Appears in Proceedings of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)

Sentries and Sleepers in Sensor Networks

Mohamed G. Gouda[1], Young-ri Choi[1], and Anish Arora[2]

[1] Department of Computer Sciences, The University of Texas at Austin,
1 University Station C0500, Austin, TX 78712-0233, U.S.A.
{gouda, yrchoi}@cs.utexas.edu
[2] Department of Computer and Information Science, The Ohio State University,
2015 Neil Avenue, Columbus, OH 43210-1277, U.S.A.
anish@cis.ohio-state.edu

Abstract. A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network can use their antennas to communicate in a wireless fashion by broadcasting messages over radio frequency to neighboring sensors in the same network. In order to lengthen the relatively short lifetime of sensor batteries, each sensor in a network can be replaced by a group of n sensors, for some $n \geq 2$. The group of n sensors act as one sensor, whose lifetime is about n times that of a regular sensor as follows. For a time period, only one sensor in the group, called *sentry*, stays awake and performs all the tasks assigned to the group, while the remaining sensors, called *sleepers*, go to sleep to save their batteries. At the beginning of the next time period, the sleepers wake up, then all the sensors in the group elect a new sentry for the next time period, and the cycle repeats. In this paper, we describe a practical protocol that can be used by a group of sensors to elect a new sentry at the beginning of each time period. Our protocol, unlike earlier protocols, is based on the assumption that the sensors in a group are perfectly identical (e.g. they do not have unique identifiers; rather each of them has the same group identifier). This feature makes our protocol resilient against any attack by an adversary sensor in the group that may lie about its own identity to be elected a sentry over and over, and keep the legitimate sensors in the group asleep for a long time.

Key words: Energy management, Sentry election, Self-stabilization, Sensor Networks, Sentry-Sleeper protocol

1 Introduction

A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network can use their antennas to communicate in a wireless fashion by broadcasting messages over radio frequency to neighboring sensors in the same network. Due to the limited range of radio transmission, sensor networks are usually multi-hop. Sensor networks can be used for military, environmental or commercial applications such as intrusion detection [2], disaster monitoring [1] and habitat monitoring [9].

One of the challenging problems in designing sensor networks is to lengthen the lifetime of sensor batteries. One approach to solve this problem is to exploit the idea that in some densely deployed networks, a fraction of the sensors can go to sleep for predefined time periods, while the remaining sensors stay awake and perform the assigned tasks in the network. The sleeping sensors save their energy and lengthen the lifetime of their batteries, without significantly degrading the performance of the applications running on the sensor network. Examples of this approach can be found in [6], [5], [17], [8], [18], [20], [12], [4].

In the current paper, we generalize this idea to be applicable to any, possibly sparsely populated, sensor network: replace each sensor in the network by a group of n sensors, for some $n \geq 2$. The group of n sensors are deployed in a location where a single sensor would have been deployed in the sparse network. This group of n sensors act as one sensor as follows. For a time period, only one sensor in the group, called *sentry*, stays awake and performs all the tasks assigned to the group, while the remaining

sensors, called *sleepers*, go to sleep to save their batteries. At the beginning of the next time period, the sleepers wake up, and all the sensors in the group elect a new sentry for the next time period, and the cycle repeats.

Note that the sensors in a group are identical in every way so that each of them can behave in exactly the same manner in performing the assigned tasks, when this sensor is elected a sentry of the group. This implies that no sensor has an identifier that distinguishes it from other sensors in its group. Rather, every sensor in a group has the same group identifier.

The identifiers of two sensor groups in the same network, however, are distinguishable so that when a sensor receives a message, the sensor can determine whether this message was sent from a sensor in its own group or it was sent from a sensor in a different nearby group. Note that a sensor in a group needs to exchange messages with other sensors in its group in order to elect a new sentry at the beginning of each time period. A sensor also needs to exchange messages with sensors in adjacent groups in order to perform the assigned tasks, when this sensor is elected a sentry of its group.

An alternative approach to lengthen the lifetime of sensor batteries in a sparsely populated network is to provide each sensor with a large battery whose lifetime is n times the lifetime of a regular battery. However, this alternative approach is less reliable than our approach (where each sensor in the sparsely populated network is replaced by a group of n sensors) as follows. If a sensor fails in a network, then the network can compensate for the failed sensor provided the network is designed using our approach rather than the alternative approach.

