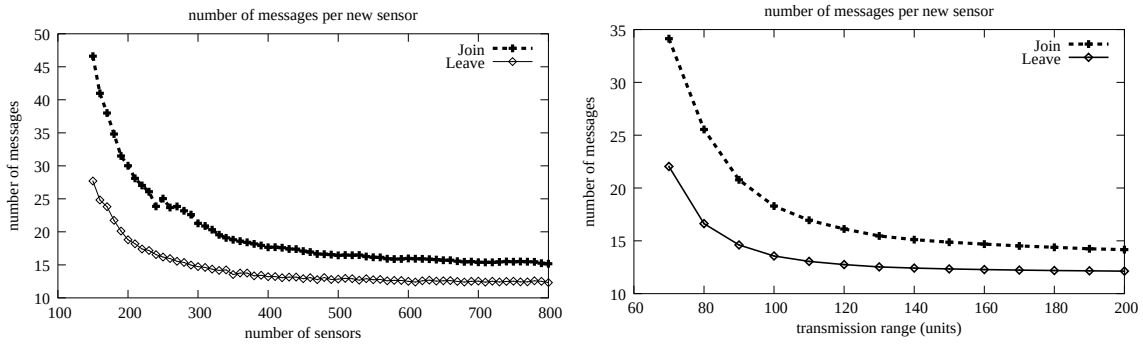


Fig. 2.16. Plots showing the total number of messages required for the deployment of new sensors.

tion. As before, we assume that all the sensors have similar capabilities, which for these experiments consist of identical radio transmission ranges. Note that the experiments in Section 2.6.1 only consider the sensing range.

In the first experiment, we initially place between 150 and 800 sensors in a region of size $1000 \times 1000\text{m}^2$. 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. 2.16(a) shows the results of this experiment. The average number of messages transmitted for each join/leave operation, decreases abruptly with the 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 routing 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. 2.16(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.

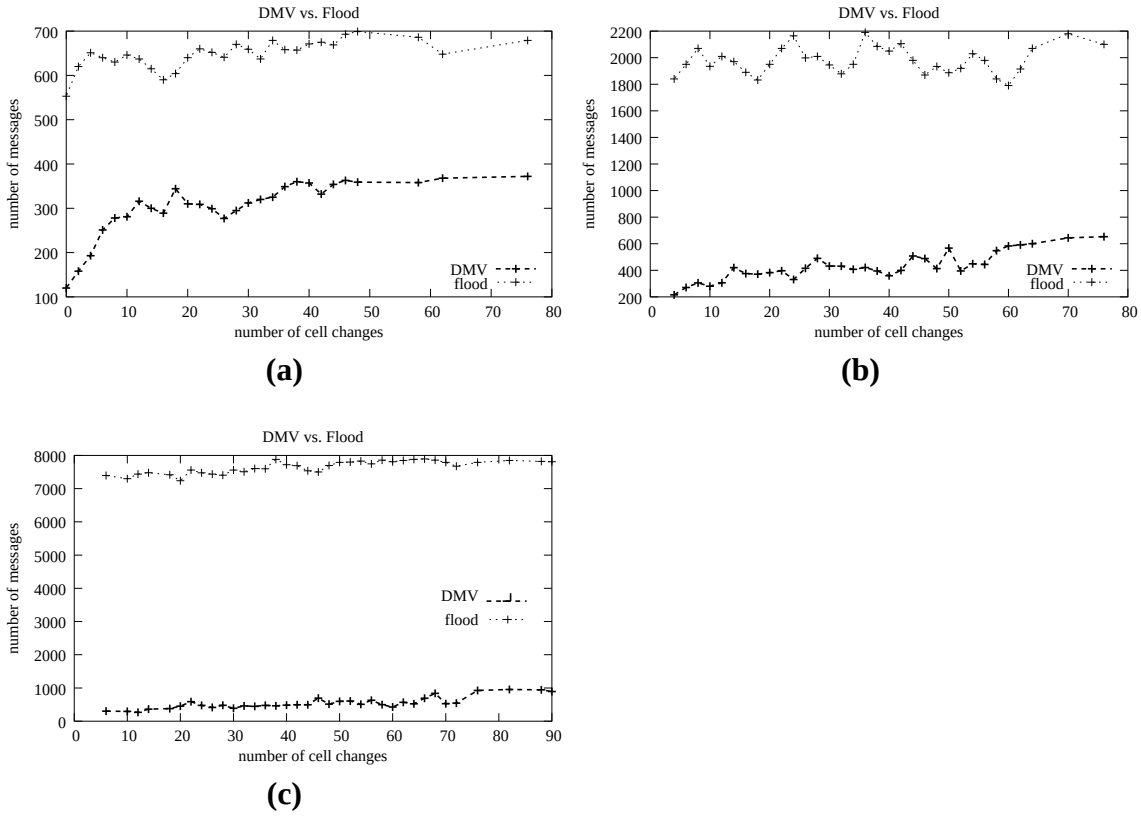


Fig. 2.17. DMV vs. flooding algorithm: evolution of 20, 30, and 40 sensors moving with a speed of 3 units/s in a square of size 200.

2.6.2 Dynamic Sensor Networks

In this section we experiment with moving sensor networks. The simulations are performed in a square area of size 200×200 units, where each sensor has a sensing range of 50 units and a transmission range of 40 units. Initially, the sensors are randomly distributed inside the square. We use a mobility model similar to the random waypoint model to simulate the movements of sensors. That is, each sensor chooses a random destination point, and starts moving towards it with the maximum velocity chosen for that experiment. After reaching the destination, the sensor chooses a new destination point and repeats the

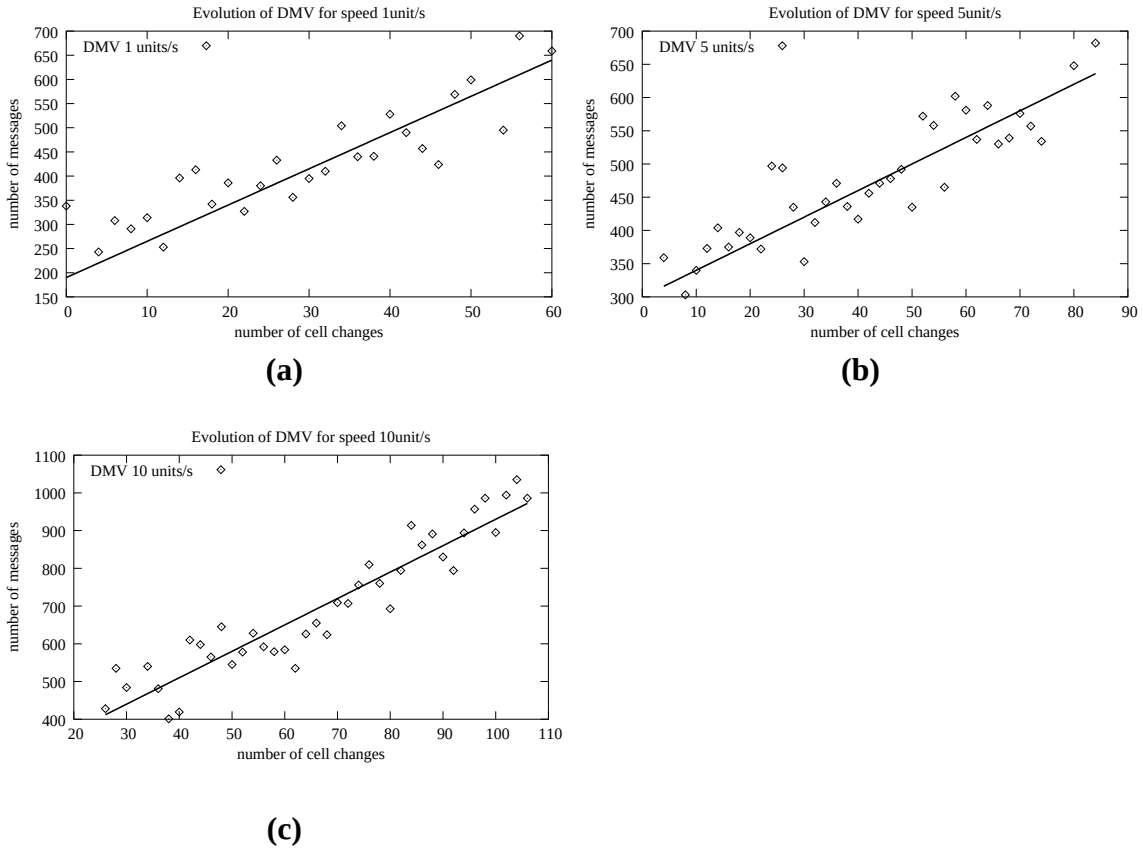


Fig. 2.18. Performance of DMV: evolution of 30 sensors at speeds of 1, 5, and 10 units/s.

process. All simulations are performed with 10 different random seeds, each simulation taking 90s.

In the first set of experiments we compare DMV against a baseline flooding algorithm, where each sensor floods the network with its new position information, every time it moves. For this set of measurements, we keep the speed of the sensors at a constant 3 units/s.

The experiments show the dependency between the total number of messages generated by DMV and the flooding algorithm and the total number of Voronoi cell changes (total NVCC), for 20, 30, and 40 sensors. For a sensor, NVCC represents the number of

Voronoi neighbors that have changed after the network has evolved into a different configuration as a result of sensor movements. The total NVCC for a network is the sum of the NVCCs for all sensors. Intuitively, DMV should require the number of messages exchanged to be proportional with the total NVCC, since the larger the number of cell changes, the larger the number of sensors that have to be notified about changes in their set of Voronoi neighbors.

Fig. 2.17 shows the results of our experiments. DMV not only performs better, but as the number of sensors increases, the difference in performance increases considerably. If for 20 sensors DMV is twice more efficient than the flooding based algorithm in terms of the total number of messages, for 40 sensors DMV is 8 times more efficient. Moreover, the plots show that every time the sensors move, around 20 messages are being sent by each sensor in order to be able to recompute its local Voronoi information and stabilize.

In the second set of experiments we measure the performance of DMV for networks of 30 sensors, moving at constant speeds of 1, 5 and 10 units/s. As Fig. 2.18 shows, the graphs for the different speeds overlap. This is normal, since we expect that the same NVCC will generate a similar number of messages irrespective of how fast the sensors move. In addition, the range of NVCC increases as the speed increases, since higher speeds imply a larger moving distance. For a speed of 10 units/s, the number of Voronoi cell changes can be greater than 100. However, the total number of messages required by DMV to stabilize grows almost linearly with the NVCC.

