Wireless Netw
DOI 10.1007/s11276-010-0262-2

Sleep scheduling with expected common coverage in wireless sensor networks

Eyuphan Bulut · Ibrahim Korpeoglu

Abstract Sleep scheduling, which is putting some sensor nodes into sleep mode without harming network functionality, is a common method to reduce energy consumption in dense wireless sensor networks. This paper proposes a distributed and energy efficient sleep scheduling and routing scheme that can be used to extend the lifetime of a sensor network while maintaining a user defined coverage and connectivity. The scheme can activate and deactivate the three basic units of a sensor node (sensing, processing, and communication units) independently. The paper also provides a probabilistic method to estimate how much the sensing area of a node is covered by other active nodes in its neighborhood. The method is utilized by the proposed scheduling and routing scheme to reduce the control message overhead while deciding the next modes (full-active, semi-active, inactive/sleeping) of sensor nodes. We evaluated our estimation method and scheduling scheme via simulation experiments and compared our scheme also with another scheme. The results validate our probabilistic method for coverage estimation and show that our sleep scheduling and routing scheme can significantly increase the network lifetime while keeping the message complexity low and preserving both connectivity and coverage.

Keywords Wireless ad hoc and sensor networks · Sleep scheduling · Distributed algorithms · Network protocols

1 Introduction

In recent years, advances in wireless communications and electronics have enabled the development of low-power and small size sensor nodes. A *Wireless Sensor Network* (WSN) consists of a large number of these sensor nodes deployed in a geographic area. Wireless sensor networks are utilized in a wide range of applications including battlefield surveillance, smart home environments, habitat exploration of animals and vehicle tracking.

Each sensor node in a WSN has three basic units; a sensing unit, a processing unit and a communication unit. The sensing unit can sense various phenomena including light, temperature, sound and motion around its location [1]; the processing unit can process and packetize the sensed data; and the transmission unit can send the packetized data to a *base station* (also called *sink* node) possibly via multihop routing.

In general, a sensor node can be considered to have two associated ranges: a transmission range (R_t) and a sensing range (R_s). As a simple and quite common model, a sensor node can be assumed to detect every event happening within a circular area with radius R_s around itself. Similarly, a sensor node can be assumed to communicate with all other sensor nodes located within the circular region with radius R_t around itself.

This work is supported in part by European Union FP7 Framework Program FIRESENSE Project 244088.
This work has been done while the first author (Eyuphan Bulut) was an M.S. student in the Department of Computer Engineering of Bilkent University.

E. Bulut
Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
e-mail: bulute@cs.rpi.edu

I. Korpeoglu (✉)
Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey
e-mail: korpe@cs.bilkent.edu.tr

Published online: 28 July 2010

In a sensor node, energy is primarily consumed by its three basic units. It is usually observed and assumed that the most energy consuming operations are data receiving and data sending which are provided by the communication unit. Energy consumption in sensing unit is usually assumed to be less than these operations. However, in some studies such as [2], it is assumed that energy dissipated to sense a bit is approximately equal to the energy dissipated to receive a bit. Processing operations, on the other hand, are assumed to be consuming very little energy compared to sensing and communication operations. Therefore it is important to be able to put sensing and communication units into sleep mode whenever possible.

There are several ways of reducing the energy consumption in a sensor network in order to increase the network lifetime. In sufficiently dense networks, a common technique is to put some sensor nodes into sleep and use only a necessary set of active nodes for sensing and communication. This technique is called *sleep scheduling* or *density control*. A sleep schedule has to provide an even distribution of energy depletion among sensor nodes so that the network can function for a long time. Using only a required set of nodes as active, can also reduce redundant network traffic, decrease packet forwarding delay and help in avoiding packet collisions.

While putting nodes into sleep or active mode, a sleep scheduling algorithm should be able to maintain *connectivity* and *coverage*. A sensor network is connected if every functioning node in the network can reach the sink via one or multiple hops. Coverage is defined as the area that can be monitored by the active sensor nodes which can reach the sink. Both connectivity and coverage are important objectives to meet to properly monitor a given region. While deciding to put a sensor node into sleep, it is important to know if the area sensed by the sensor node can be sufficiently covered by some active neighboring nodes and if the sensor node is crucial for the connectivity of the network.

In this paper, we first provide a probabilistic and analytical method to estimate the amount of overlapping sensing coverage between a node and its neighbors. The method helps in estimating whether a node can be put into sleep without violating desired coverage. The method assumes that a large number of sensor nodes are deployed uniformly and randomly to target region. Based on this assumption and by just knowing the *number* of neighbors of a node, the expected amount of overlapping coverage is computed, without requiring to know the exact locations of nodes. This coverage estimation method is the first main contribution of the paper.

We then propose a distributed sleep scheduling and routing scheme that also utilizes our coverage estimation method. Our scheme assumes a static sensor network where nodes are densely, randomly and uniformly distributed. It works with local interactions only, reduces the energy consumption in the network, and works with low control messaging overhead while each node is learning about the status of the neighborhood nodes and deciding its mode for the next round. The routing scheme is a tree-based routing scheme that can adapt to node-state changes and to node failures due to lack of energy. Our sleep scheduling scheme considers communication and sensing units of a sensor node separately and is able to put only one unit into sleep instead of putting all units into sleep together. The scheme also can maintain a desired coverage and connectivity. This combined sleep scheduling and routing scheme is the second main contribution of our paper.

The remaining of the paper is organized as follows: In Sect. 2, we discuss the related work in comparison with our work here. In Sect. 3, we provide an analysis and method for coverage estimation of a node's sensing area by its neighbors. In Sect. 4, we introduce and detail our combined sleep scheduling and routing scheme. In Sect. 5, we present our simulation experiments for evaluating our scheme and discuss the results. Finally, in Sect. 6 we give our conclusions.

2 Related work

The papers [3] and [4] give a detailed description and comparison of the most recent energy saving algorithms based on sleep scheduling technique. An important aspect that distinguishes the proposed algorithms is whether they are centralized or distributed. Usually, centralized algorithms can provide more accurate results about which nodes should be sleeping, but they usually suffer from high messaging overhead and difficulty in quickly adapting to changing conditions. Distributed algorithms, on the other hand, have less messaging cost, can adapt to dynamic conditions better, are scalable, but it is more difficult to obtain optimal results with them.

There are various centralized sleep scheduling techniques proposed. Two similar solutions are [5] and [6]. They work in a dense deployment and aim to provide energy efficiency while preserving coverage. They are based on dividing the nodes into disjoint sets, so that each set can independently accomplish monitoring the area, and the sets are activated periodically while the nodes in other sets are put into low-energy mode. There are also sleep scheduling schemes based on ILP techniques, basing their decisions on remaining energy levels of nodes ([7] and [8]). But these solutions can not scale well for very large networks. The works of [9] and [10] also apply centralized and greedy approaches. While deciding which nodes should stay active, [9] considers the nodes with better coverage first, whereas [10] considers the nodes with more remaining energy first. Our scheme in this paper is a distributed one, hence differing from these works in this aspect.

There are also various sleep scheduling schemes following distributed approach. GAF [11] divides a region into equal-sized grid cells and tries to leave only one node active in each cell. In PEAS [12, 13], a node decides to go into sleep mode if there is an active neighbor in its probing range. Otherwise, it stays active. In SPAN [14], all nodes are classified as either a coordinator or a non-coordinator such that at the end every node is in the radio range of at least one coordinator. Only coordinators forward traffic. These studies do not focus on preserving coverage, and therefore they are different from our work here.

There are also some protocols that consider maintaining coverage. [15] shows a way of finding the overlapping sensing area between a node and its neighbors. However, it only considers 1-hop neighbors. But, 1-hop neighbors may not include all sensor nodes that might cover the same area. We also consider 2-hop neighbors in this paper. Additionally, even though [15] guarantees coverage, it does not guarantee connectivity.

There are some algorithms, OGDC [16] and CCP [17], that consider both coverage and connectivity, as we do in this paper. In [16], Zhang and Hou prove that coverage implies connectivity if the ratio between the transmission range and sensing range is at least two. Depending on this, they propose Optimal Geographic Density Control (OGDC) algorithm to maximize the number of sleeping sensor nodes while maintaining coverage. A sensor node is active only in the case it minimizes the overlapping area with the existing active sensor nodes and it covers an intersection point of two sensors. A sensor node decides this by using its own location and the location of other active nodes. In [17], Wang et al. propose coverage and connectivity configuration protocol (CCP) which tries to maximize the number of sleeping nodes while maintaining k-coverage and k-connectivity. Here, k-coverage means each point in the monitoring area of the sensor network is sensed by at least k different nodes of the network. The authors prove that k-coverage implies k-connectivity and to decide k-coverage, a node only needs to check whether the intersection points inside its sensing area are k-covered. Similar to OGDC, CCP assumes the transmission range is at least twice the sensing range. But if it is not the case, it combines its algorithm with SPAN so that SPAN can control connectivity. In this case, a node decides to sleep if it satisfies the eligibility rules in both schemes. Otherwise it stays active.

Our work here is different from the above two schemes and others in many aspects. Below we summarize the main features and contributions of our work and how it is different from the similar work described in this section.

- A probabilistic and analytical method is proposed to estimate the overlapping sensing coverage between a node and its neighbors. The method is then used by the proposed sleep scheduling scheme to reduce the number of control messages required to learn the status of neighbors.
- A combined sleep scheduling and routing scheme is proposed. Hence we consider sleep scheduling and routing together. Most other works consider routing and sleep scheduling independently from each other, which may cause extra overhead.
- Both the sensing coverage and connectivity of the network are maintained for a wide range of transmission range (R_t) and sensing range (R_s) values. Previous work usually considers them one at a time, or for restricted values of R_t/R_s.
- Different units of a sensor node are considered separately for switching on and off. We define three modes of operation: full-active (both sensing unit and communication unit is on), semi-active (sensing unit is off, communication unit is on) and inactive (both sensing and communication unit is off). Previous work usually does not consider the units separately and defines just two modes of operation: active or sleep. Hence, our scheme is multi-mode.
- The desired coverage is a parameter of the proposed scheme. In this way, the protocol can work to maintain a desired partial coverage (let say 70% of the sensing area of each sensor node has to be covered). This is different from many previous studies which consider overlapping coverage amount as a boolean value.

3 Expected common coverage analysis

In this section we provide an analysis and method about how to find the expected overlap between a node's sensing area and its neighbors' sensing areas. Then this method is used in our combined sleep scheduling and routing protocol. However, the coverage estimation analysis and method we propose may find its place in some other applications as well. For example, it can be adapted to estimate if a point in a region is k-covered or not.

A sensor node is *coverage redundant* (or coverage eligible) if its sensing area is covered (fully or partially, depending on the requirements) by the sensing areas of some other active nodes. Many sleep scheduling protocols consider only the 1-hop communication neighbors (i.e. nodes in the transmission range) to check whether they cover the sensing area of the node. There may be, however, nodes that are not reachable in 1-hop, but still may have overlapping sensing coverage with the node.

Consider the example illustrated in Fig. 1. The nodes B, C, and D are 1-hop neighbors of the node A, and the nodes E, F, G, H are other nodes which have common

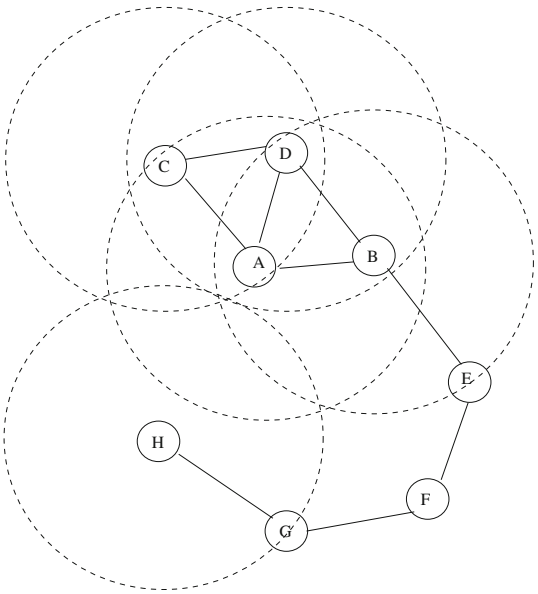


Fig. 1 Node A's sensing area is totally covered by not only the 1-hop nodes of A but also another node H

sensing areas with node A. If node A only considers its 1-hop neighbors while deciding whether it is sensing area is covered by other nodes (that is, it is coverage eligible or not), it decides to be non-eligible, since the sensing area of node A is not totally covered by 1-hop neighbors. However, if other closer nodes to node A could be considered, the sensing area of node A is totally covered by other nodes. Hence, we should also consider the effect of other nodes which are closer than $2R_s$ to a sensor node, while applying coverage check on the node.