The protocol used by a group of n sensors to elect a new sentry at the beginning of each time period is called a *sentry-sleeper protocol*. The goal of a sentry-sleeper protocol is two-fold:

i. Ensure that at each instant not all the sensors in a sensor group are sleeping. Thus, at each instant at least one sensor in the group is awake and so can perform the tasks assigned to the group.
ii. Reduce the time periods where two or more sensors in a sensor group are awake in order to reduce the wasteful use of sensor batteries. (Note that if two or more sensors in a sensor group are awake during a time period, then each of them performs the same tasks assigned to the group during that period.)

Other sentry-sleeper protocols are reported in [5], [4], [6], [18], [8]. The main assumption in these papers is that the sensors in a "sensor group" have distinguishable identities; i.e. they have different physical locations, different connectivity, different message traffic, or different identifiers. Thus, the sensors in the network decide which one stays awake among their neighboring sensors based on these different identities, so that they can not only save their batteries but also provide some level of the performance of the applications running on the network. Unlike these protocols, our protocol for electing a sentry at the beginning of each time period is based on the assumption that the sensors in a group are perfectly identical; i.e. they have identical locations, connectivity, traffic, and identifiers. This feature makes our protocol scalable and resilient against any attack by an adversary sensor in the group that may lie about its own identity (i.e. lie about their locations, connectivity, ...) to be elected a sentry over and over, and keep the legitimate sensors asleep for a long time.

2 Sensor States and Transitions

Before we can explain the main features of our sentry-sleeper protocol, we need to explain, in this section, the different states of a sensor and the transitions between them.

Every sensor in a sensor group can be in any one of two states: an idling state or a sleeping state. In the idling state, the sensor does nothing but wait until either its timeout expires (in which case the

sensor executes a timeout action), or it receives a message (in which case the sensor executes a receiving action). An action, whether a timeout action or a receiving action, of a sensor consists of a number of statements that update the local variables of the sensor, send at most a message, or set the timeout of the sensor to expire at a later time.

Also the sensor can execute the special statement "go-to-sleep" at the end of an action. If a sensor executes this statement, the sensor changes its state from idling to sleeping. In the sleeping state, the sensor does nothing but wait until its timeout expires, then it executes a timeout action and changes its state from sleeping to idling. Figure 1 shows the two states of a sensor and the different transitions between them.

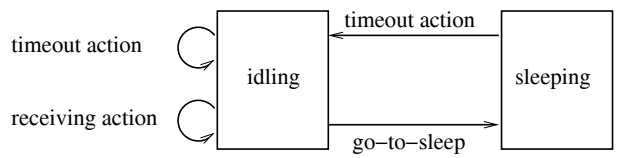

Fig. 1. Two states of a sensor

There are two main differences between the idling state and the sleeping state of a sensor. First, in the idling state, the sensor can receive messages that are sent by other sensors and execute corresponding receiving actions, whereas in the sleeping state, the sensor cannot do so since it turns off its radio as well as its processor and sensing devices to save energy during its sleep. Second, the consumed energy when the sensor is in the idling state is much larger than the consumed energy when the sensor is in the sleeping state (as discussed in [10] and [19]). Therefore, for the sensor to save its energy as much as possible, it should stay in its sleeping state as long as possible.

3 Sensor Network Execution

In this section, we present a formal model of the execution of a sensor network. We use this model to specify our sentry-sleeper protocol in the next section. We also use this model to verify and analyze the protocol in Section 5 and 6, and to develop our simulation in Section 7.

The *topology* of a sensor network is a directed graph that satisfies the following two conditions. First, each node in the topology represents a distinct sensor in the sensor network. Second, each directed edge (u, v) from node u to node v in the topology indicates that every message sent by sensor u can be received by sensor v (provided that neither sensor v nor any "neighboring sensor" of v sends a message at the same time when sensor u sends its message).

If the topology of a sensor network has a directed edge from a sensor u to a sensor v, then u is called an *in-neighbor* of v and v is called an *out-neighbor* of u. Note that a sensor can be both an in-neighbor and an out-neighbor of another sensor in the sensor network.

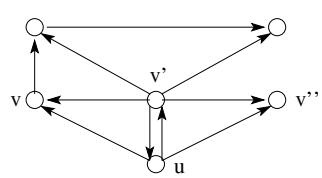

Fig. 2. Topology of a sensor network

As an example, Figure 2 shows the topology of a sensor network. This network has six sensors, and sensor u in this network has three out-neighbors, namely sensors v, v', and v''. Thus, if sensor u sends a

message, then this message can be received simultaneously by the three sensors v, and v', and v''. Note that sensor u is both an in-neighbor and an out-neighbor of sensor v' in this network.