The variations noticed in the plots are due to two factors: (i) a smaller number of Voronoi changes can generate more updates, since updates are propagated by intermediate sensors, and (ii) even though a movement generates only few cell changes, many simple paths may be broken, increasing the number of messages necessary for routing to Voronoi generators.

2.7 Related Work

The problem of coverage of a set of entities has been studied in a variety of contexts. Among the early formulations of this problem is the "Art Gallery problem", which requires the placement of minimum number of observers so that every piece of artwork is visible to some observer [26]. Lieska, Laitinen and Lahteenmaki [27] present algorithms for optimal sensor placement with a view to optimizing specified service criteria. Haas [28] presents algorithms for optimizing coverage under constraints on message path length. In contrast, we focus on defining the coverage of a sensor network in an attempt to preserve the maximum coverage of the network while extending its lifetime.

2.7.1 Coverage-Boundary

The problem of network coverage is related to the problem of frequency assignment in cellular networks [29], 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, we do not assume global knowledge of the network topology.

The problem of sensor coverage has also received considerable attention in robotics (see [30] for a survey). Given a bounded domain the problem requires a robot equipped with a sensor to build a complete map of the environment without any initial knowledge. This requires the robot to pass through specified points of the unknown region. The notion of a *hierarchical generalized Voronoi graph* is used to incrementally construct the map using only line of sight data.

In other related approaches, some topology control algorithms [31] use Gabriel and RNG graphs to minimize the energy required to maintain network connectivity. Two devices A and B are said to be Gabriel neighbors if their diametric circle, i.e., the circle having AB as its diameter, does not contain any other devices. Two devices A and B are RNG neighbors if there does not exist any device closer to both A and B. One could

surmise that a device would check if its sensing circumference is completely covered by other devices by simply intersecting its disk with the disks of its Gabriel or RNG neighbors. However, as illustrated in Fig. 2.19, such simple constructions do not work correctly.

Fig. 2.19 shows that device B is not a Gabriel neighbor of device A since the circle with diameter AB has device C inside it. Note that device A is not on the coverage boundary since devices $B..G$ completely cover A's circumcircle of coverage. If device A would query only its Gabriel neighbors, it would discover that the arc xy on its circumcircle is not covered by any of them, and wrongly infer that it belongs to the coverage boundary. In Fig. 2.19(b) device B is not A's RNG neighbor since device C is closer to both A and B. Similarly, device F is closer to both A and G than A is to G. It is clear that by asking only its RNG neighbors consisting of device C, A will inaccurately decide that it is on the coverage boundary. A is not on the coverage boundary since the devices $B..G$ completely cover the circumcircle of A's coverage.

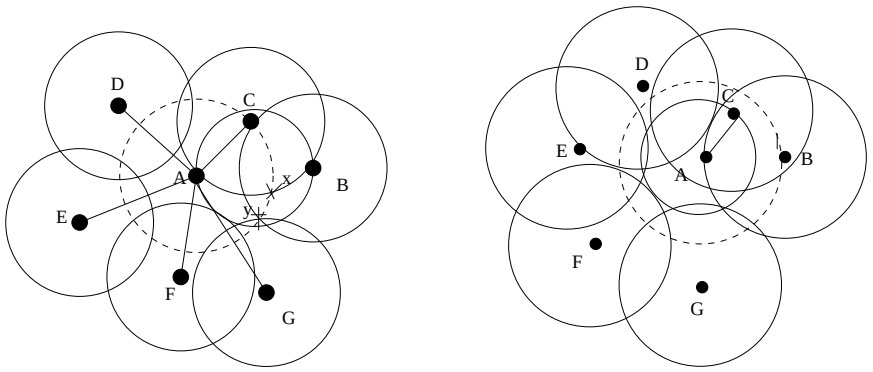


Fig. 2.19. Counter-example to the use of a simple Gabriel-graph or RNG-graph based algorithms: points represent devices, large circles centered around points represent devices' coverage areas. The dotted circle represents A's coverage circumcircle. The direct links show device A's Gabriel and RNG neighbors respectively.

Also, as shown in Section 2.3, the centralized coverage boundary problem has a $\Omega(n \log n)$ lower bound. Since the construction of the Voronoi diagram, which is the basis of our solution for the centralized coverage-boundary problem, takes $O(n \log n)$, our solution is optimal.

2.7.2 Coverage-Preserving Redundancy Elimination

Tian and Georganas [3] 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. Therefore, unlike our solution, this mechanism discovers only a subset of all the redundant sensors.

Zhang and Hou [32] provide a distributed algorithm for extending the network lifetime by turning off "redundant" sensors. Their mechanism for determining if a sensor is redundant requires a sensor to divide its coverage area into small grids and then to use a bitmap to indicate whether the center of each square of the grid is covered by some other sensor. For small values this method becomes expensive, and for large values it may be overly conservative. Since only the neighboring sensors are probed for grid coverage, they only find a subset of all the redundant sensors. However, we present a precise and efficient solution to this problem.

Ye et. al [33] present an algorithm that extends the network lifetime by maintaining a necessary set of working sensors and turning off redundant ones. A sensor is alternately sleeping or active. When a sensor wakes up, if it has an active sensor inside its transmission range, it turns off again. Hence, unlike our solution, their algorithm has no claim towards maintaining the coverage of the network.

Several other solutions have been proposed for related coverage problems. Slijepcevic and Potkonjak [34] 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. [35] define the coverage using the best covered and least covered paths between two sensors in the network as metrics. Fang et. al [36] present an algorithm

for routing around connectivity holes in a sensor network. Cardei and Wu [37] present an extensive survey of coverage problems in sensor networks. Shakkottai et. al [38] study the coverage of a unit square by a given number of sensors, under the assumption that sensor failures will affect the coverage. However, our work focuses on a different definition and purpose of coverage of wireless sensor networks.

2.8 Conclusions

In this chapter 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. Our redundant sensor elimination algorithm turns off simultaneously more than half of the total number of redundant sensors, approximately 20% of the total number of sensors deployed. The number of messages required to update the Voronoi neighborhood of sensors, due to events such as sensor failures or new sensor deployments, shows a significant decrease with the increase in sensor density. This is due to better chances of directly sending messages between Voronoi neighbors.

3. READER COLLISION AVOIDANCE IN RFID SYSTEMS

Radio Frequency Identification (RFID) systems consist of two types of components: RFID tags, comprising a small integrated circuit for storing information and an antenna used for communication and RFID readers capable of reading the information stored on non line-of-sight RFID tags placed in its vicinity and communicate it through a wired or wireless interface to a central database.

The investment of major retailers such as Wal-Mart and Tesco, mandating their manufacturers to place tags on cases and pallets is a serious motivation for the large scale deployment of RFID systems. This investment is based on recent technological advances that have made the mass production of very cheap tags, indeed possible , their cost being envisioned to drop below the 5 cents/tag threshold [39]. The main advantages of RFID systems are price efficiency (envisioned billions of dollars in savings for Wal-Mart [40]) and accuracy of stock management (GAP documented an increase of accuracy from 85% to 99.9% when using RFID technology [41]).

The miniaturization of readers (SkyeRead M1-Mini [2]), coupled with their enhancement with Wi-Fi or cellular capabilities (SmartCode [42]), broadens the range of applications for RFID systems. Wireless RFID systems, similar to sensor networks, can be deployed on-line instead of being statically pre-installed. Unlike sensor networks, Wireless RFID systems have the ability to decouple the sensing and communication functions. Since tags interfaced with external sensors, such as temperature and shock sensors or tamper indicators, have already been produced [1], wireless RFID systems can be easily extended with new sensing capabilities by deploying corresponding RFID tag types. Moreover, the existing compatibility between recent readers (SkyeRead M1-Mini [2]) and MICA2DOT motes makes possible the integration of wireless sensor networks and wire-

less RFID systems. A hybrid wireless sensor and RFID infrastructure combines the ease of deployment with affordable identification and monitoring of objects.

RFID readers identify tags placed in their vicinity by broadcasting queries. Tags receiving such queries reveal their identity using the energy of the received signal. During this detection phase, several tags may respond simultaneously to the query of an RFID reader, generating a *tag collision.* A tree walking algorithm [4] is used to solve this problem. The reader sends out prefixes of variable length, ranging from one to the maximum length of the tag identifier. For every prefix that is received, a tag will respond with an acknowledgment, if the prefix of its identifier matches with the prefix sent by the reader. A reader on listening to an acknowledgment will explore that part of the sub-tree rooted at the prefix. This procedure is performed until the maximum length of the prefix is equal to that of the tag identifier. In the absence of interference, an RFID reader can detect all the tags placed in its vicinity.

However, the tree walking algorithm will not solve the following problem, similar to the hidden terminal problem of ad-hoc networks. During the tag detection phase, readers that cannot communicate directly may simultaneously send prefix queries. Tags situated in the vicinity of two or more readers may then be unable to correctly decode the queries, leading to scenarios where readers erroneously conclude the absence of tags, matching their current queries, in their vicinity.

Previous solutions to the reader collision problem have relied on time or frequency division mechanisms [43]. However such solutions are difficult to implement in the absence of a centralized coordinator. Applying medium access control mechanisms from wired or wireless networks to RFID systems may prove to be equally difficult. Carrier sensing multiple access with collision detection (CSMA/CD), used in standard Ethernet, uses collision detection mechanisms and exponential back-offs to successfully transmit messages. However, such mechanisms cannot be used in RFID systems due to the inability of RFID readers of detecting reader collisions at tags. CSMA/CA uses ready-to-send (RTS) and clear-to-send (CTS) message exchanges between sending and receiving nodes in order to avoid collisions. In RFID systems, readers are not aware of the identities of tags in their

vicinity, making impossible the exchange of RTS/CTS messages between pairs of readers and tags. Moreover, CTS messages sent by multiple tags as a reply to a RTS broadcast of a reader may collide at readers, making their decoding impossible. Finally, the resource constraints of tags greatly reduce their ability of assisting readers in this process.

In this chapter we present a distributed and localized solution to the reader collision problem. Our Reader Collision Avoidance (RCA) algorithm, is based on foundational techniques in randomized algorithms. In RCA, a reader retransmits each query periodically at random intervals. This is done by dividing time into disjoint epochs and each epoch into multiple disjoint time frames, and having each reader pick a frame uniformly at random and send its current query during that frame. As proved in Section 3.3, the number of retransmissions per query, in the worst case, is $O(\log \psi)$, where ψ is the maximum number of readers in the system. This ensures with high probability that all tags within the vicinity of the reader are detected correctly. In Section 3.4, we experimentally demonstrate that in realistic scenarios, involving random deployment of readers and tags, much fewer retransmissions per query suffice to allow correct detection of tags.