When R_t/R_s ratio decreases, we encounter such cases more frequently. Therefore, a general coverage check algorithm should work for a wide range of R_t/R_s values. In today's sensor node technology, we may see different values for this ratio. It mostly lies in the range [1/2, 3] [18, 19].

On the other hand, even a node can not communicate with another node having an overlapping sensing area, this may not always have a critical effect on coverage check. Moreover, it may be too costly to learn about such multi-hop neighbors and constantly maintain their current status information. In Fig. 1, for example, even though node H is not far away from node A, node A can communicate with node H over 5 hops. As the hop count increases to reach such nodes, messaging overhead to maintain up-to-date information about these nodes increases as well. Therefore, there is a tradeoff between a good coverage check and control messaging overhead.

To find out how much other nodes cover the sensing area of a node, we may collect precise information (i.e. location, status) about the other nodes and then use geometric computations to find out the overlap. This may be, however, costly in terms processing and communication. Another alternative is using probabilistic models. Assuming the nodes are randomly deployed with uniform distribution, we can derive a probabilistic model which gives the expected coverage of a node's sensing area using only the *number* of other nodes that may have common coverage with this node and the R_t/R_s value.

Next, we are proposing such a probabilistic and analytical method to compute the expected coverage. Here, note that, the analysis is based on a network model where nodes are identical and uniformly distributed. Hence it can be applicable for certain applications and scenarios where a large number of nodes are expected to be randomly and uniformly distributed. The method needs to be modified for networks consisting of heterogeneous and non-uniformly distributed nodes. We leave this out of the scope of this paper.

3.1 Expected common sensing coverage with 1-hop neighbors

In this section we derive a model to find out the expected common sensing coverage (i.e. overlap) between a node and its 1-hop communication neighbors. The expected common sensing coverage (which is a value between 0 and 1) depends on the number of 1-hop neighbors of the node (n), the transmission range of the nodes (R_t), and the sensing range of the nodes (R_s).

Assume we have a sensor node of interest located at point O. Let X denote a random variable indicating the distance of the sensor node to a point in its sensing range. Possible values x of X are $0 \leq x \leq R_s$. The probability density function for X is $f_X(x) = 2x/{R_s}^2$.

Assume that the probability of a point P that is inside the sensing area of the node and that is x m away from the node is covered by a neighbor of the sensor node is $p(x)$. Obviously, this probability is not same for all points and it depends on the distance x of the point to the sensor node. When there are n neighbors of the node, then the probability that a point is covered by any of these neighbors is $1 - (1 - p(x))^n$. If we integrate $p(x)$ over the sensing area of the node, we can find out the expected common coverage (overlap) between a node's sensing area and its n 1-hop neighbors' sensing areas.

Consider the Fig. 2. We have the sensor node located at point O. We want to find $p(x)$ of point P. For point P to be covered by a neighbor of the sensor node, there should be a neighbor inside the shaded region. In other words, a neighbor which is not more than R_s distant from point P and which is inside R_t of node should exist. Therefore, $p(x)$ of point P is equal to the ratio of shaded area in Fig. 2 to whole communication area of the sensor node, i.e., πR_t^2.

To calculate the area of the shaded region, we first place our model into an $x - y$ coordinate plane as it is shown in Fig. 3a. Then we calculate the area of the shaded region

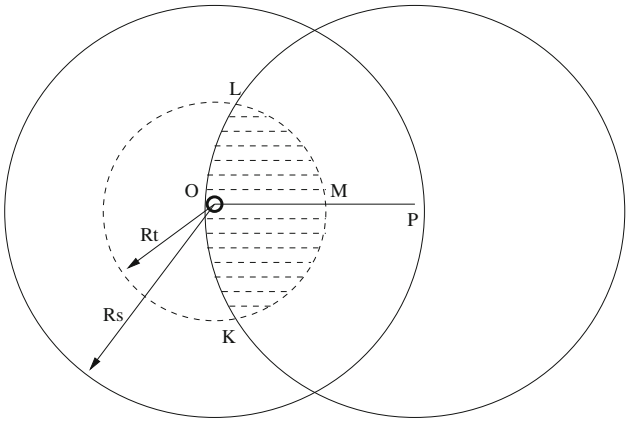


Fig. 2 Probability that a point P inside the sensing area is covered by a neighbor of the node is proportional to the shaded area

using the integral of the difference of circle equations enclosing it. Note that, in some cases (Fig. 3b) we first find the complementary region and subtract it from the whole communication area. For instance, we first find $A(\text{NKTL})$ and then subtract it from πR_t^2. These two different cases separate from each other when the height of the required region becomes R_t. Figure 3c shows this case. For the points which have longer distance to the center (i.e. sensor node) than this point the first approach is used, otherwise the second approach is applied.

In Fig. 3a, let x denote the distance between the sensor node and the point (i.e. $x = |\text{OP}|$). Note that, x is not the x-axis value anymore for this analysis. Let $|\text{OS}| = b$; then $|\text{PS}| = x - b$, and:

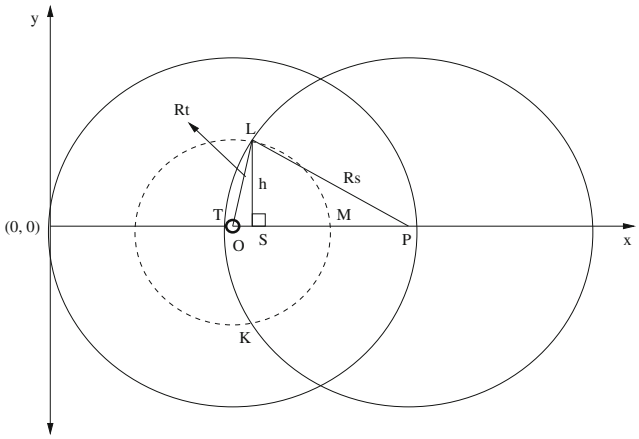
$$b = \frac{R_t^2 - R_s^2 + x^2}{2x}$$

$$h = \sqrt{R_t^2 - b^2}$$

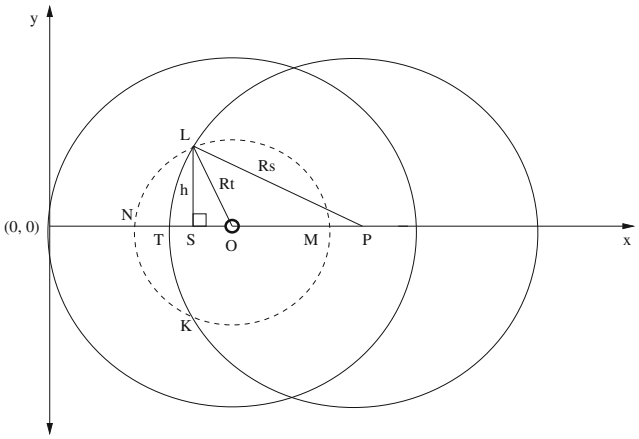
$$A(\text{TKML}) = 2 \int_{y=0}^{y=h} \left(\sqrt{R_t^2 - y^2} + \sqrt{R_s^2 - y^2} - x \right) dy$$

$$p_1(x) = \frac{A(\text{TKML})}{\pi R_t^2}$$

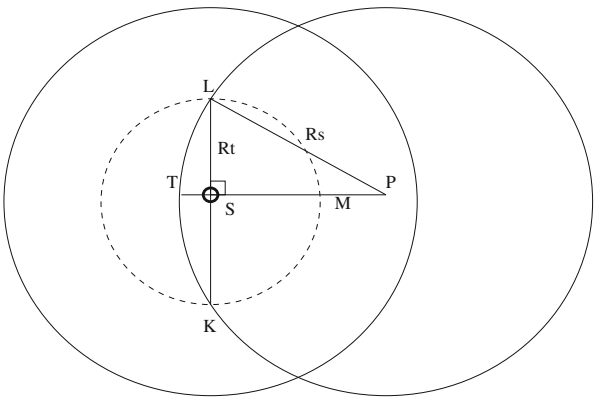
And in Fig. 3b, let $|\text{OS}| = b$ again. Then $|\text{PS}| = x + b$, and in a similar way:



(a) Projection on the right of center O.



(b) Projection on the left of center O.



(c) Projection on center O.

Fig. 3 Different projections of the height of the region

$$A(\mathrm{TKML}) = \pi R_t^2 - A(\mathrm{TKNL})$$

$$A(\mathrm{TKNL}) = 2\int_{y=0}^{y=h}\left(\sqrt{R_t^2 - y^2} - \sqrt{R_s^2 - y^2} + x\right)dy$$

$$p_2(x) = 1 - \frac{A(\mathrm{TKNL})}{\pi R_t^2}$$

The border value x_{border}, that separates these cases is equal to $\sqrt{R_s^2 - R_t^2}$. As a result, when $R_t < R_s$, the expected value of the probability ($E[p(X)]$) that a point inside the sensing area is covered by a neighbor is:

$$p = E[p(X)] = \int_{x=0}^{x=R_s} p(x) f_X(x)dx$$

$$p = E[p(X)] = \int_{x=0}^{x=R_s-R_t} (R_t/R_s)^2 f_X(x)dx + \int_{x=R_s-R_t}^{x=x_{border}} p_2(x) f_X(x)dx + \int_{x=x_{border}}^{x=R_s} p_1(x) f_X(x)dx \text{ where,}$$

$$f_X(x) = 2x/R_s^2$$

In other words, p expresses the expected common coverage between a node and one of its neighbors. When $R_t > R_s$ we can find p in a similar way. But there are two different calculations in this case. The border case, which happens when $R_t = R_s\sqrt{2}$, is illustrated in Fig. 4.

When $R_s \le R_t \le R_s\sqrt{2}$:

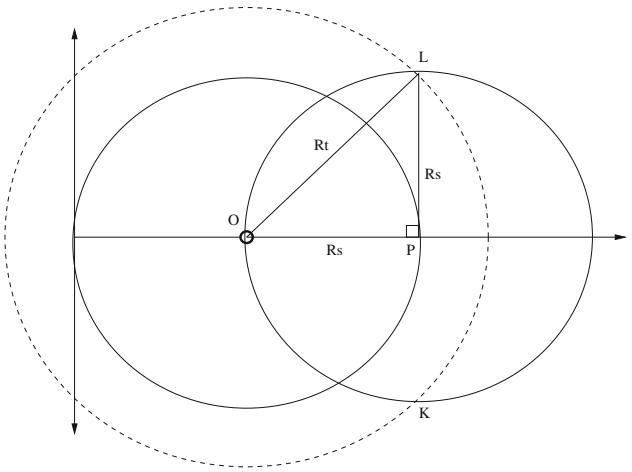


Fig. 4 Border case when $R_t > R_s$

$$p = \int_{x=0}^{x=R_t-R_s} (R_s/R_t)^2 f_X(x)dx + \int_{x=R_t-R_s}^{x=x_{border}} p_2(x) f_X(x)dx + \int_{x=x_{border}}^{x=R_s} p_1(x) f_X(x)dx$$

When $R_s\sqrt{2} \le R_t \le 2R_s$:

$$p = \int_{x=0}^{x=R_t-R_s} (R_s/R_t)^2 f_X(x)dx + \int_{x=R_t-R_s}^{x=R_s} p_2(x) f_X(x)dx$$

Now, let p_{n,d_1,d_2} denote the expected overlap of a node i's sensing area by n nodes, where these nodes have distance from the node in the interval $[d_1, d_2]$. And, let p_n denote the expected overlap by n 1-hop neighbors. Then, $p_n = p_{n,0,R_t}$. When $n = 1$, it is equal to p_{0,R_t} or simply p.