We assume that during the execution of a sensor network, the real-time passes through discrete instants: instant 1, instant 2, instant 3, and so on. The time periods between consecutive instants are equal. The different activities that constitute the execution of a sensor network occur only at the time instants, and not in the time periods between the instants. We refer to the time period between two consecutive instants t and $t+1$ as a *time unit* $(t, t+1)$. (The value of a time unit is not critical to our current presentation of a sensor network model, but we estimate that the value of the time unit is around 100 milliseconds.)

At a time instant, the time-out of a sensor u may expire causing u to execute its timeout action. Executing the timeout action of sensor u causes u to update its own local variables and to send at most one message. It may also cause u to execute the statement "timeout-after <expression>" which causes the time-out of u to expire (again) after k time units, where k is the current value of <expression>. It may also cause u to execute the statement "go-to-sleep" which causes u to sleep until the time-out of u expires. The timeout action of sensor u is of the following form:

```
timeout-expires -> <update local variables of u>;
                   <send at most one message>;
                   <may execute timeout-after <expression>>;
                   <may execute go-to-sleep>
```

To keep track of its time-out, each sensor u has an implicit variable named "timer.u". In each time unit between two consecutive instants, the implicit variable timer.u is either "present" or "not-present". Moreover, if variable timer.u is present in a time unit, then it has a positive integer value in that time unit. Otherwise, it is not-present and has no value in the time unit.

If sensor u executes a statement "timeout-after <expression>" at instant t, then timer.u becomes present in the time unit $(t, t+1)$ and its value is the value of <expression> in this time unit.

If timer.u is present and its value is k, where $k > 1$, in the time unit $(t-1, t)$, then timer.u continues to be present and its value is $k-1$ in the time unit $(t, t+1)$.

If timer.u is present and its value is 1 in the time unit $(t-1, t)$, then sensor u executes its timeout action at instant t and timer.u becomes not-present in the time unit $(t, t+1)$, unless u executes "timeout-after <expression>" as part of its timeout action.

If a sensor u executes its timeout action and sends a message at instant t, then any sensor v, that is an out-neighbor of u, receives a copy of the message at instant t, provided that the following two conditions hold.

i. Sensor v does not send any message at instant t. (This condition indicates that either sensor v does not execute its timeout action at t, or it executes its timeout action at t but this execution of the timeout action does not include sending a message.)
ii. Sensor v has no in-neighbor, other than sensor u, that sends a message at instant t. (If v sends a message at t or if an in-neighbor of v, other than u, sends a message at t, then this message is said to *collide* with the message sent by u at t with the net result that v receives no message at t.)

If a sensor u receives a message at instant t, then u executes its receiving action at t. Executing the receiving action of sensor u causes u to update its own local variables. It may also cause u to execute the statement "timeout-after <expression>" which causes the time-out of u to expire after k time units, where k is the value of <expression> in the time unit $(t, t+1)$. It may also cause u to execute the statement "go-to-sleep" which causes u to sleep until the time-out of u expires. The receiving action of sensor u is of the following form:

```
rcv <msg> -> <update local variables of u>;
             <may execute timeout-after <expression>>;
             <may execute go-to-sleep>
```

It follows from the above discussion that at a time instant, a sensor u executes exactly one of the following:

i. u sends one message, but receives no message.
ii. u receives one message, but sends no message.
iii. u sends no message and receives no message.

In the next section, we specify our sentry-sleeper protocol using the formal model of sensor protocols in this section.

4 The Sentry-Sleeper Protocol

The goal of our sentry-sleeper protocol is to make a group of n sensors act as a single sensor whose lifetime is $N * F$ time units, where F is the lifetime of a regular sensor, and $1 < N < n$. The n sensors in the sensor group constitute a sensor network whose topology is fully-connected, i.e. there are two opposite-direction edges between every two nodes in the topology.

During a time period, called a *turn*, $(n - 1)$ sensors of the sensor group are in their sleeping states and the remaining sensor is in its idling state. In a turn, each of the sleeping sensors is called a *sleeper*, and the awake sensor is called a *sentry*. At the end of a turn, the sleepers wake up and all sensors in the group elect a new sentry for the next turn. This cycle of a turn followed by an election of a new sentry is repeated over and over until the batteries of all sensors in the group are exhausted.