Chapter Organization In Section 3.1 we describe in detail the reader collision issue. In Section 3.2, we describe our randomized algorithm (RCA) for the reader collision problem. We experimentally evaluate RCA in Section 3.4. We present our conclusions in Section 3.6.

3.1 The Reader Collision Problem

The area around a reader where tags can receive the reader's signal and their replies can be correctly decoded by the reader, is called the *interrogation zone* of the reader. The main functionality of readers it to detect the unique identifiers of all the tags placed in its interrogation zone.

When two readers are placed close enough for their interrogation zones to overlap but far enough to prevent their direct communication, tags placed within the intersection area of the interrogation zones may receive queries from both readers simultaneously. Such

queries, potentially part of the TWA protocol, will then interfere, preventing the concerned tags from correctly decoding the queries. Such tags may then become hidden, escaping detection by any reader in the system.

Previous solutions to this problem include time and frequency division mechanisms [43]. However such solutions are difficult to implement in the absence of a centralized controller, since readers may be oblivious to overlappings in their interrogation zones. Since readers can communicate with tags but are unable to route packets, reader-to-reader communications are local and can only occur through neighboring tags. Moreover, the considerable storage and processing constraints of tags further restrict the volume of such communications. The lack of collision detection mechanisms in RFID systems further complicate the problem. Tags affected by collisions may erroneously lead readers to conclude the absence of any tags matching the currently employed prefix.

3.2 Reader Collision Avoidance (RCA) Algorithm

We propose a randomized, distributed and localized solution to the reader collision problem. Our algorithm is presented in the context of a tree walking procedure. However, a similar approach can be extended to any scenario where a reader needs to communicate with a tag. Similar to TWA, in RCA a reader sends a broadcast query containing a certain prefix expected to match the identifiers of tags in its interrogation zone. However, unlike TWA, where the lack of an answer is considered to denote absence of matching tags, RCA backs-off for a random number of time frames and repeats the query. The purpose of the random back-off and query repetition is to ensure w.h.p. the choice of a time frame not picked by another reader, thus avoiding reader collisions.

The current design of the algorithm is made under the following conservative assumptions, thus, any relaxation of the conditions will only improve the performance of our approach.

- Our algorithm is applicable to any number of readers and tags.
- We make no assumptions on the underlying reader or tag topology.

- We do not assume the presence of a centralized entity capable of collecting the topology of the reader network or controlling the behavior of individual readers.

- We assume the presence of passive tags only, as opposed to active tags (the latter are more powerful and expensive). Therefore, tags are only able to answer readers by using the energy of their queries. Also, a tag has limited, read-only memory, used to store its corresponding unique identifier. The tag is however capable of doing prefix matching and in case of a match send back a message.

- Readers are able to detect tag collisions, occurring when multiple tags reply to the same query

The premise of the algorithm is as follows. A reader divides time into disjoint epochs and each epoch is further divided into multiple disjoint time frames. The above details can be made standard by programming them into each reader. In each epoch, a reader picks a frame uniformly at random and sends its query in that frame. If no tag answer is received, the reader repeats the query in a randomly chosen time frame of the next epoch. If a reader collision at matching tags has occurred during the query, the query duplication correlated with the random back-off decreases the chances of repeated reader collisions. Section 3.3 proves that if a query is not answered $O(\log\ \psi)$ times, w.h.p. there are no tags matching the query in the interrogation zone of the reader. If, however, an answer is received, either as a clear tag response or by detecting a tag collision, the reader recursively moves to the next query, as proposed in the TWA algorithm.

The choice of repeating a query up to $O(\log\ \psi)$ times is made under the conservative assumption that all readers interfere with each other at all tags. Hence, the bound that we provide is the worst case bound. However, this is not always the case. In our experiments (see Section 3.4), we show that in realistic scenarios of random deployment of readers and tags, much fewer repetitions are needed in order to allow readers to accurately detect tags.

The algorithm in Fig. 3.1 presents the pseudocode for RCA using an Orca [44] like syntax. Orca is a parallel programming language for distributed systems, that provides

```
1. Object implementation RFIDTag;
2.   T_id : integer; #tag identifier
3.   inQ : queue; #queue of incoming packets
4.   Operation run()
5.     guard prefixMatch(inQ.first, T_id) do
6.       bCast(new packet(TAG));
7.     od
8.   end

9.  Object implementation RFIDReader;
10.  count, nEpochs : integer; #epochs per bit read
11.  frame, n : integer; #time frames in each epoch
12.  T, T_out : integer; #time out value
13.  inQ : queue; #queue of incoming packets
14.  Operation treeWalk(prefix : integer)
15.    count := 0;
16.    while count++ < nEpochs do
17.      frame := getRandom(0, n);
18.      sleep(frame);
19.      T = getTime();
20.      bCast(new packet(prefix));
21.      guard inQ.first.type = TAG_COL || TAG do
22.        treeWalk(prefix + "0");
23.        treeWalk(prefix + "1");
24.      od
25.      guard getTime() − T ≥ T_out do
26.        sleep(n − frame − 1);
27.      od
28.    od
29.  end
```

Fig. 3.1. The generic reader and tag behavior. $getRandom(v_1, v_2)$ returns a random integer value between v_1 and v_2 and $bCast(packet)$ is used to broadcast $packet$.

elegant constructions for expressing reactive behavior, such as *guards*. Operations can consist of one or more guards with syntax

guard `expression` **do** `statementSeq` **od**,

where `expression` is a boolean expression and `statementSeq` is a sequence of statements. The operation containing guards blocks until one or more guards are true. Then one of the satisfied guards is randomly chosen and its statements are executed atomically.

The operation of a tag is shown in Fig. 3.1, lines 1-8. A tag will reply only to queries containing strings whose prefixes match its own identifier (lines 5-7). `inQ.first` is used to denote the packet currently received by the tag. The operation of a reader is shown in Algorith 3.1, lines 9-29. We divide time into epochs, where each epoch contains a fixed number, `n`, of time frames. The duration of a time interval is equal to the time necessary for a query to propagate from a reader to a tag. For each prefix queried, the reader waits for a maximum of `nEpochs` epochs (line 16) and in each epoch sends exactly one broadcast message containing the prefix. During each epoch, the broadcast message is sent in a randomly chosen time frame (lines 17-20).

The lack of a reply can denote either the absence of a tag in the interrogation zone, matching the queried prefix, or the occurrence of reader collisions at such tags. Then, if less than `nEpochs` queries with the current prefix have been sent, the reader waits until the beginning of the next epoch to repeat the above process (line 22). If no reply or collision is detected after `nEpochs` rounds, the reader ignores the subtree rooted at the queried prefix. However, the receipt of an individual reply or the detection of a tag collision stops this process, since the reader can now safely recurse on the two children of the employed prefix (lines 21-24).

Incorrect Tag Detection. Note that readers cannot interpret replies of covered tags, queried by other readers, as answers to their own queries. As an example, consider Fig. 1.4, where the tags are $\mathtt{T}_1 = \mathtt{11101}$, $\mathtt{T}_2 = \mathtt{10110}$ and $\mathtt{T}_3 = \mathtt{01100}$. If $\mathtt{R}_1$ sends a ”0” query while $\mathtt{R}_2$ sends a ”1” query, $\mathtt{T}_1$’s reply for $\mathtt{R}_2$ will not be incorrectly interpreted by $\mathtt{R}_1$ as the presence of a tag with ”0” as prefix, in its interrogation zone. This is because either (i) the simultaneous transmissions of $\mathtt{R}_1$ and $\mathtt{R}_2$ will interfere at $\mathtt{T}_1$ and no reply will be generated by $\mathtt{T}_1$ or (ii) if the query of $\mathtt{R}_2$ occurs later than the query of $\mathtt{R}_1$, the answer of $\mathtt{T}_1$ will not be interpreted by $\mathtt{R}_1$ as an answer to its query.

3.3 Analysis

We present an analysis of RCA based on two fundamental abstractions in randomized algorithms, *viz.* the coupon collector abstraction and a balls and bins paradigm. For the sake of completeness, we define the coupon collector problem as in Motwani and Raghavan [45].

Coupon-Collector Given a set of coupons containing n unique coupon types, the number of samples required to obtain w.h.p. a coupon of each type, is nH_n, where H_n is $\mathtt{O}(\log \mathtt{n})$.

Let ψ be the total number of readers and γ the total number of RFID tags in the system, τ be the number of time frames per epoch and β be the bit size of RFID tag identifiers. Our first goal is to evaluate the number of epochs per query, x, required to ensure the success of the query. To establish an upper bound, we assume a star topology in which interrogation zones of all ψ RFID readers share all τ RFID tags. Note that this is a worst case assumption. Each frame is considered to be a bin and a query of an RFID reader is modeled as a ball. We first prove the following lemma.

Lemma 3.3.1 *In each epoch of the RCA process, the expected number of readers that send a message without a collision is* $\psi e^{-\frac{\psi}{\tau}}$.

Proof When ψ readers send a message uniformly at random in any one of the τ frames of an epoch, the distribution of the messages in each frame follows a Poisson distribution [45]. Therefore, if X_i is a random variable that is equal to the number of messages sent by different readers in frame i, the probability of exactly one message being sent in frame i is given by

$$P(X_i = 1) = \frac{\psi}{\tau} e^{-\frac{\psi}{\tau}}$$

Since there are τ frames, the average number of frames where exactly one message is sent is $\psi e^{-\frac{\psi}{\tau}}$. ■

Using the coupon collector paradigm we can prove the following lemma.

Lemma 3.3.2 *The RCA process is dominated by the coupon collector process.*

Proof In RCA, an RFID reader sends a query until the upper bound, x, is reached. The approach can be modeled as a coupon collector process, where each reader is a coupon type. A coupon type is chosen if the query sent by the corresponding reader during its chosen time frame of the current epoch is the only query sent by a reader during the same time frame. From Lemma 3.3.1, on average $\psi e^{-\frac{\psi}{\tau}}$ coupon types are selected during each frame. This is similar to choosing $\psi e^{-\frac{\psi}{\tau}}$ coupons (of the coupon collector process) and then placing back the chosen coupons into the set, instead of choosing a single coupon and replacing it immediately. This increases the rate at which the coupon types are chosen. Thus, the number of epochs needed for each RFID reader to send the only query during a time frame is at most the number of samplings in the actual coupon collector process. ■

We can now prove the following theorem, providing an upper bound on the number of query repetitions in RCA.