Assume there are n nodes in interval $[0, R_t]$ after deployment (i.e. n 1-hop neighbors). Then[1],

$$p_n = p_{n,0,R_t} = \int_{x=0}^{x=R_s} (1 - (1 - p(x))^n) f_X(x)dx$$

We did some simulation experiments to check the validity of our estimation method. In our simulation experiments, we created random multiple neighbors to a node within its transmission range and calculated the overlap of the node's sensing area by its neighbors. Besides, for each multiple neighbor count, the simulation is run 1000 times and the result is obtained as the average of them. The Fig. 5 show the comparison of simulation and analytical results for three different R_t/R_s values. As it can be seen from the figure, the analytical results completely overlap with the simulation results.

[1] In this analysis, we assumed that the links between the nodes are mostly reliable and there are no frequent link failures which may affect the data acquisition significantly. However, we can reflect the failure-prone nature of sensor node connections to this formula by multiplying n by λ (the probability that a connection between two connections may fail). Moreover, we can also include non-uniform node distribution in the network by updating the density function $f_X(x)$.

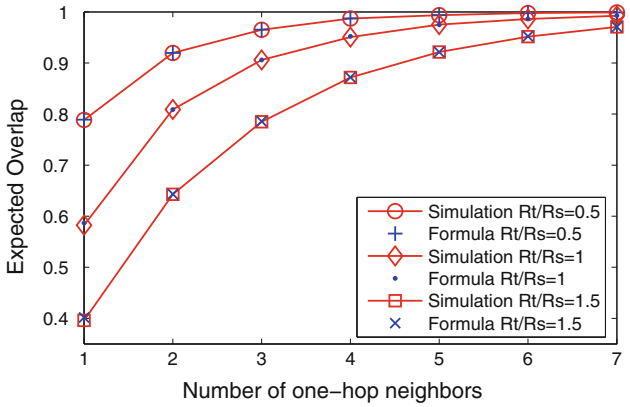


Fig. 5 Expected overlap values from simulation and formula with different R_t/R_s ratios

3.2 Expected common sensing coverage with 2-hop neighbors

A sensor node may have not only 1-hop communication neighbors but also two- or more hop communication neighbors which have overlapping sensing area with itself. Especially, when the R_t/R_s value gets smaller, multi-hop neighbors may create a significant overlap on the node's sensing area. To see the effect of multi-hop neighbors we need to calculate expected overlap as in 1-hop neighbor case. Here, we calculate the expected overlap with only 2-hop neighbors. Computing expected overlap for more than two hops is more complex and we leave it out of the scope of this paper. Additionally, as we discuss, using only 1-hop and 2-hop neighbors addresses a wide range of realistic scenarios.

Consider the Fig. 6. First, we will find the expected overlap by another node which has a distance to the node of interest in the interval $[R_t, 2R_t]$. As it is stated before, we denote this with $p_{R_t,2R_t}$. We can calculate this expected value similar to 1-hop case as follows:

$$p_{0,R_t} = \int_{x=0}^{x=R_s} \frac{A}{\pi R_t^2} f_X(x)dx$$

$$p_{0,2R_t} = \int_{x=0}^{x=R_s} \frac{A+B}{\pi(2R_t)^2} f_X(x)dx$$

$$p_{R_t,2R_t} = \int_{x=0}^{x=R_s} \frac{B}{\pi(2R_t)^2 - \pi {R_t}^2} f_X(x)dx = \frac{4p_{0,2R_t} - p_{0,R_t}}{3}$$

For example, when $R_t = R_s, p_{0,R_t} = 0.58$ and $p_{0,2R_t} = 0.25$, then $p_{R_t,2R_t} = (4(0.25) - 0.58)/3 = 0.13$.

Furthermore, if there are n such nodes (located at a distance in the interval $[R_t, 2R_t]$), the expected overlap by these nodes is:

$$p_{n,R_t,2R_t} = \int_{x=0}^{x=R_s} \left(1 - \left(1 - \frac{B}{3\pi R_t^2}\right)^n\right) f_X(x)dx$$

In the above, note that, we found the expected overlap by *possible* 2-hop neighbors, i.e. nodes that are located at a distance between R_t and $2R_t$. But for a node to be an *actual* 2-hop neighbor, being located at a distance between R_t and $2R_t$ is not sufficient because it should also be in the transmission range of a 1-hop neighbor of the node.

In Fig. 7, the existence of an actual 2-hop neighbor at point P is possible with the existence of at least one 1-hop neighbor in the shaded region. Following the same calculation approach used before, given that there are n_1 1-hop neighbors of the node, we conclude that the average probability (α_{n_1,R_t}) that there will be an actual 2-hop neighbor at any point in the range $[R_t, 2R_t]$ is:

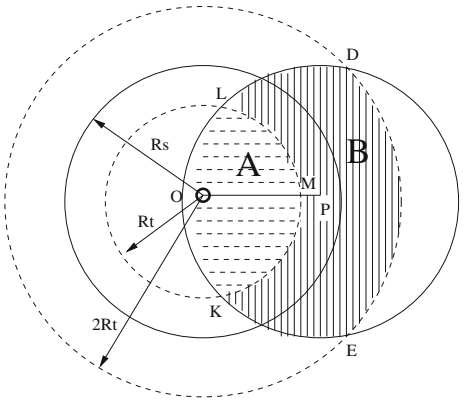


Fig. 6 Expected overlap by a node located at a point P which has a distance to the node in the interval $[R_t, 2R_t]$

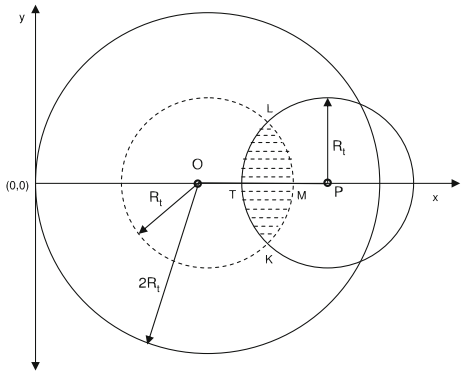


Fig. 7 The probability that there will be a 2-hop neighbor at point P is proportional with the probability that shaded area contains a 1-hop neighbor

$$\alpha_{n_1,R_t} = \int_{x=R_t}^{x=2R_t} \left(\frac{2x}{3R_t^2}(1-(1-\beta)^{n_1}) \right), \quad \text{where}$$

$$\beta = 2 \int_{y=0}^{y=\sqrt{R_t^2-\frac{x^2}{4}}} \left(2\sqrt{R_t^2-y^2} - x \right).$$

Let p_{n_1,n_2} denote the expected overlap by n_1 1-hop and n_2 2-hop neighbors. Then, to find the value of p_{n_1,n_2}, we again use the same probabilistic approach and combine the expected overlap of each hop. For instance, if there are n_1 1-hop neighbors and n_2 2-hop neighbors, the expected overlap by 1-hop and 2-hop neighbors (p_{n_1,n_2}) together can be calculated as:

$$\int_{x=0}^{x=R_s} \left(1-\left(1-\frac{A}{\pi R_t^2}\right)^{n_1}\left(1-\frac{B\alpha_{n_1,R_t}}{3\pi R_t^2}\right)^{n_2}\right) f_X(x)dx$$

Consequently, when a node knows the number of 1-hop and 2-hop neighbors, it can find the expected overlap of its sensing area by these nodes.

To check the validity of our analysis for 2-hop case, we also did simulations. When $R_s = R_t$, we found the expected overlap for different n_1 and n_2 values by using both analysis and simulation. Figure 8 shows the expected overlap for a node for all cases when $1 \le n_1 \le 3$ and $0 \le n_2 \le 6$. As Fig. 8 shows, the results obtained with analysis are matching with simulation results. For instance, if a node has one 1-hop neighbor and two 2-hop neighbors, then it can expect that 60.9% of its sensing area is covered by these neighbors. Here, note that, the overlap shows only a small increase with increasing n_2. Moreover, the effect of 2-hop neighbors on the overlap decreases when n_1 increases. These two observations are expected because 2-hop neighbors are located around 1-hop neighbors so that

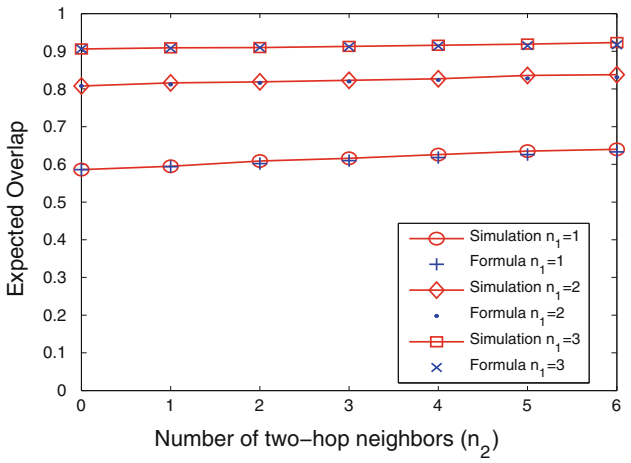


Fig. 8 Expected overlap values from simulation and formula with different n_1 and n_2 counts and when $R_t/R_s = 1$

the most part of the overlap resulting from 2-hop neighbors are already covered by 1-hop neighbors.

At the beginning of a network deployment, if we know the R_t and R_s values, we can calculate the expected overlap values for different 1- and 2-hop neighbor counts into a table (we call it *Expected Overlap Table*) where cell (i, j) of the table shows the expected overlap with i 1-hop and j 2-hop neighbors, and install this table in each sensor node. Then during network operation, sensor nodes can use the table to estimate the common coverage with their neighbors at any moment by just using their count.

4 Our combined sleep scheduling and routing scheme

In this section we introduce our combined sleep scheduling and routing scheme that works in a distributed and localized manner. It preserves both coverage and connectivity. It utilizes our probabilistic coverage estimation method, presented in the previous section, to reduce messaging overhead while collecting status information from neighboring nodes.

We assume all sensor nodes in the network are identical and have the same R_t and R_s. Besides, we assume that all nodes know their locations. This may be achieved via GPS modules or by use of localization algorithms [20, 21]. We also assume that nodes use data aggregation while forwarding the data they receive from their descendants. This is, however, not crucial. The scheme will work with no data aggregation as well.

Our combined sleep scheduling and routing scheme consists of four phases; global tier assignment, neighborhood table construction, mode selection, and operation phases. Global tier assignment phase and neighborhood table construction phase run only at network setup time; and the mode selection and operation phases run in each round (see Fig. 9).

Our scheme requires network to operate in rounds. A round is a fixed time interval, determining the frequency of mode re-assignments to nodes. A round consists of two phases executed sequentially one after another: mode selection phase and operation phase. Those two phases are of fixed length as well. The operation phase should be much longer than the mode selection phase. During mode selection phase, the mode of each node (which can either be ON-DUTY, or TR-ON-DUTY, or DEEP-SLEEP) is decided. During the operation phase, each node stays at the

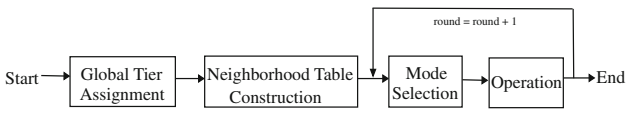


Fig. 9 Running order of the phases during the network lifetime

decided mode and data gathering from the sensor nodes to the sink happens. The round duration and the duration of its inner phases are same for all nodes and we assume all nodes are aware of this.

At each round, modes are re-assigned to nodes. At the beginning of each round, each node starts with a new mode selection phase where it selects a random delay and waits that much time before running the mode selection algorithm. Then it runs the algorithm and decides on its mode. Nodes may finish deciding their modes at different times, and wait until the start the operation phase to operate. Moreover, during the operation phase of a round, there can be multiple data gathering operations from full-active sensor nodes to the base station depending on the length of the round.

Round duration is a parameter that may affect energy consumption. The smaller the round duration is, the higher is the number of mode re-assignments, hence the higher is the energy consumption due to mode re-assignments. On the other hand, performing frequent mode re-assignments allows active nodes to be changed more frequently and enables a more even distribution of energy consumption among the nodes. This issue is discussed in [22] and a method about how to define an optimum round duration that provides minimum energy consumption is proposed.

During the execution of our algorithm, different control management messages are used. In Table 1, we list all these control messages used by our scheme together with their important fields (inside square brackets) and the situations when they are used. Here, s_{id} indicates the id of the source node (i.e. the node generating the message), d_{id} indicates the id of the destination node. Each control message has a code field indicating which control message it is. Since we have less than 16 different control messages, a 4-bit field is enough to hold the message code information. When a node receives a control message, it first looks to the 4-bit code field and then reacts according to the code and other fields of the message.