At the end of a turn, the sleepers wake up, and they along with the sentry collaborate to elect a new sentry for the next turn as follows. Each sensor in the group computes a random period, called a *resolution period*, whose length is chosen uniformly from the range 1 .. 2*$ravg-1$, where $ravg$ is the average length (measured in time units) of the resolution period. Then, each sensor sets its timeout to expire after its resolution period. The sensor that chooses the smallest resolution period in the group times-out first, and this sensor elects itself as the new sentry and starts the new turn by sending a message of the form:

`sleep(`gid`, `rt`)`

where gid is the identifier of the sensor group and rt is the remaining time in the current turn. Initially, the remaining time in the current turn is assigned the length of a turn, which is tl time units.

When a sensor u in the sensor group receives a sleep(gid, rt) message, sensor u recognizes that a new sentry is elected for the current turn and decides to sleep for rt time units. Thus, it sets its timeout to expire after rt time units, then goes to sleep. The range of rt in the received sleep message is 1 .. tl. Thus, the shortest sleeping period is 1 time unit, and the longest sleeping period is tl time units.

After the elected sentry sends the first sleep(gid, rt) message, the sentry computes a random period whose length rp is chosen uniformly from the range 1 .. 2*$ravg-1$, and sets its timeout to expire after rp time units. When the sentry times-out, it sends the next sleep(gid, $rt - rp$) message. The sentry keeps on sending sleep messages, until the remaining time in the current turn becomes zero and all the sleepers wake up to elect a new sentry for the next turn.

Notice that the sentry periodically sends a sleep message even when all other sensors in the group are supposedly asleep and cannot receive any messages. This feature is intended to handle the following case. Some sensors in the group may not receive the first sleep(gid, rt) message sent by the sentry at the beginning of a turn. These sensors can receive a later sleep(gid, rt') message, where $rt' < rt$ and go to sleep for a period of rt' time units in this turn.

In this protocol, two (or more) sensors, say u and v, in the group can select identical resolution periods and so they send their sleep messages at the same time. The net effect is that none of the sensors in the group can receive any sleep messages, since the two messages collide with one another. Only sensors u and v consider themselves as sentries, and the other sensors in the group do not recognize that a new sentry has been elected for the current turn. However, our protocol ensures that one, only one, sensor in the group eventually sends a sleep message at some instant t and makes all other sensors go to sleep at t.

A formal specification for a sensor u in the group is as follows.

```
sensor u

const gid     : integer,     {group id of sensor u}
      tl      : integer,     {length of a turn}
      ravg    : integer      {avg length of random period}
var   sentry  : boolean,     {Is u sentry?}
      awake   : boolean,     {Is u awake?}
      rp      : 1..2*ravg-1, {length of random period}
      rt      : 0..tl,       {remaining time in current turn}
      g       : integer,     {group id in received message}
      t       : 1..tl        {remaining time in received message}
begin
    timeout-expires -> if !awake ->            awake := true;
                                               sentry := false;
                                               rp := random;
                                               timeout-after rp

                       [] awake and !sentry -> sentry := true;
                                               rt := tl;
                                               send sleep(gid, rt);
                                               rp := random;
                                               rp := min(rp, rt);
                                               rt := rt-rp;
                                               timeout-after rp

                       [] awake and sentry ->
                                    if rt>0 -> send sleep(gid, rt);
                                               rp := random;
                                               rp := min(rp, rt);
                                               rt := rt-rp;
                                               timeout-after rp
                                    [] rt=0 -> sentry := false;
                                               rp := random;
                                               timeout-after rp
                                    fi
                       fi

[]  rcv sleep(g, t) -> if gid=g  -> sentry := false;
                                    awake := false;
                                    timeout-after t;
                                    go-to-sleep
                       [] gid!=g -> skip
                       fi
end
```

It is important to note that in this protocol, the sensors in a group compete to become a sentry purely based on randomization without resorting to any difference in their identities that may give an advantage

to some sensors over others in the group. In a turn, each sensor in the group has the same probability to become a sentry. Thus, each sensor can expect to become a sentry once every n turns or so. A sensor u who fails to become a sentry for a relatively long period, say for $3 * n$ or $5 * n$ turns, should suspect that some sensors in the group are not following the protocol. In this case, sensor u may decide to stay awake (and perform the assigned tasks to the group) and refuse to go to sleep.