Theorem 3.3.1 *Setting the number of time frames per epoch, τ, to be the total number of readers, ψ, in RCA, requires only $\mathtt{O}(\log \psi)$ query repetitions to ensure w.h.p. the receipt of a reader's query by the target RFID tags in its interrogation zone.*

Proof If x is the number of query repetitions, using Lemma 3.3.1 and Lemma 3.3.2, we get

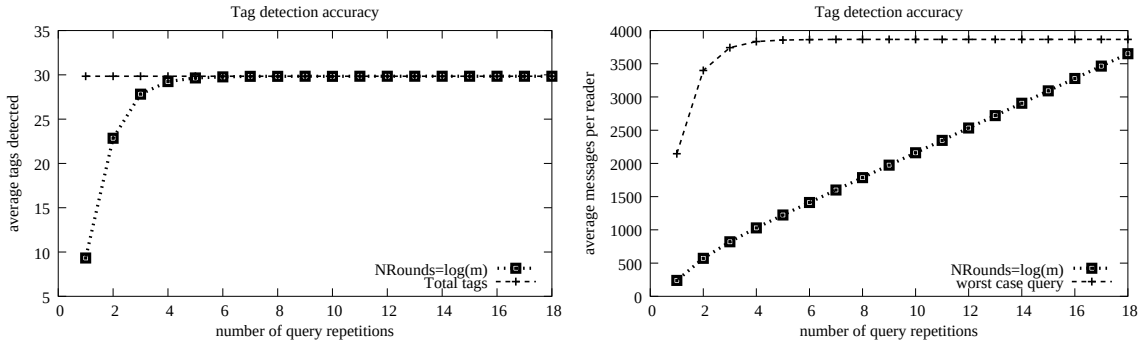
$$x\ \psi\ e^{-\frac{\psi}{\tau}} \leq c\psi \log \psi.$$

When $\tau = \psi$, $x = O(\log \psi)$. ■

We can now provide the worst case time complexity of RCA.

Complexity of RCA The time complexity of RCA, $\mathtt{T_{RCA}}$ is $\mathtt{O}(\gamma \log \beta \log \psi)$ time epochs.

Proof Since each reader covers γ tags of bit size β, the number of query types is $\mathtt{O}(\gamma \log \beta)$. Theorem 3.3.1 completes the proof. ■



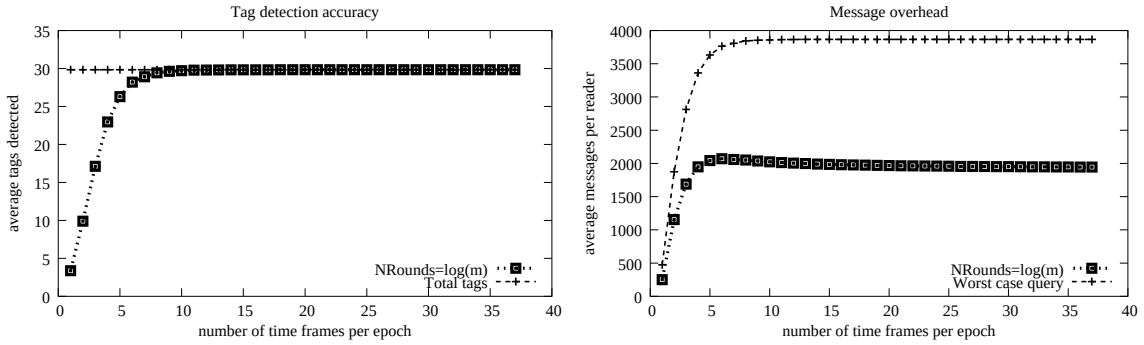
(a) **(b)**

Fig. 3.2. Performance of RCA when the number of epochs per query grows from 1 to $2 \log\ \psi$, for a total of 500 readers and 4000 tags. The number of time frames per epoch is $2 \log\ \psi$.

3.4 Simulation and Results

In this section we experimentally analyze the accuracy and message overhead introduced by our randomized solution to the reader collision problem. We compare the performance of RCA with the simple tree walk algorithm [4] and with a version of RCA, that we call RCAv.1. In RCAv.1, a reader sends each query the maximum number of times, irrespective of the result of the query. Note that in RCA, a reader will not repeat a query if the result is a success, that is, it receives an answer from a tag or it detects a tag collision. All our experiments are performed by randomly (uniformly) deploying tags and readers in a $1000 \times 1000m^2$ square.

We first evaluate the performance of RCA as a function of the number of time epochs used per query. For this, we randomly place 4000 tags and 500 readers having an interrogation radius of 50m in the $1000 \times 1000m^2$ square area and increase the number of epochs per query from 1 to $2 \log\ \psi$. Fig. 3.2(a) shows the average number of tags detected by a reader, compared with the average number of tags actually placed in the interrogation zone of the reader. The number of tags discovered by a reader quickly converges to the number of tags placed in its interrogation zone. For 9 epochs ($\log\ \psi$) per query our randomized



(a) **(b)**

Fig. 3.3. The performance of our solution when the number of time frames per epoch increases from 1 to 38, for a total of 500 readers and 4000 tags. The number of time epochs per query is $\log\ \psi$.

algorithm allows any reader to discover *all* the tags placed in its interrogation zone. This explains the usage of $\log\ \psi$ epochs per query in the following experiments.

Fig. 3.2(b) shows the number of messages generated by RCA compared with the number of messages generated by RCAv.1. The number of messages of our protocol increases linearly with the number of queries. However, for 9 epochs per query, RCA sends less than half of the messages of RCAv.1.

In the following experiment, we evaluate the performance of RCA when the number of time frames per epoch increases from 1 to 38. The total number of tags is 4000 and the number of readers is 500 with an interrogation radius of 50m. We perform the experiments using $\log\ \psi$ epochs per query. Fig. 3.3(a) depicts our observations. As expected, the number of tags detected by a reader increases with the increase on the number of frames per epoch, quickly converging to the actual number of tags physically located in the interrogation zone of the reader. This is because a larger number of frames per epochs decreases the chances of reader collisions. For 9 ($\log\ \psi$) frames per epoch, our algorithm allows readers to detect on average 0.2 less tags than the ones physically located inside the reader's interrogation zone. However, *all* the tags are detected when the number of frames per epoch is 18 ($2 \log\ \psi$). This can be explained as the result of uniformly distributing the

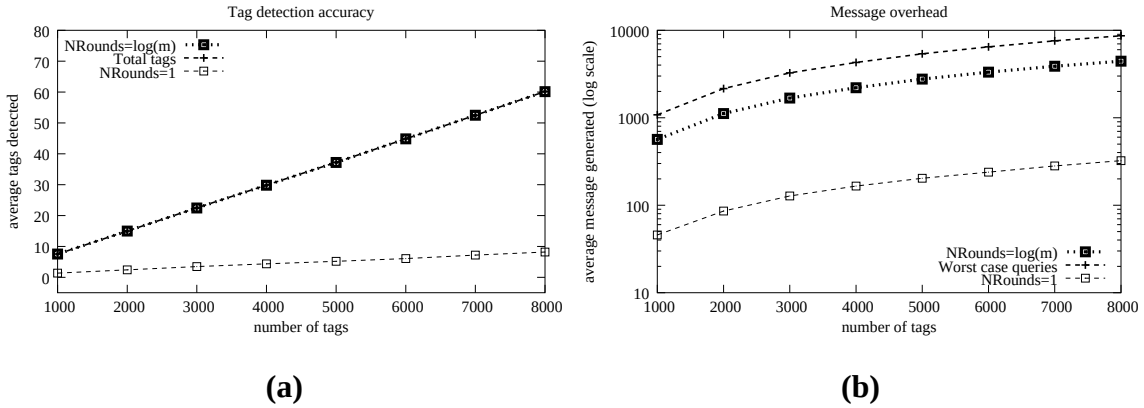


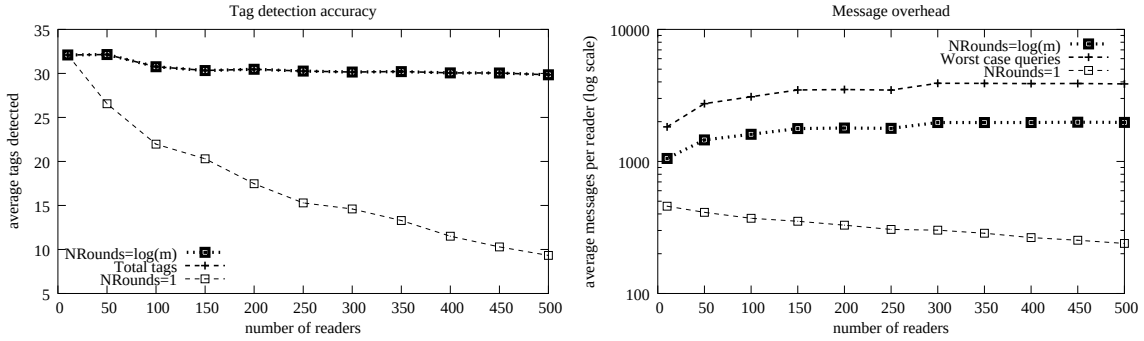
Fig. 3.4. Testing the scalability of our solution when the number of tags increases from 1000 to 8000, while the number of readers is 1000.

tags and the readers in the square. It is well known that when b balls are thrown uniformly at random in b bins, the maximum number of balls in any bin is $O(\log b)$ [45].

Fig. 3.3(b) shows the average number of messages per reader generated for this scenario. In order to read all the tags situated in its interrogation zone, whose identifiers have 12 bits, a reader sends on average 429 query types, or up to 2000 messages, This implies sending each query on average 5 times, when our algorithm places a bound of 9 ($\log\ \psi$) consecutive epochs per query. Thus, by not repeating successful queries, our randomized algorithm saves on average 4 messages per query.

Moreover, for small epoch sizes the increase in epoch size generates significant increases in the number of messages generated. This is because smaller epochs increase the chance of collisions, leaving large parts of the tag name trees untraversed. This is confirmed by Fig. 3.3(a), where for small epoch sizes the number of tags is also very small. However, the number of messages generated by RCA reaches the maximum at 5 frames per epoch. Subsequently, as the number of frames per epoch increases, the number of messages decreases. This behavior is expected, since larger epoch sizes imply lower chances of reader collisions, hence faster detection of successful queries.