In the following sections, we describe the phases of our scheme in more detail. Moreover, we also explain how our scheme handles sensor network dynamics, such as transient and permanent link and node failures, new node arrivals and node departures.

4.1 Global tier assignment phase

In this phase, the goal is to create a tree-like routing structure rooted at the sink node that will be used in routing the packets from sensor nodes to the sink node. As a result of this phase, each node in the network is assigned a tier number (indicating how many hops the node is away from the sink node) and a parent node. Each node in the network that is active or semi-active forwards the data that it has generated or received to its parent node which has a smaller tier number. By this way, shortest path routing in terms of hop count is achieved and the possible routing loops are avoided.

After deployment, the sink node initiates the process of assigning tier numbers to all nodes in the network. For that it broadcasts a *GlobTierAssignment* message containing its ID and a tier number set to zero. Each node receiving a *GlobTierAssignment* message creates its own *GlobTierAssignment* message by incrementing the tier number by one and putting its own ID, and then broadcasts this new message to its neighbors. If a node receives multiple *GlobTierAssignment* messages, it only considers the message which has lower tier number than its current tier number. However, the node can record the IDs of all neighbors and their tiers as well. Furthermore, among the *GlobTierAssignment* messages having same tier number, the one coming from the closest node is considered (by utilizing the RSSI value) and that node is recorded as the current parent.

Table 1 All control messages used by our scheme

Message	When it is used
Hello	At the beginning of each round, [s_{id}]
GlobTier Assignment	In Global Tier Assignment phase, [s_{id}, tier-no]
TierQuery	To learn tier numbers of neighbors [s_{id}]
TierReply	To reply to TierQuery messages [s_{id}, tier-no]
Discovery	In Neighborhood Table Construction phase [s_{id}, Location, TTL]
StatusUpdate	In Mode Selection phase [s_{id}, status]
StatusQuery	To learn the status of 2- and 3-hop neighbors [s_{id}, TTL]
StatusReply	To reply StatusQuery messages [s_{id}, status, d_{id}]
Bye Message	When a node predicts to lose its energy in the next round [s_{id}]
LocalTierUpdate	When a node changes its tier number [s_{id}, tier no]
LocalFindParent	When a node can not find a parent among its 1-hop neighbors [s_{id}]
Connectivity-Ok	When a node can connect to another parent node [s_{id}]
Connectivity-Not-Ok	Used when a node cannot connect to another parent node [s_{id}]

4.2 Neighborhood table construction phase

Throughout this phase, each sensor node starts a discovery phase to learn about the other nodes in its vicinity and constructs a Neighborhood Table. This is needed for coverage check: a node may need to find out all other sensor nodes which have overlapping sensing areas with itself. Our scheme tries to discover the neighbors up to three hops and within $2R_s$ distance. Using multiple hop neighbors may provide better performance in the coverage check algorithm. However, this also causes a remarkable increase in the number of control messages. Therefore there is a tradeoff between a good coverage check and control message overhead. In [22], Bulut et al. discuss this tradeoff and find the conditions that give the maximum gain. As we mentioned earlier, since R_t/R_s value usually lies in the range [1/2, 3] [18, 19], even if the R_t/R_s value is quite small (i.e. 0.5), the neighbors that are more than 3-hops away cover either minor or no *extra* part of the node's sensing area as it is shown in Sect. 3.2 (Fig. 8).

At the beginning of the Neighborhood Table Construction phase, each sensor node broadcasts a *Discovery* message which contains the ID and the location of itself and a TTL value. Since each node may search for up to three hops, the TTL is set to 3 initially. Each node receiving this *Discovery* message records the ID and the location value in the message into its *Neighborhood Table*, unless the node originating the message is further than $2R_s$. Moreover, the receiver node decreases the TTL by 1 and if the TTL is still bigger than zero, the node forwards the message to other nodes within its transmission range. If a node receives multiple *Discovery* messages with the same ID (same source), it uses the one that has traveled the smallest number of hops (i.e. that has the largest TTL value). As a result of this process, each node learns about its 1-, 2-, and 3-hop communication neighbors.

4.3 Mode selection phase

In this phase each node of the network decides its mode for the remainder of the current round. Our scheme puts a sensor node in one of three modes:

ON-DUTY (full-active): Both the communication and sensing units are turned on.
TR-ON-DUTY (semi-active): Sensing unit is turned off, but communication unit is turned on. Hence the node can not sense the environment, but can transmit and receive data.
DEEP-SLEEP (inactive): Both the communication and sensing units are turned off. The node can neither sense, nor communicate.

We ignore the energy consumption in the processing unit and therefore we assume it is always on. However, the processing unit can be turned off as well when a node enters DEEP-SLEEP mode, provided that there is a timer hardware that can wake up the node when a round ends.

Mode selection phase consists of three parts which are executed sequentially in each node: (1) backoff delay computation and waiting, (2) coverage eligibility check, and (3) connectivity eligibility check. If a node passes coverage eligibility test, that means the sensing area of the node is covered by some other nodes, hence its sensing unit can be turned off. If a node passes connectivity check, that means its neighbors can by-pass the node while transporting packets towards the sink, hence its communication unit can be turned off. While performing these checks, however, due to the independent and distributed operation, some nodes may act simultaneously and attempt to change their modes at the same time. This can cause unhealthy results for eligibility checks. Therefore, we assign randomized backoff delays for each node so that when this time expires the node decides on its mode and informs its neighbors about this new mode via *StatusUpdate* messages.

Figure 10 shows the overall mechanism in mode selection phase. At the beginning of each round, each node selects a random backoff delay time and waits that much time in TR-ON-DUTY mode (other nodes may need to communicate with this node). When the backoff timer expires, the node first applies our coverage check algorithm. If it can not pass the coverage check, it goes into ON-DUTY mode, otherwise it applies our connectivity check algorithm. If it passes connectivity check, it goes into DEEP-SLEEP mode; otherwise it goes into TR-ON-DUTY mode. When operation phase comes, each node fulfills the requirements of its new mode.

4.3.1 Backoff delay computation

If the nodes attempt to determine their modes at the same time, no reasonable results may occur due to message contentions. We resolve this problem by using a backoff delay mechanism similar to the one proposed in SPAN [14]. Each node chooses a backoff value and when it expires the node decides its mode according to the states of the nodes in its neighborhood table at that moment.

In our solution, the backoff delay depends on two factors: the remaining energy levels and the number of neighbors of the nodes. A node with a lower remaining energy should have a shorter backoff delay, so that it can be the one who will decide to go to sleep earlier. A node with larger number of neighbors should have a longer backoff delay, so that it can be less likely to go to sleep and can stay active, since it can contribute to the coverage and communication of many other nodes. Let $N(i)$ be the

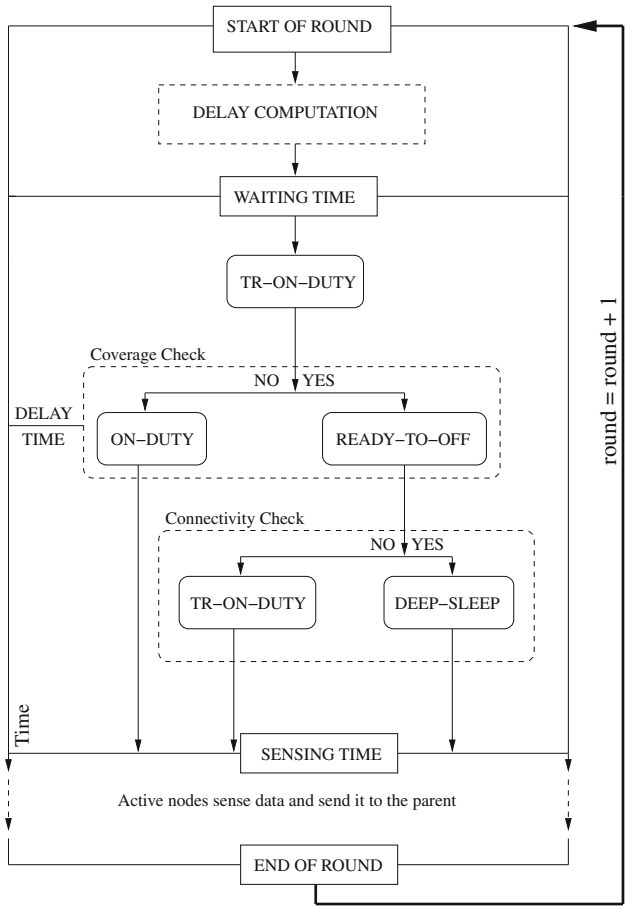


Fig. 10 The overall mechanism to decide which mode a sensor node will be in each round

neighbor count of node i and $N_{\max}$ be the maximum of $N(i)$'s in a network. The nodes having higher value of $N(i)/N_{\max}$ should have lower priorities for turning off their units due to their effect on coverage and connectivity. The following is our backoff time computation formula:

$$\text{Delay} = \left(\alpha\left(\frac{E_r(i)}{E_t(i)}\right) + \beta\left(\frac{N(i)}{N_{\max}}\right) + R\right) \times T \qquad (1)$$

Here R is a uniform random value in the interval $[0, 1 - \alpha - \beta]$, T is the size of random backoff time choices, and α, β are weights of energy and coverage parameters.

Although this delay mechanism produces different delays, some nodes may still have same or very close backoff delays. Thus, some nodes may decide their mode at the same time and some blind points covered by no nodes may occur. To prevent such kind of cases, we force each node to wait a short period of time T_w after deciding its mode. This time interval should be enough to receive possible *StatusUpdate* messages from a neighbor. If a message is received from a neighbor in this period, the node recalculates its off-duty eligibility. Otherwise, it changes its status to what it has decided. Besides it sends a *StatusUpdate* message to its neighbors. Choosing bigger T values can also decrease the contentions.

Note that, starting from the beginning of each round (including the backoff time), each node can receive messages from its neighbors and reply them accordingly. For instance, a *StatusUpdate* message from a 1-hop neighbor can be received and Neighborhood Table of the node can be updated. But, the states of 2-hop and 3-hop nodes are updated when they are needed, as described in the next section.

4.3.2 Coverage eligibility check

As soon as the backoff timer expires for a node, it runs the coverage eligibility algorithm with its current neighborhood information to check if its sensing area is covered by its neighbors with a ratio greater than a threshold value d_r. Here, d_r is a user defined parameter ($0 \leq d_r \leq 1$). If the amount of common coverage is 0, that means no neighbor is covering the sensing area of the node. If it is 1, that means the sensing area of the node is completely covered by the nodes in the neighborhood.

Given the location of nodes, there are various ways of computing common sensing area of a node with other nodes [15, 16, 23]. Some of these methods can only be used to decide whether the area is totally covered by other nodes or not, but can not be used to find out how much of the area is covered. A method that can give how much of a node's sensing area is covered by other nodes is a grid based approach. It first assumes a very fine grained grid put over the sensing area of the node. Hence the sensing area consists of many tiny grid cells. Then for each grid cell, it checks if the cell is covered by another node. This can be done by computing the distance of the grid cell to each of the other nodes. If the distance is less than R_s for any of the other nodes, then the grid cell is covered by another node. This is done for each grid cell, and then the percentage of the sensing area of the node covered by other nodes is computed.

To achieve a good coverage check, a node should know all of other nodes which have common sensing area with itself. However the number of nodes within $2R_s$ distance may be remarkably large and this may require creating lots of control messages to have an updated knowledge of these nodes' current modes. Here, we propose to utilize the expected overlap tables which are derived in the previous section. Note that in the expected overlap table of a node, the table cell (n_1, n_2) contains the expected overlap of this node's sensing area when it has n_1 1-hop and n_2 2-hop active neighbors.

Algorithm 1 shows the pseudo-code of our coverage check algorithm (N_1 stands for set of 1-hop neighbors and

d_r stands for desired coverage ratio). We assume *Expected Overlap Table* is installed to each node before deployment. According to the desired overlap, the node decides on the set of nodes in its Neighborhood Table that will be considered in the coverage check (we call this set as *coverSet*). Here, if the required set is only 1-hop neighbors, no extra messaging is needed to learn their status because 1-hop neighbor information is always updated via *StatusUpdate* messages. However, if the set also includes other nodes, the node should try to learn their current status from its neighbors. By this algorithm, we not only reduce the load due to control messages which update the status of nodes in the Neighborhood Table, but also make a good coverage check of nodes.