In the next simulation, we experiment with between 1000 and 8000 randomly placed tags, while maintaining the number of readers constant, 1000. The interrogation radius



(a) **(b)**

Fig. 3.5. Testing the scalability of our solution when the number of readers increases from 10 to 500, for 4000 tags.

of readers is set to 50m. The number of time frames per epoch is set to $2 \log \psi$ both for our randomized algorithm and for the tree walk algorithm. Fig. 4.4(a) shows the average number of tags detected by a reader, when using RCA ($\log \psi$ epochs per query) and when using TWA (one epoch per query), compared with the average number of tags placed inside the interrogation zone of readers. While RCA is very accurate, with an average difference of 0.02 tags/reader from the actual value, TWA discovers around 7 times less tags per reader than it should.

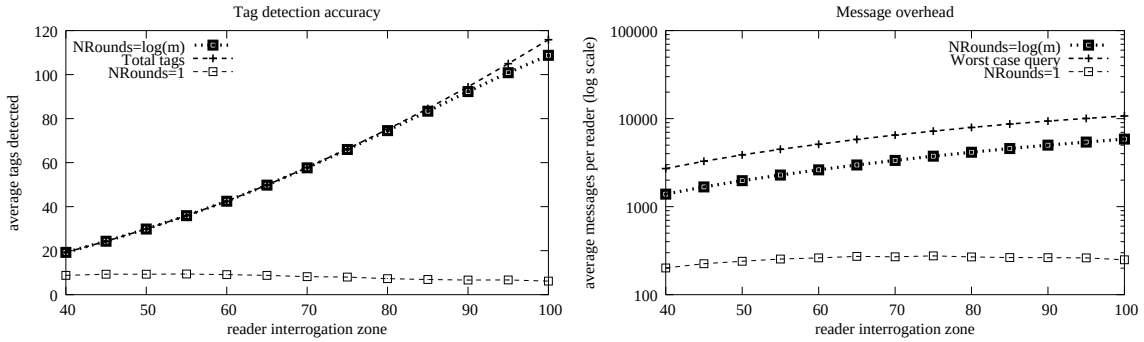
Fig. 4.4(b) shows the corresponding number of messages per readers generated by the two algorithms, on a logarithmic scale, compared with the total number of messages per reader generated by RCAv.1. While RCA generates 10 time more messages than TWA, this is simply due to the fact that the number of successful queries for the tree walk algorithm is around 3 times less than in our case. This also explains why the tree walk allows a reader to detect only a small fraction of the number of tags covered. However, by stopping with a given query when the result is detected to be a success, our algorithm reduces by half the number of messages generated by RCAv.1.

In the following, we measure the performance of RCA when we increase the number of readers from 10 to 500 but keep the number of tags distributed at 4000. The interrogation radius of readers is 50m and the number of time frames per epoch is set to $2 \log \psi$

both for our randomized algorithm and for the tree walk algorithm. Fig. 4.6(a) shows the performance in terms of the number of tags discovered by RCA and by TWA. For a small number of readers, the simple tree walk algorithm accurately detects the tags deployed in the interrogation zone of readers. This is because the interrogation zones of readers barely intersect, practically eliminating reader collisions. However, as the number of readers increases, effectively increasing the overlapping areas of the interrogation zones of readers, the accuracy of TWA quickly decreases. In contrast, RCA, by using $\log\ \psi$ epochs per query is very accurate, consistently discovering *all* the tags deployed.

Fig. 4.6(b) shows the average number of messages generated by readers when RCA, TWA and RCAv.1 are run by readers in the scenario described in the previous paragraph. The values are shown on a logarithmic scale. As before, the simple tree walking algorithm generates few messages, since only few queries are successful, leaving most of the tag name trees untraversed. Moreover, note that for TWA the number of messages per reader decreases as the number of readers increases. Initially, when only 10 readers are deployed, their interrogation zones barely overlap, allowing them to detect most of the tags covered and generating almost half of the messages generated by our randomized algorithm. As the number of readers deployed increases, so does the number of reader collisions, detecting less tags and hence generating less messages. However, the number of messages generated by RCA quickly saturates and is only half of the messages generated by RCAv.1.

The last experiment evaluates the performance of RCA when the interrogation radius of readers increases from 40m to 100m, while the number of readers randomly deployed is 500 and the number of tags is 4000. The number of time frames per epoch is $2\log\ \psi$ for the entire experiment. Fig. 4.7(a) shows the accuracy of RCA compared with TWA. Our algorithm discovers *all* the tags until the interrogation radius reaches 85m. However, even for an interrogation radius of 100m, the readers running our algorithm detect on average only 7 out of 115 tags less than they should. This is explained by the observation that as the interrogation radius increases so does the size and number of intersections of interrogation zones of readers. Since the number of epochs per query, $\log\ \psi$ and the number of time frames per epoch is constant, more collisions are generated, leading to a



(a) **(b)**

Fig. 3.6. Testing the performance of our solution when the interrogation zone of the readers increases from 40m to 100m, for 500 readers and 4000 tags. We performed the experiments such that even when the interrogation zone is 40m the readers completely cover the tags.

decreased accuracy. However, the performance of TWA is considerably inferior. When the interrogation radius of readers is 100m, readers running TWA discover only 5% of the tags detected by readers running RCA.

Fig. 4.7(b) shows the number of messages generated by the same algorithms. It confirms the results shown in Fig. 4.4(b) and Fig. 4.6(b). TWA generates only a fraction of the messages generated by RCA, since the number of queries correctly detected as successful is very small. However, we consistently reduce the number of messages sent by eliminating repetitions of successful queries.

3.5 Related Work

The reader-collision problem in systems was first documented in [43]. The solution proposed, of allocating different frequencies to interfering readers, is centralized. A simple decentralized version, where readers listen for collisions and use randomized backoff when detecting one, is discussed. In contrast, our work assigns different time slots for transmitting readers. Moreover, our solution guarantees w.h.p. that each reader is able to correctly read all the tags placed in its interrogation zone.

Closest to our goal is the work of Waldrop et. al [46]. They propose Colorwave, a decentralized Medium Access Control (MAC) protocol for reader networks whose purpose is to allocate disjoint time slots for reader transmissions. The protocol is based on the presence of an interference graph whose links denote interference between the end-points corresponding to readers. Hence, an interesting extension to this work would be a description of the interference graph construction. As shown in Fig. 1.4, interference at certain tags is difficult to detect, since even the presence of such tags may not be known.

Privacy-related issues of RFID systems have been extensively studied in [4, 47, 48]. A detailed description of computation and communication mechanisms and constraints of RFID systems, together with several suggestions for RFID protection are presented in [4]. A solution for preserving the privacy of tags, using hash functions for locking tags, is proposed in [47]. Locked tags are prevented from revealing their unique identifier until unlocked with the corresponding inverse hash value. The work in [48] provides an in-depth presentation of security and privacy challenges of RFID systems and proposes the use of additional, ”blocker” tags in order to prevent unauthorized readers from accessing protected tags.

Medium Access Control (MAC) protocols for wired and wireless networks share several characteristics with our reader collision avoidance algorithm. ALOHA [49] was the first MAC protocol for packet radio networks In ALOHA [49], when the transmission of a node results in collision, the node has to wait for a random interval before retransmitting. However, RFID systems do not have the mechanisms to detect collisions occurring at tags, making ALOHA unsuitable for avoiding reader collisions. IEEE 802.11b [50] is based on a multiple access with collision avoidance (MACA) [51] protocol that employs a handshake to avoid hidden-node problems. The sender broadcasts an RTS message and the receiver replies with a CTS message. All the nodes that hear the RTS and CTS messages delay their transmissions. Such a protocol cannot be used in RFID systems, since the purpose of a reader is to detect *all* the tags in its interrogation zone. Such a reader does not know the identities of the tags and thus cannot send individual RTS messages. Moreover, the simultaneous reception of CTS messages initiated by tags leads to tag collisions.

Carrier sensing multiple access with collision detection (CSMA/CD) [52], employed in the standard Ethernet is based on the ability of nodes to detect collisions. After detecting a collision, a node waits a random interval before retransmitting. In case of subsequent collisions, the node wait twice as much before attempting to retransmit, also known as exponential back-off. However, as noted before RFID systems lack the ability of detecting collisions.

3.6 Conclusions

In this chapter we present a distributed, local, and randomized solution to the reader collision problem in RFID systems. Current solutions to the problem are dependent on a coordinator and are based on time division or frequency division multiplexing. Our system is scalable in the number of readers and tags and is not dependent on a central coordinator. We give an analysis based on fundamental abstractions of the coupon collector process and the balls and bins paradigm. In our simulations RCA enables readers to accurately detect the tags placed in their interrogation zones with a small number of message repetitions ($\log \psi$) while requiring only $2 \log \psi$ time frames for each message. Moreover, by stopping the process as soon as a tag answer is received, RCA is able to reduce to half (on average) the total number of messages required to achieve the desired accuracy.

4. REDUNDANT-READER ELIMINATION IN WIRELESS RFID SYSTEMS

The dense deployment of wireless RFID systems, due to the need of accurate monitoring of areas of interest, coupled with the limited battery lifetime of wireless readers, raise several important problems. In this paper, we address the problem of extending the lifetime of wireless reader networks. We propose a solution based on the identification of redundant readers, whose deactivation will not affect the tag coverage of the initial reader network. We define redundant readers in terms of covered tags instead of continuous areas of coverage. In Figure 4.1, all readers are redundant, however, only a subset of the readers may be simultaneously deactivated.

While the problem of determining coverage redundancy has been extensively studied in wireless sensor networks [3, 32, 33, 53], it differs from the redundant reader elimination problem in several aspects. First, coverage is defined in terms of contiguous circular areas associated with sensors, whereas in RFID systems coverage is defined in terms of discrete points (RFID tags). Second, it relies on the existence of location information or at least the ability to estimate distances between adjacent sensors. Due to the limited resources of tags, we claim that in RFID systems such an assumption is not reasonable. Third, the limited resources of tags coupled with the potential inability of readers of acting as routers, considerably restrict the solution space of the redundant reader problem.

In this chapter we prove that even with centralized knowledge of the RFID system topology, an optimal solution for the redundant reader elimination problem is NP-hard. We present a randomized, decentralized and localized approximation algorithm for the redundant-reader elimination problem, called RRE. For each reader, the first step of RRE consists in detecting the set of tags placed in its vicinity. Similar to RCA, the difficulty of RRE rests on the potential occurence of reader collisions at tags. The absence of global

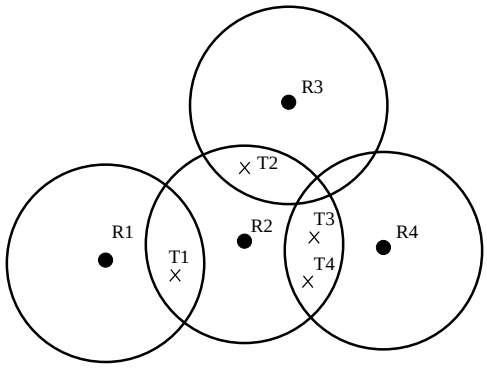


Fig. 4.1. Redundant reader example: readers R_1, R_2, R_3 and R_4 are redundant since the tags covered by each is covered by at least one other reader. This redundancy information would not be detected by a sensor redundancy detection algorithm, since the coverage areas of any of the readers are not subsumed by the others. The optimal solution requires only R_2 to be active, while the other readers may be turned off.

topology information, where readers might not be aware of such events, makes the task of accurate query scheduling difficult. During the first step of RRE, each reader attempts to write its tag count (number of covered tags) on all its covered tags. A tag placed in the vicinity of several readers will overwrite the count stored on behalf of a reader only if the new value is larger. The reader that has issued the highest count for a tag, will *lock* the tag. Then, in the last step of RRE, each reader sequentially queries all its covered tags to discover the ones it has locked. A reader that has not locked any of its covered tags is declared redundant.

Chapter Organization Section 4.1 describes the constraints of the RFID system considered in this work. Section 4.2 introduces the redundant-reader elimination problem and proves its NP-hardness. Section 4.3 presents our distributed and localized solution, proposes ways to avoid reader synchronization requirements and algorithms for adapting the solution to topological changes and analyzes the complexity of the solution. Section 4.5 presents the efficiency of our solution and the simulation environment used for experiments and Section 4.6 draws the conclusions.

4.1 Network Model

The design of our algorithms is made under the following conservative assumptions, thus, any relaxation of the conditions will only improve the performance of our approach. Our algorithm is applicable to any number of readers and tags and we make no assumptions on the underlying reader or tag topology. We do not assume the presence of a centralized entity capable of collecting the topology of the reader network or controlling the behavior of individual readers. Thus, our algorithm does not rely on the ability of readers to communicate. Furthermore, we assume the presence of passive tags only, as opposed to active tags (the latter are more powerful and expensive). Therefore, tags are only able to answer readers by using the energy of their queries. Also, a tag has limited memory. Part of it is read-only, used to store its corresponding unique identifier. However, tags also have writable memory. Moreover, a tag is capable of doing prefix matching and send a reply message in case of a match. readers are able to detect tag collisions, occurring when multiple tags reply to the same query.

4.2 The Redundant-Reader Problem

In random deployment scenarios, due to the small interrogation zone of readers and the requirement of coverage of tags, it is desirable to distribute large numbers of readers. Consequently, many readers will be redundant, their absence not affecting the coverage of the reader network. A subset of all redundant readers can be safely turned off in order to save their battery power. Deactivated readers can later be re-activated to replace failed readers or to balance the battery usage of other redundant readers, effectively extending the lifetime of the reader network.

In this section we study the reader redundancy problem and provide an efficient distributed algorithm for correctly detecting and deactivating redundant readers, without leaving any of the originally covered tags uncovered. We first formally define redundant readers.

Definition 4.2.1 *A redundant reader covers a set of tags, also covered by other readers.*

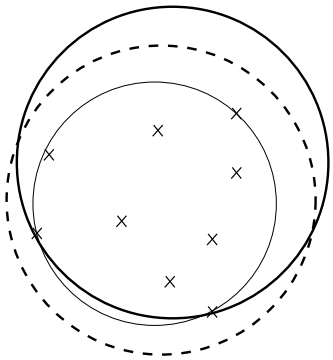


Fig. 4.2. Set of points covered by a circle of radius R, shown with an interrupted perimeter. There is a circle of radius R going through points A and B and covering all the other points. Shrink this circle until it first touches one more point. The resulting circle, has radius less than or equal to R.

According to this definition, all the readers in Fig. 4.1 are redundant. A simple solution to detect the redundant readers is to have all readers simultaneously broadcast a query containing the empty string. Since all the tags that receive such a query must answer, a reader that receives no reply is redundant. This is either because the reader covers no tag, or because interference occurred at all its covered tags. Such a solution has two important drawbacks. First, it requires time synchronization between all readers. Second, turning off all the redundant readers may leave blind spots [3]. We define a blind spot in this context as a tag that was covered by at least two redundant sensors, but whose concurrent deactivation leaves the tag uncovered. For example, in Fig. 4.1, the simultaneous deactivation of $\mathtt{R}_1$ and $\mathtt{R}_2$ leaves tag $\mathtt{T}_1$ uncovered.

In order to maximize the number of readers that can be deactivated, the minimum number of readers that cover all tags need to be discovered. We define then the redundant reader problem.

Redundant-Reader Problem Given a set of tags and a set of readers covering all the tags, find the minimum number of readers that cover all the tags.

For example, in Fig. 4.1, $\mathtt{R}_2$ is the only reader that needs to be active. In order to prove that the redundant-reader problem is NP-hard, we first prove the following lemma, illustrated in Figure 4.2.

Lemma 4.2.1 *Given a set of* n *points placed inside a circle of radius* R*, there exist 3 out of the* n *points whose circumcircle, of radius less than or equal to* R*, covers all the points.*

Proof If all n points are covered by a circle of radius R, then a circle of radius R going through 2 of the points and covering all the other points exists (see Figure 4.2). If the circle has a third of the n points on its perimeter, then we have completed the proof. Otherwise, shrink the circle until its perimeter touches a third point. The resulting circle has radius less than R, is the circumircle of three of the n points and covers all the other points. ■

We can now prove the following important result.

Theorem 4.2.1 *The redundant-reader problem is NP-hard.*

Proof We prove the NP-hardness of the redundant-reader problem by reduction from the geometric disk cover (DC) problem, known to be NP-hard [54]. The input for the DC problem consists of a set of points and a value R. The output consists in the minimum number of disks of radius R that cover all the points.

We use the following polynomial-time reduction from DC to the redundant-reader problem. Add a disk of radius R centered at each point in the input set of DC. Then, for all combinations of 3 points of the input set of DC, add a disk of radius R, centered at the mass center of the 3 points. It is clear that all the points are covered. Moreover, due to Lemma 4.2.1, any disk in the solution for the DC problem is contained in the set of disks introduced by our transform. The reduction has $O(m^3)$ complexity, where m is the number of input points. Using an algorithm for the redundant-reader problem, we can find the minimum number of disks needed to cover the points. This is the solution for the DC problem, which implies that the redundant-reader problem is NP-hard. ■

4.3 The Redundant Reader Elimination Algorithm

In this section we propose a randomized, distributed and localized approximation algorithm for the redundant-reader problem. As specified in Section 4.1, we make no assumption on the topology of the reader network, effectively claiming no direct communication between readers. We assume however the existence of writable tags, able to store information upon requests from in-range readers. We assume that RCA, where each reader has previously collected information from all the tags placed in its interrogation zone (see Chapter 3), has been executed by all readers.

RRE consists of two steps. In the first step, each reader attempts to write on all its covered tags the number of covered tags. A tag only stores the highest value seen, along with the identity of the corresponding reader. For this, each reader issues a write command containing its reader identification number and the number of tags covered. Similar to RCA, the write operation is performed exactly once during each epoch, for $e \log \psi$ time epochs, where ψ is the number of readers. During each epoch, the time frame for the write operation is randomly chosen. At the completion of $e \log \psi$ epochs, each tag stores the highest number of tags covered by a reader situated in its vicinity, along with the identity of that reader, called *locker* of the tag.

During the second step, a reader queries each covered tag and reads the identity of the tag's locker. A reader that locked at least one tag is responsible for monitoring the tag and will have to remain active. However, a reader that has not locked any tag can be safely turned off. This is because all the tags covered by the reader are already covered by other readers that will stay active. The reading queries issued by a reader for each of its tags is similarly repeated during random time frames for $e \log \psi$ time epochs, in order to avoid reader collisions occurring at that tag.

The algorithm in Fig. 4.3 illustrates our solution, that assumes writable RFID tags. The functionality of a writable tag is shown in operation `run` of `WritableRFIDTag` (lines 4-13). The reader and tag objects inherit the corresponding variables defined in Fig. 3.1 (see Chapter 3). When a writable tag receives a write command, containing the identifier

```
1. Object implementation WritableRFIDTag;
2.   R_id : integer; #identifier of locking reader
3.   count = 0 : integer; #count of highest bidder
4.   Operation run()
5.     guard inQ.first.type = write do
6.       if inQ.first.c > count then
7.         R_id := inQ.first.rid;
8.         count := inQ.first.c;
9.       fi;
10.    guard inQ.first.type = read do
11.      bCast(new packet(T_id, R_id, count));
12.    od
13. end

14. Object implementation RFIDReader;
15.  R_id : integer]; #reader identifier
16.  tags : array[integer] of integer; #covered tags
17.  redundant = true : boolean;
18.  Operation isRedundant(prefix : integer)
19.    while count + + < nEpochs do
20.      frame := getRandom(0, n);
21.      sleep(frame);
22.      bCast(new packet(write, R_id, tags.size));
23.      sleep(n − frame − 1);
24.    od
25.    for i in 1..tags.size do
26.      while count + + < nEpochs do
27.        T = getTime();
28.        frame := getRandom(0, n);
29.        sleep(frame);
30.        bCast(new packet(read, tags[i]));
31.        guard inQ.first.tid = tags[i] do
32.          if inQ.rid ! = R_id then
33.            redundant := false;
34.          od
35.        guard getTime() − T > n do od
36.      od
37.    od
38.    if redundant = true do turnOff(); fi
39. end
```

Fig. 4.3. The generic RFID reader and writable tag behavior for detecting redundant readers.

of the reader issuing the command and its number of covered tags, it copies the values locally if the number of covered tags is larger than the value currently stored. When the command received is a read, the tag returns a packet containing its identifier followed by the reader identifier and count value locally stored.

The detection of redundant readers is exhibited in operation `isRedundant` of `RFIDReader` (lines 18-39). First, a reader selects a random time frame during each of `nEpochs` consecutive epochs, and sends a broadcast write packet containing its identifier and tag count (lines 19-24). Subsequently, it queries each of its covered tags, using a read command, for `nEpochs` consecutive time epochs, in order to find the tag's locker (lines 25-37). Note that after sending a read command, at the chosen time frame, the reader waits either to receive a reply from the queried tag or for the epoch to end (lines 31-35).