Algorithm 1 CoverageCheck (N_1: set of 1-hop neighbors, d_r: desired coverage ratio)

```
1: if (ExpectedOverlapTable(|N1|, 0) ≥ dr) ∨ (Rt/Rs ≥ 2) then
2:     coverSet = N1;
3: else
4:     Broadcast StatusQuery message with TTL 1;
5:     Wait Tw time for receiving StatusReply messages;
6:     N2 = The set of 2-hop nodes in ON-DUTY mode;
7:     if ExpectedOverlapTable(|N1|, |N2|) ≥ dr then
8:     coverSet = N1 ∪ N2;
9:     else
10:        Broadcast StatusQuery message with TTL 2;
11:        Wait Tw time for receiving StatusReply messages;
12:        coverSet = {active nodes in Neighborhood Table};
13:    end if
14: end if
15: coveredArea = 0;
16: desiredCover = dr × πRs^2;
17: for each node i in coverSet do
18:     coveredArea = coveredArea ∪ SensingArea(i);
19:     if coveredArea ≥ desiredCover then
20:         return TRUE;
21:     end if
22: end for
23: return FALSE;
```

If the node decides to use the nodes in two and three hops in coverage check, it needs to update the status information of these nodes. The node creates a *StatusQuery* message and broadcasts it with a TTL value which is set to 1 if the node wants to learn the updated status of 2-hop neighbors, and to 2 if the node wants to learn the status of 3-hop neighbors. A node receiving a *StatusQuery* message decreases the TTL value by 1. If TTL becomes zero after decrementing, the node replies back with a *StatusReply* message which contains the updated status of 1-hop neighbors in the Neighborhood Table of the node. Otherwise, the *StatusQuery* message is forwarded to other nodes. Moreover, when R_t/R_s is greater than 2, the algorithm defines the *coverSet* as the set of 1-hop neighbors because only they may have an overlapping area with the node.

It is important to note that our protocol never causes under coverage (i.e. coverage below the desired ratio). In Algorithm 1, using the *Expected Overlap Table* we find the upper hop level of the members of *coverSet* which will be used in coverage check algorithm. Once we have decided the *coverSet*, we do a real coverage check using only the information of nodes in *coverSet* (lines 18–23). Here, note that, expectedly the neighbors in *coverSet* may provide a higher overlap than the desired ratio but when the node performs the real coverage check using the locations of only these neighbors in *coverSet*, it may result that the sensing area of the node is not covered sufficiently by these neighbors so that the node decides to be in active mode. The reverse case is also possible: even if the *coverSet* with all neighbors in Neighborhood Table (all active neighbors up to three hops) does not provide the desired coverage expectedly, the real check may result in that the desired ratio of the node's sensing area is covered. But whatever the case is, a node is not put into sleep mode without a real coverage check. The most important benefit of using this *coverSet* and *Expected Overlap Table* idea is to save from unnecessary control messages. Although our protocol may result some nodes to be in active mode due to the difference of real coverage check and expected ratio, it achieves same or better performance (as it is shown in simulations) as the protocols updating the status of all hop neighbors continuously and it achieves this with less energy consumption by eliminating unnecessary control messages.

Moreover, in real sensor network applications, sometimes a node may not receive status information of some of its 2- and 3-hop neighbors due to transient link failures. In those cases, the node assumes that such neighbors are in DEEP-SLEEP mode during a round. Note that, this does not affect the regular running of our scheme and also it does not increase the cost of the protocol. But it can affect the number of neighbor nodes used in the coverage eligibility check of a node (*coverSet* in Algorithm 1 may change) and this may cause that node to select different modes.

4.3.3 Connectivity eligibility check

If a sensor node passes coverage check, it runs connectivity check algorithm as the next step of the mode selection phase. Even the sensing area of a sensor node is covered by its neighbors desirably, the node may be vital for the connectivity of other nodes. The active nodes (ON-DUTY

or TR-ON-DUTY) in a sensor network must be connected to be able to send their data to the sink node.

A node can be turned off without harming connectivity, if and only if its 1-hop active neighbors can send their data to sink over a path that does not contain this node. Hence the node should check if its 1-hop neighbors consider itself as the next-hop node (i.e. parent) in the route to the sink. If a node passes coverage check, it first changes its mode to a temporary mode called *READY-TO-OFF* and informs its 1-hop neighbors via *Status Update* messages. Then each 1-hop neighbor evaluates its current condition as follows and sends a reply message to the questioning node.

If the parent node of the neighbor node is not the questioning node, it sends a *Connectivity-Ok* message, since it does not need the questioning node for sending its data to the sink. Otherwise, the node looks for another possible parent (*PP*) among its neighbors. It sends a *Tier-Query* message asking their tier numbers to its 1-hop neighbors. According to the *TierReply* messages, it creates the set of *PP* nodes which have one less tier number than its tier number. If *PP* set is not empty, the node selects one of them as a new parent node and sends a *Connectivity-Ok* message to the questioning node. Otherwise it sends back a *Connectivity-Not-Ok* message, because there is no way to send its data to the sink node without using the questioning node and without possibly making the path longer.

After waiting sufficient time for the operations above, the questioning node checks whether it has received a *Connectivity-Not-Ok* message or not. If it is the case, it changes its mode to TR-ON-DUTY mode, otherwise it passes the connectivity check as well and goes into DEEP-SLEEP mode. Additionally, the node informs its 1-hop neighbors about its new mode via a *StatusUpdate* message, so that they can update their Neighborhood Tables.

In each round, the connectivity eligibility check algorithm is executed as part of mode selection. While putting some nodes into sleep and modifying the paths due to this, the connectivity check algorithm does not make the paths longer, since it selects a new parent that has one less tier number. If such a new parent can not be found for at least one child, the communication unit of the node is not turned off.

4.4 Operation phase

After mode selection phase is over, each sensor node has its mode determined for the operation phase, hence for the rest of the round. The node stays at that mode until that round finishes. When the round finishes and next round starts, all sleeping nodes (nodes in DEEP-SLEEP) wake up and start in TR-ON-DUTY (semi-active) mode. They again enter the mode selection phase and select a mode for the new round.

4.5 Handling network dynamics

The proposed scheme is designed to handle also some common sensor network dynamics to a certain degree such as transient and permanent node failures (due to hardware/software problems or energy depletion), transient and permanent link failures (due to obstacles, interference, fading, or relocation), and new node additions.

To detect transient link failures between nodes, our protocol requires each node to broadcast a *Hello* message to its 1-hop communication neighbors at the beginning of each round. Then, if a node i cannot receive a *Hello* message from one of its 1-hop neighbors (let's call that neighbor as node j) due to the failure of the wireless link in-between, it assumes that the node j will stay in DEEP-SLEEP mode during that round[2]. Therefore, node i decides its mode for the current round without considering the existence (so the status) of node j (hence the protocol behaves conservatively).

If, however, node i's current parent is j, node i needs to find a new parent node. In such a case, node i looks for a new parent node among its 1-hop neighbors. First, it searches for a neighbor having the same tier number with its previous parent (so that it can connect to that node without changing its tier number and without making the path to the sink node longer). If there is no neighbor with the same tier number, a non-child neighbor that has the smallest tier number (if exists any) is selected as the parent. This will cause the node i to change its tier number. Additionally, in this case, node i will instruct its children to update their tier numbers as well by broadcasting a *LocalTierUpdate* message in its subtree.

In some cases, a node i may not find a new parent to connect to among its 1-hop neighbors. In this case, node i sends a *LocalFindParent* message to its subtree (its descendants) and wants them to find a parent node which is not in this subtree. The first child node finding such a parent becomes the new root of this subtree. Then it starts a local tier update procedure by broadcasting a *LocalTierUpdate* message to the other nodes in this subtree.

It is important to not trigger the parent finding and tier update procedure (i.e. local recovery) for short durational failures. For this we can use a threshold time to trigger local recovery. That means, local recovery will be triggered only when the failure duration is longer than the threshold. Otherwise, local recovery is not triggered. Some data messages may be lost during this time. This is acceptable for link layers that are not designed to be totally reliable.

[2] We consider the links between nodes individually. If other neighbors of node j can receive Hello message from j (that link may not fail) even though node i can not receive it, they continue with the regular procedure and consider node j's status while deciding their own status.

We suggest the value of the threshold-time to be in the order of at least a certain fraction of one-round-duration.

Note that, while nodes continue to die, network may become disconnected after some time and in this case it may not be possible to find an alternative parent even from the nodes in the subtree, i.e. even *LocalFindParent* message does not work. No recovery can be done when network becomes physically disconnected.

In the above procedure, the probability that a node i may find a new parent among its 1-hop neighbors depends on the density of the network. Hence if the network is dense enough, then an alternative parent can be found instantly most of the time. However, depending on the tier of the new parent, we still may need to update the tiers in the subtree. We next show the relationship between the probability of finding a new parent among 1-hop neighbors, the density of the network, and link-failure probability.

Let n_i and c_i denote the number of neighbors and the number of children of a node i, respectively. Assume that node i's link to its parent has failed and node i is looking for a new parent node. The number of possible parents for i among its 1-hop neighbors is $P_i = n_i - c_i - 1$ (i.e. all neighbors except the children and the previous parent). Then, assuming that the average number of neighbors of a node is d and the total node count in the network is N, the average value of such possible parents for any node, P_{avg}, becomes:

$$P_{\text{avg}} = \frac{\sum_{i=1}^{i=N} n_i - c_i - 1}{N} = \frac{Nd - N - N}{N} = d - 2$$

Here, note that a node can be the child of only one node in the network at a given time. Therefore, the sum $\sum c_i$ is equal to N. As a result, the above formula states that a node can select one of the $d - 2$ neighbors as its new parent node on the average.

However, the link between a node i and its possible parent node may fail for some time. Assume the failure probability for such a link is p_f. Then, the probability of finding a new parent P_{parent}, becomes $1 - {p_f}^{d-2}$. In Fig. 11, we show the computed values of P_{avg} for different p_f and d values. Clearly, as p_f decreases and d increases, P_{avg} increases.

In many real applications of sensor networks, $d = 4 - 5$ is assumed to be a reasonable average neighbor count. Moreover, although it can change with respect to the sensor types, environment and hardware/software, etc., in some recent papers such as [24] and [25], the average link failure rate in sensor networks is assumed to be 15%. Hence, when $d = 5$ and $p_f = 0.15$, P_{parent} is 99.66%, which is a very high average probability. Therefore we can say that most of the time a node can instantly find a new parent among its 1-hop communication neighbors so that not much extra messaging will be required while searching for an alternative parent.

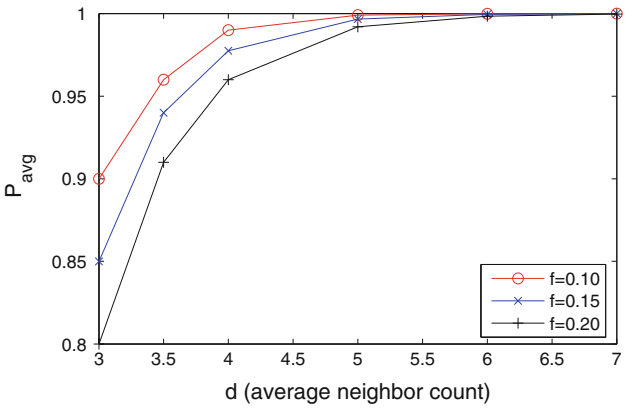


Fig. 11 The probability of finding a new parent node among the neighbors vs. average neighbor count with different link failure (p_f) rates

Above, we explained how the proposed scheme handles link failures. The failure of a node can simply be considered as the failure of all links from the node to its neighbors. Then the same procedure described above can be used to handle node failures. If a node failure occurs unexpectedly, some data can be lost until the situation is handled. But if the node failure happens due to the battery exhaustion, proactive rerouting may be performed before the node dies, and in this way data packet losses can be minimized. In our protocol, if a node notices that it will soon die due to very low remaining energy in its battery, it informs its 1-hop neighbors via a *Bye* message and asks them to find new parents and bypass itself while routing their messages towards the sink. Then each child of the dying node tries to find a new parent node similar to the procedure described above.