4.3.1 Extensions

Synchronization We have assumed until now that all readers have already executed RCA, detecting all the tags placed in their interrogation zone. This assumption ensures that at the completion of the first step of RRE, tags placed in the vicinity of at least two readers store the highest number of tags covered by the readers. For example, in Figure 4.1, the count of tag T_3 is 4, from reader R_2. However, if we assume that initially readers are not aware of the identity of adjacent tags and RCA needs to be run just before RRE, the following scenario can occur (see Figure 4.1 for illustration). Since R_4 only covers two tags, whereas R_2 covers four, R_4 will complete RCA before R_2 and also the first step of RRE. Then, R_4, upon discovering itself to be the locker of T_3 and T_4, will also decide to stay active, even though it is redundant.

In order to solve this problem, we require active readers to maintain their list of locked tags and to listen for tag answers to queries initiated by other readers. When a reader, R, hears such a message, of format R_x, T_y, c (see Fig. 4.3 line 11), saying that the locker of tag T_y is R_x with a tag count c, if c is larger than its own tag count, the reader R removes tag T_y from its list of locked tags. When the list is empty, the reader becomes redundant and can

be safely turned off. Theorem 3.3.1 (see Section 3.3) proves that if such a scenario occurs, a reply of content R_x, T_y, c *will* be received by R for all tags T_y covered by readers with a larger tag count. Using the example in Figure 4.1, if R_4 has T_3 and T_4 in its list of locked tags at the completion of RRE, during R_2's execution of the first step of RRE, R_2 will find at least one time frame during the nEpochs epochs, when no other reader is transmitting. Thus, R_4 will overhear the replies of T_3 and T_4.

System Adaptivity The above description of RRE assumes a static system. However, in a real system, tags and readers may fail while new ones may be randomly deployed. We present a simple extension of RRE, that maintains the invariant of having at least one active reader for each covered tag, when new tags are deployed, potentially in areas covered only by redundant (inactive) readers and when active readers fail, leaving tags covered only by redundant readers, uncovered. For this, we need to periodically re-activate redundant readers and execute RRE on all the readers. Then, the following problem occurs, illustrated using Figure 4.1. If the only active reader, R_2, fails, when R_1, R_3 and R_4 become re-activated, tags $T_1, .., T_4$ have the associated count 4. Thus, the readers again discover their supposed redundancy and become inactive.

We solve this problem by requiring each active reader to periodically, every T time units, execute RCA to detect all its covered tags, including newly deployed ones and set to 0 the counter of each of its covered tags before executing RRE. A tag will agree to set its counter to a smaller value, 0, since 0 is a control value (a reader covering no tags will not issue such a write command). Of course, this can lead to a situation where R_2 sets the counter of its tags to 0 and then to 4, followed by the activation of R_4, R_4's setting the counter of its tags to 0 and then to 2. Then, R_4 and R_2 might both decide to stay active. For this, we have two observations. First, such a scenario will not leave blind points. Second, a solution to this problem would be to set the period T of a reader to be inversely proportional to the number of covered tags. Then, R_2 will execute this procedure more often than R_4, making R_4 discover its redundancy.

4.4 Analysis

Since the number of tags covered by a reader is not known before running RCA, accurately evaluating the time necessary for RCA to complete is difficult. Even though the duration of the first step of RRE is fixed, $\log \psi$ time epochs, the second step of RRE may start at different times even for readers that have started RCA simultaneously. The question is then if, due to the lack of synchronization among readers, RRE can leave uncovered tags. We define the following safety property which should hold for any distributed algorithm for the redundant-reader elimination problem and prove that RRE satisfies it.

Safety. An algorithm for the redundant-reader elimination problem is said to be safe, if it will not turn off readers that cover tags not covered by active readers.

Theorem. RRE is safe.

Proof Let us assume that a tag $\mathtt{T}_1$ is situated inside the interrogation zones of two readers, $\mathtt{R}_1$ and $\mathtt{R}_2$. Furthermore, $\mathtt{R}_1$ covers fewer tags than $\mathtt{R}_2$. Then, it is likely for $\mathtt{R}_1$ to start the second step of RRE before $\mathtt{R}_2$ has succeeded writing its tag count on its covered tags. Then, both $\mathtt{R}_1$ and $\mathtt{R}_2$ will believe to be the locker of $\mathtt{T}_1$. However, $\mathtt{T}_1$ will not be left uncovered, since both $\mathtt{R}_1$ and $\mathtt{R}_2$ are required to stay active. This will only decrease the number of redundant readers able to be simultaneously deactivated. ■

Complexity of RRE. $\mathtt{T}_{\mathtt{RRE}} = \mathtt{O}(\gamma \log \beta \log \psi)$.

Proof The complexity of RCA, is $\mathtt{O}(\gamma \log \beta \log \psi)$ (see Section 3.3). The first step of RRE, where each reader sends a write command to all its tags, takes $e \log \psi$ epochs. The second step, where readers send queries to each of their tags, takes $\gamma \mathtt{e} \log \psi$ epochs. Thus, $\mathtt{T}_{\mathtt{RRE}} = \mathtt{O}(\gamma \log \beta \log \psi)$. ■

4.5 Simulation Results

All our experiments are performed by randomly (uniformly) deploying tags and readers in a $1000 \times 1000m^2$ square. We evaluate the efficiency of RRE in terms of the number

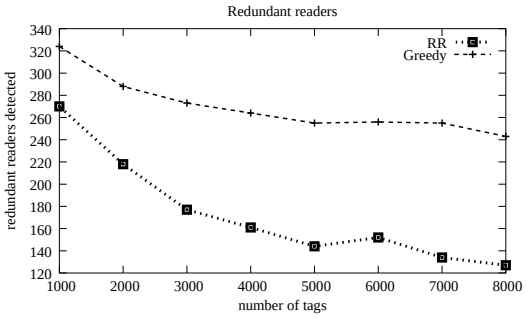


Fig. 4.4. Number of redundant readers discovered by RRE and Greedy when the number of tags randomly deployed increases from 1000 to 8000. The number of readers is constant, 500, throughout this experiment.

of redundant readers detected. We measure the performance of our redundant-reader detection algorithm and of a centralized greedy approximation algorithm of the redundant-reader problem, in terms of the number of readers able to be turned off simultaneously. The centralized greedy algorithm, GREEDY, sequentially selects the unvisited reader with the highest density of covered, unvisited tags. It then marks the selected reader and its covered tags as visited. GREEDY stops when there are no more unvisited tags. The set of selected readers will be active, while the remaining ones can be safely deactivated. GREEDY is correct, in the sense that deactivated readers will not leave tags uncovered.

In the first experiment we randomly place 500 readers and between 1000 and 8000 tags in the deployment 1000 $\times$ 1000 m^2 square. We measure the number of redundant readers discovered by RRE and GREEDY. Figure 4.4 shows the results of this experiment. For

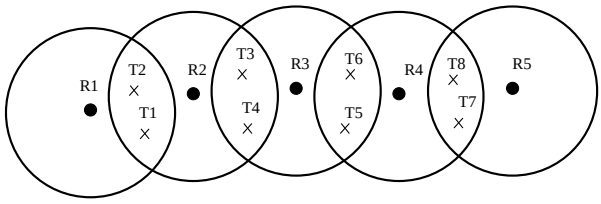


Fig. 4.5. Difficulty of consistently breaking ties.

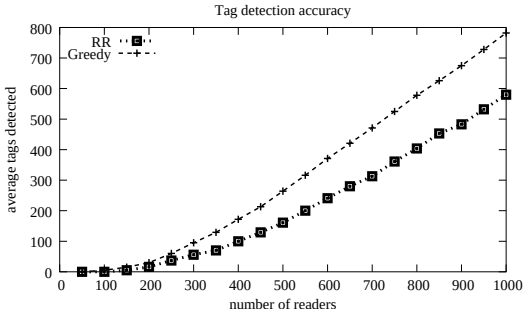


Fig. 4.6. Number of redundant readers discovered by RRE and Greedy when the number of readers randomly deployed increases from 50 to 1000, for a total of 4000 tags.

smaller numbers of tags deployed, RRE is reasonably close to GREEDY, by discovering 83% of the redundant readers discovered by GREEDY. As the number of tags increases, RRE discovers however only half of the redundant readers of GREEDY. Both GREEDY and RRE discover less redundant readers as the number of deployed tags increases. Both algorithms base their decision on the number of tags covered by readers. By increasing the tag density, the distribution of tags per reader becomes more uniform, making it more difficult to choose good, active readers. However, the decrease is more acute for RRE,

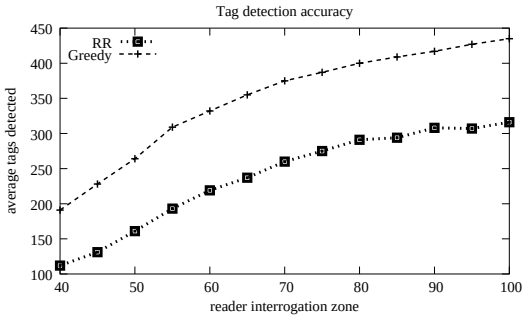


Fig. 4.7. Number of redundant readers discovered by RRE and Greedy when the interrogation radius of readers increases from 40 to 100m. The number of readers is 500 and the number of tags is 4000, for the entire duration of the experiment.

since in scenarios where readers whose interrogation zones overlap cover equal numbers of tags, consistently breaking ties becomes a difficult problem. We illustrate such a scenario in Figure 4.5, where each of readers R_2, R_3 and R_4 covers four tags. While the optimal solution requires only R_2 and R_4 to be active, we can imagine a run of RRE where R_4 locks $T_5, .., T_7$, R_3 locks T_3 and T_4 and R_2 locks T_1 and T_2, effectively requiring all three readers to be active. The example can be easily extended, and one can see that in the worst case RRE can require $2r - 1$ active readers, where r would be sufficient.

The second experiment compares the performance of RRE and GREEDY when the number of randomly deployed readers increases from 50 to 1000, when the total number of tags is 4000. Figure 4.6 shows the results of this experiment. For scarce deployment of readers, very few of the readers are redundant. As their density increases, however, so does the number of redundant readers. For example, for 1000 readers, GREEDY discovers almost 800 to be redundant. While initially RRE is very accurate, as the number of readers increases, RRE discovers less redundant readers. From 500 to 1000 readers, RRE consistently discovers 20% less redundant readers than GREEDY. This is again due to the difficulty of breaking ties in RR. As the number of deployed readers increases, the number of readers whose interrogation zones overlap, also increases, leading to more contentions.