Note that, as nodes die and the number of nodes in the network decreases, one may think that the local parent search procedure done on the subtree (initiated by *LocalFindParent* messages) will be invoked more frequently and therefore the messaging cost will increase. However, our simulation results show that even nodes continue to die, the local parent search is invoked very infrequently. In a network of 100 nodes with $R_t = R_s = 10$ m on 50 m by 50 m region, on the average 60.6 nodes die before the total sensing coverage of connected nodes (with sink) becomes below 20% of whole network region. Of these 60.6 dead nodes, in the death of only 1.2 of them, the children of dying node cannot find a node to assign as a new parent from their neighbors so that they want help from the nodes in their own subtrees by *LocalFindParent* messages. Besides, the average number of nodes in such a subtree is 18.3, which is much less than the total node count in the network.

Moreover, our protocol can handle new node arrivals due to new node deployments or due to infrequent node

relocations. A newly arriving node will not have a parent assigned or it will not be able to find its parent in its new neighborhood. Therefore, the new node, after detecting its new neighbors, will select one of them as its new parent and will set its tier number accordingly. The neighbors that will detect this new node will also update their neighbor list (at the beginning of the round when *Hello* messages are received) and if selecting this new node provides smaller tier number, they will also update their own tier numbers and tier numbers of their descendents.

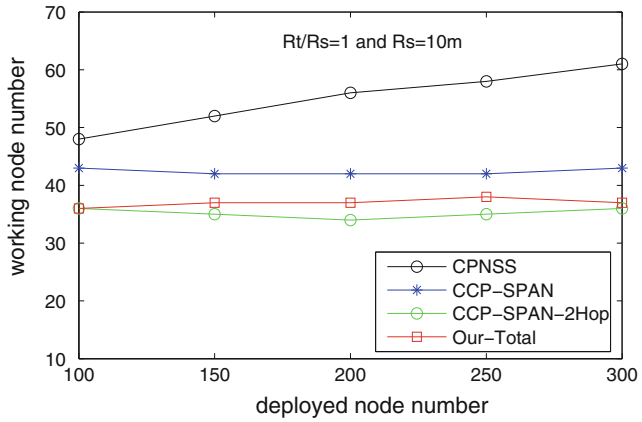
5 Performance evaluation

To evaluate our algorithm, we implemented a visual sensor network simulator in Java. The reason why we used a self-constructed custom simulator rather than a well-known one such as *ns*2 is, by this way, we could test our algorithm visually (nodes' locations and status after and before running our scheme) and skipped dealing with the unnecessary layers of nodes and protocol stack.

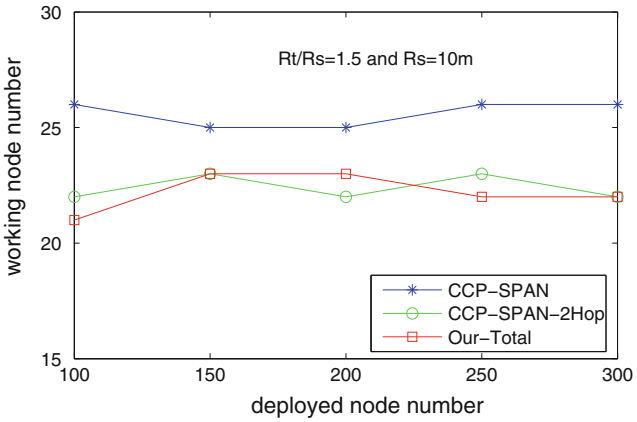
We assume that nodes are randomly and uniformly deployed (as it is assumed in [8, 16, 17]) and the sink node is located at the center of the region. We performed two types of simulations: (1) coverage performance tests to see the performance of our coverage check algorithm; and (2) system lifetime tests to see the extension of network lifetime with our solution.

5.1 Coverage performance tests

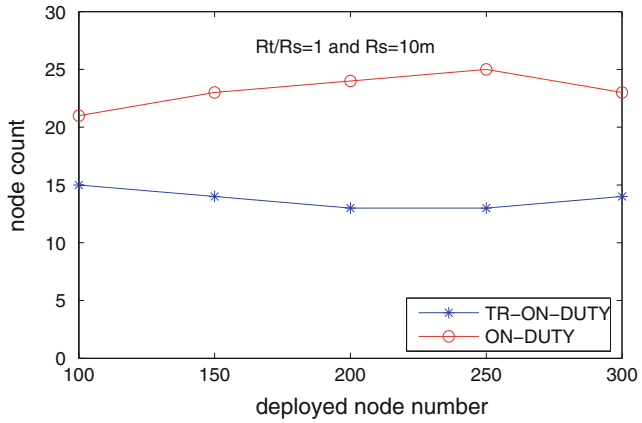
In this part of simulations we wanted to see what percent of nodes can be put into sleep mode by our algorithm maintaining the initial coverage and connectivity. We used a sensor network model similar to the one used in CCP [17]. We deployed 100 static nodes into 50 m by 50 m region with 10 m of R_s and R_t values. The coordinates of nodes are determined in a random manner at each run of the



(a) Working node count vs. Deployed node count. R_s=10m and R_t=10m.



(b) Working node count vs. Deployed node count. R_s=10m and R_t=15m.



(c) TR-ON-DUTY and ON-DUTY node count vs. Deployed node count. R_s=10m and R_t=10m.

Fig. 12 Working node counts and their distribution in our protocol

simulation experiments. The reported results in the graphs are obtained as the average of 100 different runs.

Figure 12a and b show the active node count for different number of nodes in the network when R_t/R_s is 1 and 1.5, respectively (we set $d_r = 100\%$). Both graphs show that our algorithm needs less number of nodes than CCP-SPAN algorithm and nearly the same number of nodes as CCP-SPAN-2Hop algorithm. However, the number of working nodes needed in our algorithm include both ON-DUTY and TR-ON-DUTY nodes. Figure 12c shows the number of TR-ON-DUTY and ON-DUTY node count for the case in Fig. 12a. As it is stated before, the nodes in TR-ON-DUTY mode turn off their sensing units hence their energy consumption is less than the nodes in ON-DUTY mode in which all units are turned on.

We also did simulations to see the effect of R_t/R_s ratio on the number active nodes. This time, we have decreased the value of R_s to 6.25 m to see the difference more clearly when R_t/R_s ratio is bigger. The number of deployed nodes is 800 when the R_t/R_s ratio is 0.5, and 200 in all other ratios.

Figure 13 shows the number of working node count for different R_t/R_s ratios. Our algorithm needs less number of nodes than CCP-SPAN algorithm for all ratios. On the other hand the difference in the number of working nodes needed by our algorithm and CCP-SPAN-2Hop algorithm is very small for all ratios; our algorithm needs a little less number of nodes. When the R_t/R_s ratio is 0.5 the difference is the biggest. This is due to our predictive coverage check algorithm which gives better results for small R_t/R_s ratios.

While in CCP algorithm nodes always update the status of their 1-hop neighbors via *HELLO* messages, in CCP-2Hop algorithm, nodes also update the status of their 2-hop neighbors. When they are combined with SPAN algorithm, each node needs to update their 2-hop neighbors because SPAN needs them for the coordinator announcement rule. However, in our algorithm, at first we only know the status

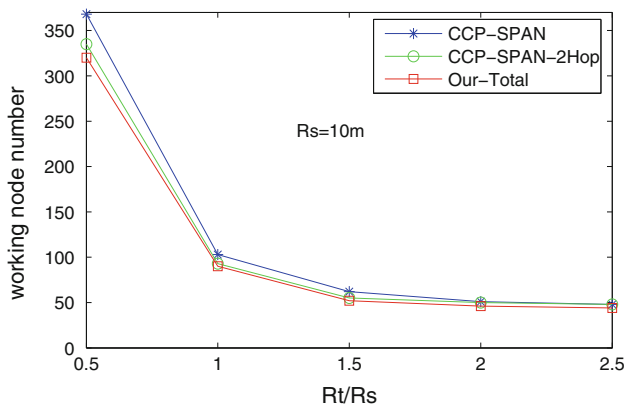


Fig. 13 Working node count vs. R_t/R_s ratio

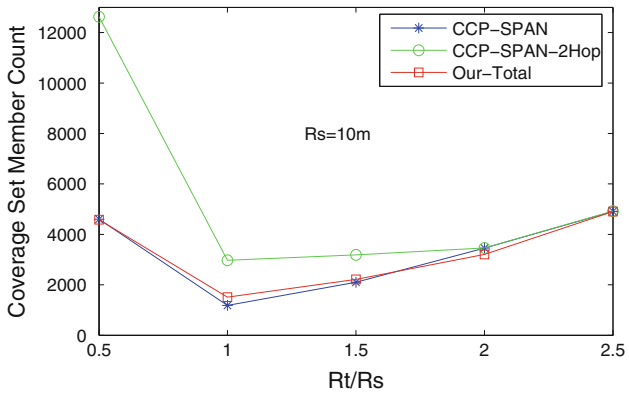
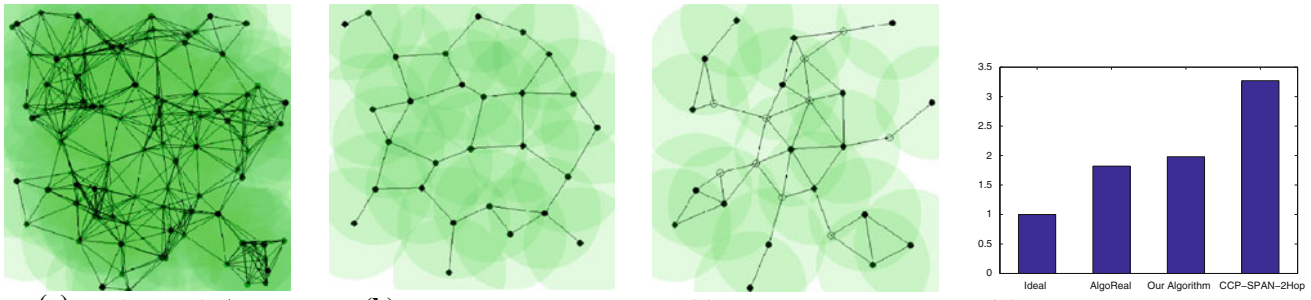


Fig. 14 Total number of neighbors used while performing the coverage check rule in a network

of 1-hop neighbors and whenever it is needed, we update the status of 2- and 3-hop neighbors.

We wanted to compare the overhead due to these kinds of messages used by our algorithm and CCP-SPAN and CCP-SPAN-2Hop algorithm. Figure 14 shows the total number of neighbors whose information is used by nodes to perform their coverage check. Our algorithm uses slightly more nodes than CCP-SPAN algorithm; however, it uses remarkably less number of nodes than CCP-SPAN-2Hop algorithm. It achieves the same number of working nodes as CCP-SPAN-2Hop algorithm while using less number of neighbors in coverage check. This figure only shows the total number of neighbors used in coverage check, but if we consider the messaging cost, our algorithm needs less messaging than both algorithms due to 2-hop neighbor updates required by SPAN algorithm. Besides, the more neighbors are used, the more messaging cost is incurred, since the status of neighbors need to be updated.

Furthermore, we also did simulations to show the coverage conservative behavior of our algorithm. We ran the two algorithms on a network with 100 nodes ($R_t = R_s$) and computed the percentage of each k-covered area within the whole network region. We set $d_r = 90\%$ in these simulations. In Fig. 15, we show the working nodes and their coverages in the initial network (Fig. 15a), in the network formed after running CCP-SPAN-2Hop algorithm (Fig. 15b) and in the network after running our algorithm (Fig. 15c). Moreover in Fig. 15d, we show the computed average number of nodes covering each point in the whole network region (it is computed as $\sum_{k=1}^{n} p_k k$, where p_k is the percentage of k-covered regions within the whole network region, and n is the number of nodes in the network). Both our algorithm and CCP-SPAN-2Hop algorithm keep more than 90% of the area covered (i.e. 99 and 98%, respectively). However, the redundant coverage resulted by CCP-SPAN-2Hop algorithm is significantly more than the redundant coverage resulted by our algorithm as shown in Fig. 15d (our algorithm achieves this by shutting down the



(a) Initial Network (100 nodes).