The last experiment measures the dependency between the number of redundant readers discovered by RRE and GREEDY and the interrogation zones of readers. We randomly deploy 500 readers and 4000 tags, and increase the interrogation radius of readers from 40 to 100m. Figure 4.7 shows that as expected, with the increase in the interrogation radius of readers, both RRE and GREEDY discover an increasing number of redundant readers. This is because active readers cover larger areas, effectively necessitating fewer active readers to cover all the tags. Note that while RRE discovers less redundant readers than GREEDY, the difference is almost constant for smaller interrogation zones. Due to an increase in the number of interrogation zone overlappings, leading to an increased difficulty of breaking ties, the difference between GREEDY and RRE increases slightly for large interrogation zones.

4.6 Conclusions

In this chapter we address the problem of detecting and temporarily turning off redundant readers. We define redundancy in terms of discrete sets of points, tags, and prove that the optimization version of the problem is NP-complete. We present a distributed and localized approximation algorithm, RRE, for the redundant reader elimination problem, based on a randomized querying technique. Our simulations show that our redundant-reader elimination heuristic performs close to a centralized greedy approximation of the optimum. We note that while our solution scales well with an increase in the number of readers and with the increase in their interrogation radius, its efficiency degrades quicker for increasing numbers of tags covered. This is due to the difficulty of consistently breaking ties at tags, when readers in their vicinity have the same tag count. Part of our future work is to further explore this issue and improve the performance of RRE.

5. CONCLUSIONS

In this thesis we have studied various facets of coverage of wireless sensor and RFID systems. In the context of wireless sensor networks we have

- defined the coverage-boundary of wireless sensor networks and reduced the problem to determining the Voronoi cell of each sensor
- reduced the detection of redundant sensors, whose deactivation will not reduce the initial coverage of the sensor network, to the computation and maintenance of the Voronoi neighbors of each sensor
- provided a distributed hash table for general purpose ad-hoc networks based on Voronoi diagrams
- proposed distributed algorithms for maintaining the solution to the above problems when topological changes occur

Moreover, we have extended the redundancy definition to wireless RFID reader networks and have provided a distributed and localized approximation for the optimization problem proved to be NP-hard. Our algorithm also provides an efficient solution for an important problem of RFID systems, namely the reader collision problem.

The solutions presented for sensor networks rely on the ability of distributively computing and maintaining the Voronoi cell of each sensor. Since Voronoi neighbors may be $O(n)$ hops away, building the initial Voronoi cell requires the collection of global information for each sensor. Thus, alternative solutions to this resource consuming operation need to be devised. One possibility is to use the central collection point to compute the initial solution and distribute to individual sensors their corresponding Voronoi cell. However, even the overhead associated with this solution can be significant. Another way could be

to construct an initially imprecise solution, using only local neighborhood information. Later, information about locally discovered Voronoi neighbors can be propagated to the affected sensors. It would be interesting to explore the accuracy of this solution and its impact on the coverage-boundary computation and the redundancy elimination problem. Clustering can be another approach for this problem. Using existing distributed algorithms for clustering networks, certain sensors could be given a special role of cluster head, in charge of all the sensors within its cluster. Cluster head sensors would be responsible with computing the Voronoi diagram of their clusters and exchange this information with the neighboring cluster heads.

Hybrid solutions, between the clustering and centralized scheme or between the clustering and imprecise scheme might be interesting to explore. In the former, cluster heads aggregate the information of their cluster by computing the Voronoi diagram of the contained sensors. This information is then propagated to the central server, which in turn, further aggregates into a global Voronoi diagram and signals inaccuracies to cluster heads. In the latter hybrid scheme, cluster heads exchange Voronoi information only with their neighboring cluster heads, or cluster heads within a bounded number of hops. With the additional information, cluster heads recompute the Voronoi diagram and signal newly discovered Voronoi neighbors to the cluster heads in charge of them. The advantage of this schemes resides in the active participation of only a limited number of sensors, the cluster heads.

Another problem consists in the efficient and correct distributed maintenance of the Voronoi diagram for moving devices. Our solution is only an approximation and might not correctly maintain the Voronoi cells for rapid device movements. While solutions for this problem have been proposed, their centralization restricts their applicability.

Our randomized, distributed and localized approximation algorithm for the redundant reader elimination optimization problem further opens several research issues. First, as observed in Section 4.5, as the number of distributed tags increases, the number of redundant sensors discovered by RRE decreases relative to the centralized Greedy algorithm. Then, an interesting problem, deserving further attention, is to improve the performance

of RRE, assuming the existence of writable RFID tags. Another avenue for research is to develop a distributed algorithm for the redundancy-elimination problem, assuming only passive, read-only RFID tags. Due to price constraints, read-only tags will likely be more widely accepted and deployed, making such algorithms necessary. Finally, an important assumption behind RRE is that RFID readers are not able to directly communicate and relay information. Removing this assumption, an algorithm similar to the one used in safely turning off redundant sensors could be employed. It would be interesting to see if a variant of Luby's distributed MIS approximation algorithm could improve the efficiency, while using solely read-only tags.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Battery Assisted Passive Microwave RFID tags. URL `http://www.alientechnology.com/products/rfidbattery/bap_tags.php`.

[2] SkyeTek. http://www.skyetek.com/readers_Mini.html, January 2004.

[3] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of the First ACM Wireless Sensor Networks and Applications(WSNA)*, pages 32–41. ACM Press, 2002.

[4] S. E. Sarma, S. A. Weis, and D. W. Engels. RFID systems and security and privacy implications. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 454–469. Springer-Verlag, 2003.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[6] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[7] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *LNCS*, 2218:329–351, 2001.

[8] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley, April 2001.

[9] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: A geographic hash table for data-centric storage in sensornets. In *Proceedings of the First ACM Wireless Sensor Networks and Applications (WSNA)*, September 2002.

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

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

[12] L. Hu and D. Evans. Localization for mobile sensor networks. In *Proceedings of MobiCom*, pages 45–57. ACM Press, 2004.

[13] D. Niculescu and B. Nath. Vor base stations for indoor 802.11 positioning. In *Proceedings of MobiCom*, pages 58–69. ACM Press, 2004.

[14] A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of MobiCom*, pages 70–84. ACM Press, 2004.

[15] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proceedings of MobiHoc*, pages 45–55. ACM Press, 2001.

[16] Y.-B. K. and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of MobiCom*, pages 66–75, 1998.

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

[18] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A distance routing effect algorithm for mobility (DREAM). In *Proceedings of MobiCom*, 1998.

[19] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of MobiCom*, pages 120–130. ACM Press, 2000.

[20] Z. J. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228–240, 1999.

[21] 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.

[22] P. Bose and P. Morin. Online routing in triangulations. In *Proceedings of International Symposium on Algorithms and Computation (ISAAC)*, 1999.

[23] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2(2):55–80, 1992.

[24] G. Albers, L. J. Guibas, J. S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points. *International Journal on Computational Geometry and Applications*, 8(3):365–380, 1998.

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

[26] J. O'Rourke. Open problem from art gallery solved. *International Journal on Computational Geometry and Applications*, 2:215–217, June 1992.

[27] K. Lieska, E. Laitinen, and J. Lahteenmaki. Radio coverage optimization using genetic algorithms. In *IEEE International Symposium On Personal, Indoor and Mobile Radio Communications (PIMRC)*, 1998.

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

[29] 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 Foundations Of Computer Science (FOCS)*, pages 691–700, 2002.

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

[31] S. A. Borbash and E. Jennings. Distributed topology control algorithm for multihop wireless networks. In *Proceeding of World Congress on Computational Intelligence (WCCI)*, May 2002.

[32] H. Zhang and J. 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.

[33] F. Ye, G. Zhong, S. Lu, and L. Zhang. PEAS: A robust energy conserving protocol for long-lived sensor networks. In *23rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[34] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *IEEE International Conference on Communications (ICC)*, 2001.

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

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

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

[38] S. Shakkottai, R. Srikant, and N. Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In *Proceedings of IEEE INFOCOM*, April 2003.

[39] S.E. Sarma. Towards the five-cent tag. Technical Report MIT-AUTOID-WH-006, MIT Auto ID Center, 2001.

[40] E. Kalischnig. RFID: Making sense of sensor-based technology. Manufacturing and Logistics IT, July 2004.

[41] A. Bednarz. Wireless technology reshapes retailers. Network World, 12 August 2002.

[42] SmartCode. http://www.smartcodecorp.com/.

[43] D. W. Engels and S. E. Sarma. The reader collision problem. In *IEEE International Conference on Systems, Man and Cybernetics*, 2002.

[44] H. E. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Ruhl, and M. F. Kaashoek. Performance evaluation of the Orca shared-object system. *ACM Transactions on Computating Systems*, 16(1):1–40, 1998.

[45] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[46] J. Waldrop, D.W. Engels, and S.E. Sarma. Colorwave: a MAC for RFID reader networks. In *Wireless Communications and Networking (WCNC)*, 2003.

[47] S. Weis, S. E. Sarma, R. L. Rivest, and D. W. Engels. Security and privacy aspects of low-cost radio frequency identification systems. In *Security in Pervasive Computing*, pages 201–212. Lecture Notes in Computer Science, 2003.

[48] A. Juels, R. L. Rivest, and M. Szydlo. The blocker tag: selective blocking of RFID tags for consumer privacy. In *Proceedings of ACM Computer and Communications Security (CCS)*, pages 103–111. ACM Press, 2003.

[49] N. Abramson. The ALOHA system: Another alternative for computer communications. In *AFIPS Conference Proceedings, Fall Joint Computer Conf*, 1970.

[50] *IEEE Std 802.11b-1999*. 1999. URL `http://standards.ieee.org/`.

[51] P. Karn. MACA a new channel access method for packet radio. In *ARRL/CRRL Amatuer Radio 9th Computer Networking Conference*, 1990.

[52] Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Institute of Electrical and Electronics Engineers, 1996.

[53] B. Carbunar, A. Grama, J. Vitek, and O. Carbunar. Coverage-preserving redundancy elimination in sensor networks. In *Proceedings of IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, 2004.

[54] R. Fowler, M. Paterson, and S. Tanimoto. Optimal packing and covering in the plane are NP complete. *Information Processing Letters*, 12(3):133–137, 1981.

VITA

VITA

Bogdan Cărbunar was born on December 2nd 1975 in Oradea, Romania, of Mariana and Octavian Cărbunar. He did his studies in Bucharest, where he received a B.S. in computer science from ”Politehnica University” in 1999. Since then, he has been a Ph.D. student at Purdue University.