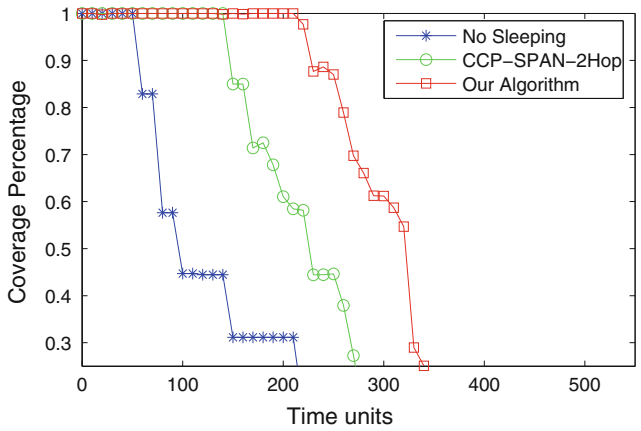
(b) After CCP-SPAN-2Hop Algorithm (28 full-active nodes).

(c) After Our Algorithm (19 full-active (filled circles) 10 semi-active nodes (empty circles)).

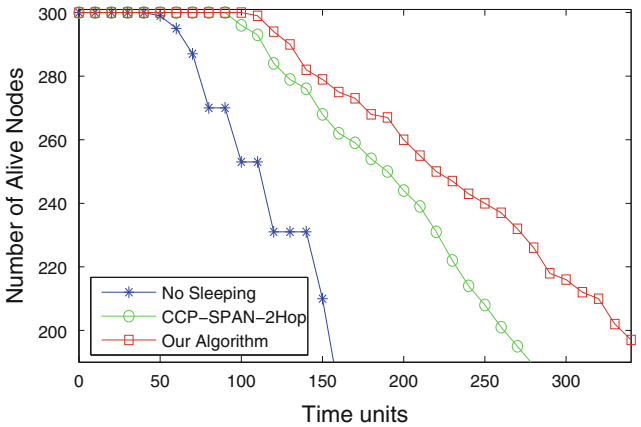
(d) Average number of nodes covering each point in the network region.

Fig. 15 Sample network snapshots before and after running two algorithms

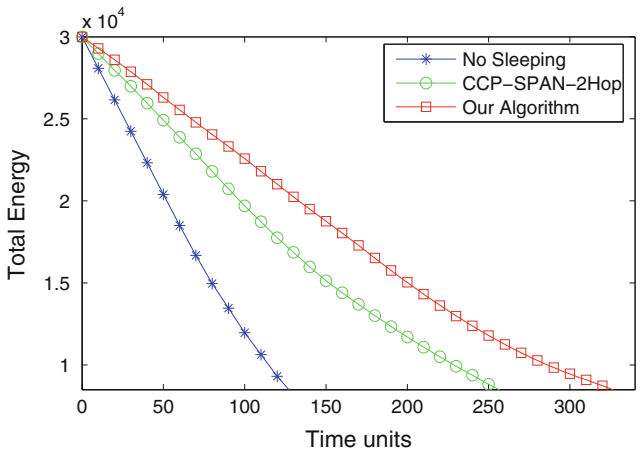
sensing units of some unnecessary nodes). In Fig. 15d, with the second vertical bar, we also show the average number of nodes covering each point in case of using all 3-hop neighbors in coverage check eligibility rule (we call it as *AlgoReal*) of our algorithm (normally, we use them if expected coverage ratio is not bigger than desired ratio). In other words, nodes do not consider the *Expected Overlap Tables* and always use information of active 2- and 3-hop neighbors, which requires continuous status update process for those nodes. From the figure, we observe that using



(a) Coverage percentage



(b) Number of still alive nodes



(c) Total energy in nodes

Fig. 16 Network statistics from a sample run when $R_t/R_s = 0.5$

expected values causes an insignificant redundant coverage over *AlgoReal*, but on the other hand, it achieves a great saving in control message overhead.

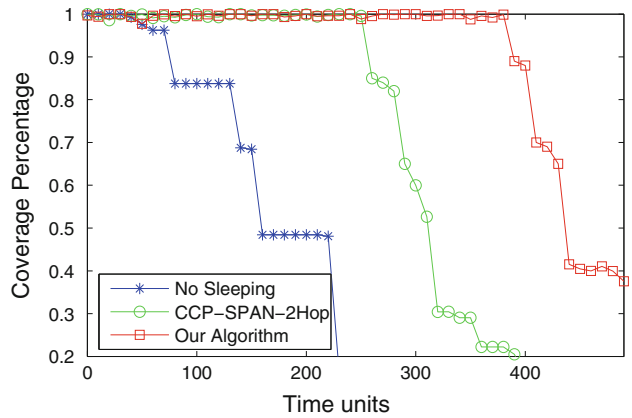
5.2 System lifetime tests

In this part of the simulations, we evaluated the energy efficiency and sensing coverage performance of our algorithm. In most of the early studies, only the energy spent in communication unit is considered. However, sensor nodes also consume energy in sensing and processing units. Since the energy consumed while processing data is very small compared to the energy consumed in other units, it is generally ignored. However, in some simulations authors also ignore the energy consumption in sensing unit. But according to [2] and [26], the energy dissipated for sensing one bit of information is approximately equal to the energy dissipated in receiving a bit. Therefore, in our simulations we consider the energy consumptions in both sensing and communication units.

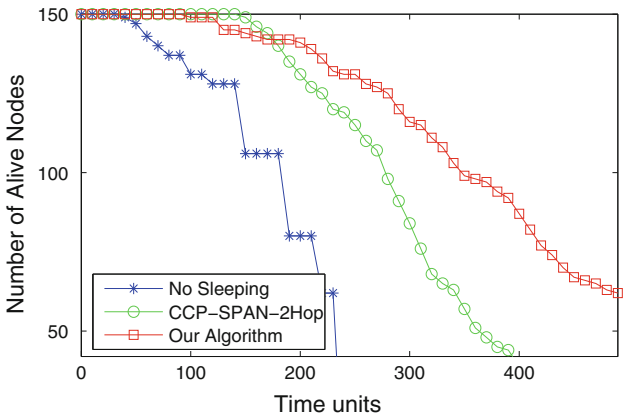
In the communication unit, we use the following energy consumption model for a node i in a network that applies a tree-based routing scheme. This is the model used in [27]. We also assume that the sensor nodes in the network perform data aggregation while forwarding their data.

$$E_{i,Communication} = E_{Receiving} * n_i + E_{Sending} \quad (2)$$

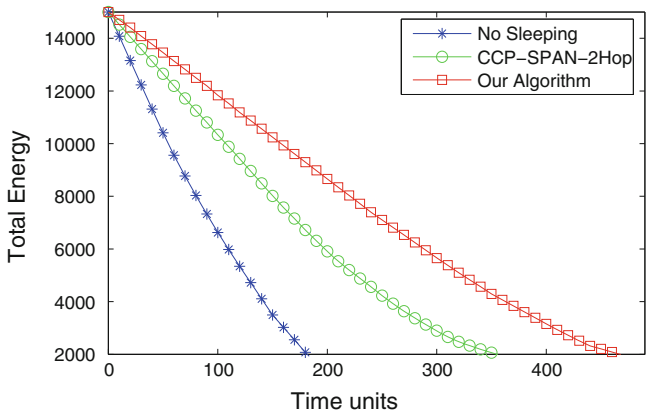
A node spends energy while receiving data from its n_i children and while sending the aggregated data packet to its parent. The constants $E_{Receiving}$ and $E_{Sending}$ depend on the communication technology. Some studies assume they are equal [27], and some others consider $E_{Sending}$ to be slightly larger than $E_{Receiving}$ [28, 29]. We assume a ratio of 2/2.5 for $E_{Receiving}/E_{Sending}$. Besides, we also consider the energy consumption in sensing unit. According to [2] and [26], $E_{Receiving}$ is equal to $E_{Sensing}$ for a bit worth of information. Therefore, we assume energy consumption ratio while sensing, receiving and sending data as 2:2:2.5, respectively. Moreover, if the size of a data message is q times the size of a *StatusUpdate* message, we assume that



(a) Coverage percentage



(b) Number of still alive nodes



(c) Total energy in nodes

Fig. 17 Network statistics from a sample run when $R_t/R_s = 1$

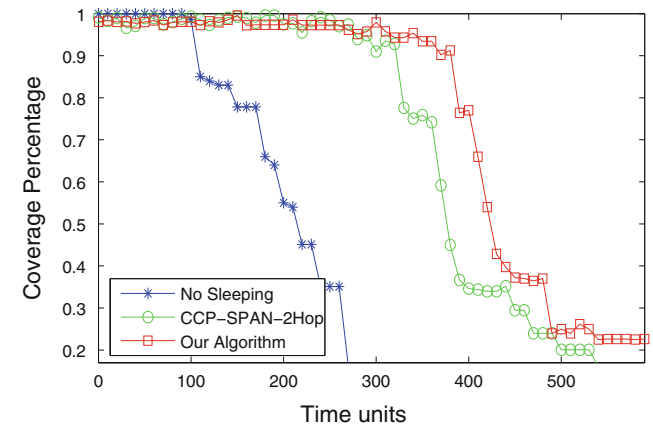
q is 20 in all simulations. However, we also made a simulation experiment showing the effect of the q constant.

For energy and coverage performance experiments, we set $R_s = 10$ m and $d_r = 90\%$. Besides, to see the effect of range ratio, we performed three different simulations for ratios 0.5, 1.0 and 1.5. The number of deployed nodes are 300, 150 and 80, respectively. The region is a square of 50 m $\times$ 50 m. The base station is located at the center of the region. Initially, each node is assumed to have 100 units of energy. In each round of communication, each node senses the environment multiple times, packetizes the information, and sends it towards the sink. The system is simulated until the coverage becomes very low.

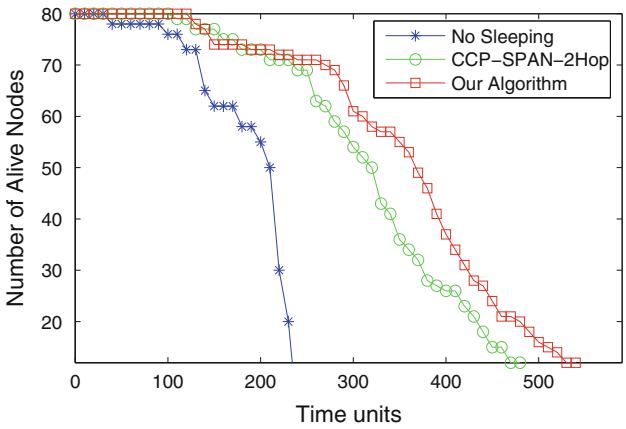
Figure 16 a, b and c show the coverage percentage, number of still alive nodes, and total remaining energy in nodes respectively. The range ratio is set to 0.5, that is, $R_t = 5$ m and $R_s = 10$ m. Here, coverage percentage is calculated considering only the coverages of nodes which can reach the sink node, i.e. which can send data to the sink node. From Fig. 16a, we observe that our algorithm maintains better coverage percentage in the later times of the network lifetime than CCP-SPAN-2Hop algorithm. This is mainly achieved by a quite distributed selection of active nodes in each round and the putting each unit of the sensor nodes into sleep separately. The higher number of alive nodes shown in Fig. 16b and the higher total energy in the nodes shown in Fig. 16c are also the consequences of these properties of our algorithm.

Figure 17 a, b, and c show the same metrics when the R_t/R_s ratio is 1 and Fig. 18a, b, c show them when range ratio is 1.5. Note that as R_t/R_s ratio increases, the performance of CCP-SPAN-2Hop algorithm gets closer to our algorithm due to the decreasing messaging cost of CCP-SPAN-2Hop. We also observe this fact in Fig. 14.

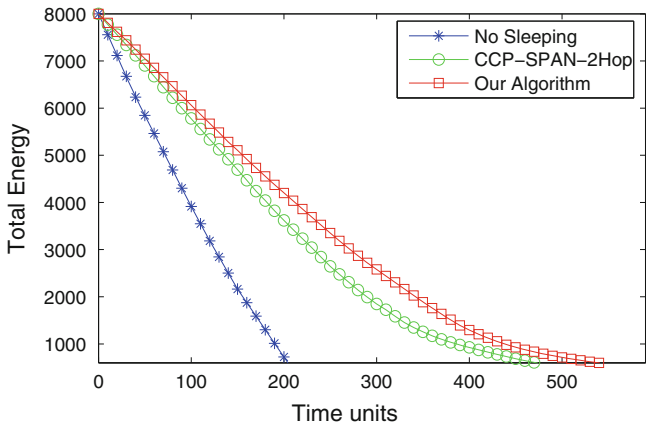
We also simulated the effect of changing q value. Figs. 19 a, b and c show the remaining energy of nodes in a sample network (with nodes having the same R_s and R_t) when CCP-SPAN-2Hop and our algorithm are applied, and when q is equal to 10, 20 and 50, respectively. Note that, when q increases, the energy consumption in CCP-SPAN-2Hop



(a) Coverage percentage

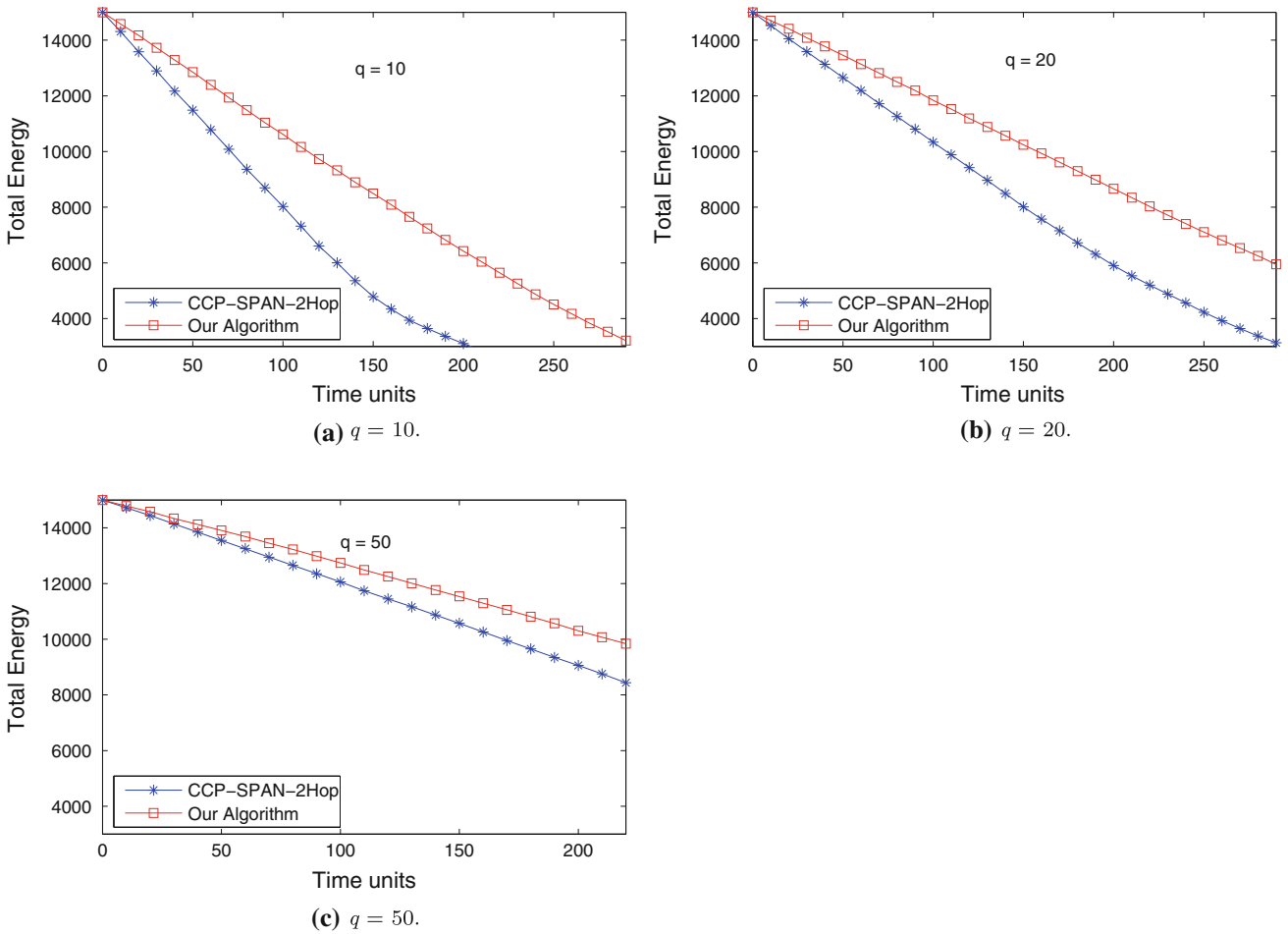


(b) Number of still alive nodes



(c) Total energy in nodes

Fig. 18 Network statistics from a sample run when $R_t/R_s = 1.5$



(a) $q = 10$.

(b) $q = 20$.

(c) $q = 50$.

Fig. 19 Total energy in nodes in a sample run with different q values

and our algorithm get closer to each other. Our algorithm reduces the amount of messages for maintenance and reduces the effect of these messages in energy consumption. However, if the size of these messages gets smaller with respect to data messages, then performance of our algorithm and CCP algorithm get closer.

6 Conclusion

In this paper, we investigated the sleep scheduling problem for energy conservation in wireless sensor networks. We first analyzed the coverage of a node's sensing area, and we found out that a sensor node can find the expected coverage ratio of its sensing area by knowing the transmission/sensing range ratio and the number of its 1-hop and 2-hop neighbors assuming uniform node distribution. Based on this analysis, we proposed a method to find out expected coverage that can be used in various problems as well.

As the second contribution of the paper, we provide a combined sleep scheduling and routing scheme that preserves connectivity and coverage with predictive coverage and multiple mode selections. In our solution, a sensor node can be in one of the three different modes. In ON-DUTY (full-active) mode, the sensor node has both its sensing and communication units turned on. In TR-ON-DUTY (semi-active) mode, the sensor node has the communication unit turned on, but the sensing unit turned off. In SLEEP (inactive) mode, the sensor node has both the sensing and communication units turned off. This is different than previous studies which only consider a sensor node as either active or sleeping.

Our algorithm is also flexible in terms of the transmission and sensing range ratio. That is, we do not assume a certain range ratio. This is also different from most of the previous studies which provide a solution for only a certain ratio. Our solution preserves a desired coverage and connectivity independent of this range ratio. Furthermore, our algorithm also reduces the number of control messages that

update the information about the states of the sensor nodes in the network. We use the update messages whenever they are required. The simulation results show that our scheme can be effectively used in dense sensor networks for energy efficient sleep scheduling while preserving connectivity and coverage with low control messaging overhead.

References

1. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002). Wireless sensor networks: A survey. *Computer Networks*.
2. Younis, M., Youssef, K., & Arisha M. (2002). Energy-aware routing in cluster-based sensor networks. *IEEE/ACM MASCOTS*.
3. Wang, L., & Xiao, Y. (2006). A survey of energy-efficient scheduling mechanisms in sensor networks. *Mobile Network Applications, 11*(5), 723–740.
4. Cardei, M., & Wu, J. (2005). Energy-efficient coverage problems in wireless ad hoc sensor networks, Computer Communications, special issue on Sensor Networks, Florida Atlantic University.
5. Slijepcevic, S., & Potkonjak, M. (2001). Power efficient organization of wireless sensor networks. *IEEE International Conference on Communications (ICC)*, 472–476. Los Angeles, California, USA.
6. Cardei, M., MacCallum, D., & Cheng, X. (2002). Wireless sensor networks with energy efficient organization. *Journal of Interconnection Networks, 3–4*(3).
7. Megerian, S., & Potkonjak, M. (2003). Low power 0/1 coverage and scheduling techniques in sensor networks, UCLA Technical Report.
8. Yardibi, T., & Karasan, E. (2008). A distributed activity scheduling algorithm for wireless sensor networks with partial coverage. *Wireless Networks Journal*.
9. Gupta, H., Das, S. R. & Gu, Q. (2003). Connected sensor cover: Self-organization of sensor networks for efficient query execution. In *Proceedings of MobiHoc'03*. USA.
10. Bulut, E., & Korpeoglu, I. (2007). DSSP: A dynamic sleep scheduling protocol for prolonging the lifetime of wireless sensor networks. In *Proceedings of IEEE Conference AINA*.
11. Xu, Y., Heidemann, J. S., & Estrin, D. (2001). Geography-informed energy conservation for Ad Hoc routing. *International Conference on Mobile Computing and Networking*.
12. Ye, F., Zhong, G., Lu, S., & Zhang, L. (2003). PEAS: A robust energy conserving protocol for long-lived sensor networks. In *The 23rd international conference on distributed computing systems*.
13. Ye, F., Lu, S., & Zhang L. (2001). Gradient broadcast: A robust, long-lived large sensor network, Technical Report, http://www.irl.cs.ucla.edu/papers/grab-tech-report.p.
14. Chen, B., Jamieson, K., Balakrishnan, H., & Morris, R. (2002). Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks, 8*, 481–494.
15. Tian, D., & Georganas, N. (2002). A coverage-preserving node scheduling scheme for large wireless sensor networks. In *Proceedings of international workshop on wireless sensor networks and applications*. USA.
16. Zhang, H., & Hou, J. C. (2005). Maintaining sensing coverage and connectivity in large sensor networks. *Wireless Ad Hoc and Sensor Networks Journal, 1*(12), 89–123.
17. Wang, X., Xing, G., Zhang, Y., Lu, C., Pless, R., & Gill, C. (2003). Integrated coverage and connectivity configuration in wireless sensor networks. In *Proceedings of Sensys 03* (pp. 28–39).
18. Crossbow Technology Inc. http://www.xbow.com, available Sept. 07.
19. Dust Inc. Products, http://www.dust-inc.com/products, available Sept. 07.
20. Ray, S., Lei, W., & Paschalidis, I. (2006). Statistical location detection with sensor networks. *IEEE/ACM Transactions on Networking, 14*(SI), 2670–2683.
21. Niculescu, D., & Badrinath, B. R. (2003). Ad hoc positioning system (APS) using AOA. In *Proceedings of INFOCOM*.
22. Bulut, E., Zheng, J., Wang, Z. & Szymanski B. (2008). Balancing the cost-quality tradeoff in cooperative ad hoc and sensor networks. In *Proceedings of IEEE MILCOM*. CA: San Diego Convention Center.
23. Huang, C., & Tseng, Y. (2003). The coverage problem in a wireless sensor network. In *WSNA 03: Proceedings of the 2nd ACM international conference on wireless sensor networks and applications*. USA.
24. Babbit, T., Morrell, C., & Szymanski, B. K. (2009). Self-selecting reliable path routing in diverse wireless sensor network environments. In *Proceedings of IEEE International Symposium on Computers and Communication, ISCC 09* (pp. 1–7). Sousse, Tunisia.
25. Al-Fares, M. S., Sun, Z., & Cruickshank, H. (2009). High survivable routing protocol in self organizing wireless sensor network. *IAENG International Journal of Computer Science, 36*, 2.
26. Jia, Y., Dong, T., & Shi, J. (2005). Analysis on energy cost for wireless sensor networks. In *ICESS'05: Proceedings of the second international conference on embedded software and systems*.
27. Heinzelman, W., Chandrakasan, A., & Balakrishnan, H. (2000). Energy-efficient communication protocols for wireless microsensor Networks. *International Conference on System Sciences*.
28. Kasten, O. Energy consumption, http://www.inf.ethz.ch/kasten/research/bathtub/energy/consumption.html.
29. Jung, E., & Vaidya, N. (2002). An energy efficient MAC protocol for wireless LANs. *INFOCOM*.

Author Biographies

Eyuphan Bulut received the B.S. and M.S. degrees in computer engineering from Bilkent University, Ankara, Turkey, in 2005 and 2007, respectively. He is currently a Ph.D. student in Computer Science Department of Rensselaer Polytechnic Institute in United States. His research interests include wireless ad hoc and sensor networks, delay-tolerant networks, and distributed systems.

Ibrahim Korpeoglu received his Ph.D. and M.S. degrees from University of Maryland at College Park, both in Computer Science. He is currently an Assistant Professor in the Computer Engineering Department of Bilkent University, Ankara, Turkey. Prior to joining Bilkent University, he worked in Ericsson, IBM T.J. Watson Research Center, Bell Labs, and Telcordia Technologies, in USA. He has served on the program committees of several conferences and published numerous papers in the area of computer networking. His research interests include computer networks, wireless ad hoc and sensor networks, mobile computing, and P2P networks.