5 Stabilization of the Protocol

In this section, we sketch a proof that our sentry-sleeper protocol is self-stabilizing. A *state* of this protocol is defined by a value for each variable and each implicit variable timer.u for each sensor u in the protocol. Note that a state of the protocol corresponds to a time unit between two consecutive instants, since the values of all variables and all implicit variables do not change during any time unit between consecutive instants.

We assume that every state (whether legitimate or illegitimate) of the protocol satisfies the following three conditions.

1. For every sensor u, the implicit variable timer.u is present and its value is at most tl time units. (Note that this assumption is maintained by the execution of the protocol.)
2. For every sleeping sensor u, the value of its $awake$ variable is false. (Note that this assumption is maintained by the execution of the protocol.)
3. For every awake sensor u, the value of its timer.u is distinct from the value of timer.v for any other awake sensor v. (Note that this assumption is probabilistically maintained by choosing the value $ravg$ to be large relative to the number of sensors in the group.)

In our sentry-sleeper protocol, a *legitimate* state is defined as a state that satisfies the following *invariant*:

At least one sensor in the group is awake, and
at most one sensor in the group is a sentry.

Therefore, in a legitimate state, the number of sleepers is in the range 0 .. $n-1$, and the number of sentries is in the range 0 .. 1.

The protocol is *self-stabilizing* iff it satisfies the following two conditions [3].

i. *Closure*: Starting from any legitimate state, the execution of any action in any sensor in the protocol yields a legitimate state.
ii. *Convergence*: Starting from any illegitimate state, the protocol is guaranteed to reach a legitimate state.

First, we show that starting from any legitimate state, the execution of any action in any sensor in the protocol yields a legitimate state. The protocol has two cases to consider. In the first case, the executed action is a timeout action in a sensor u in the group. In this case, there are three possibilities to consider when the timeout action is executed.

i. The value of $awake$ in u is false: In this case, u concludes that u wakes up from sleeping (by the assumption 2), and makes the value of $awake$ true. Thus, u becomes awake, and so the invariant holds.
ii. The value of $awake$ in u is true and the value of $sentry$ in u is false: In this case, u elects itself as the new sentry and makes other sensors in the group sleep by sending a sleep message. Note that no other awake sensor can execute this timeout action that causes the sensor to send a sleep message at the same time (by the assumption 3). Thus, u is awake and becomes the only sentry in the group, and so the invariant holds.

iii. The value of $awake$ in u is true and the value of $sentry$ in u is true: In this case, there are two cases to consider depending on the remaining time in the current turn. First, if the remaining time is bigger than zero, u sends another sleep message. Thus, u is still awake and is still the only sentry in the group. Second, if the remaining time is zero, u recognizes that the current turn is finished, and withdraws from a sentry by making its value of $sentry$ false. Thus, u is still awake, but is not a sentry any more. In both cases, the invariant holds.

In the second case, the executed action is a receiving action in a sensor u in the group. In this case, there are two possibilities to consider when the receiving action is executed.

i. When u receives a sleep message from another sensor v in the same group: In this case, u recognizes that sensor v is elected a sentry for the current turn, so u goes to sleep for the specified sleeping period in the received message. Thus, sensor v is awake and is the only sentry in the group, and so the invariant holds.
ii. When u receives a sleep message from a sensor in a different group: In this case, u ignores the message and does nothing. Thus, the invariant holds.

Hence, starting from any legitimate state, the execution of a timeout action or a receiving action in any sensor in the group yields to a legitimate state.

Next, we show that starting from any illegitimate state, our protocol is guaranteed to reach a legitimate state within finite executions of actions in the group. There are two states that violate the invariant as follows:

i. A state where all sensors in the group are sleeping: In this case, a sensor u in the group is guaranteed to execute its timeout action within tl time units (by the assumption 1). By executing the timeout action of u, u becomes awake. Thus, at least one sensor in the group will wake up within tl time units, and then only one of the awake sensors will eventually become a sentry.
ii. A state where two or more sentries exist in the group: In this case, only one sentry whose timer value is the smallest, say sensor u, times-out first and then executes its timeout action to send a sleep message at some instant t (by the assumption 3). The other sentries receive the sleep message from u and go to sleep at t. Thus, all the sentries except u go to sleep within finite executions of actions.

Therefore, starting from any illegitimate state, the execution of a timeout action in some sensor makes the protocol reach a legitimate state within finite executions of actions in the group.

6 Protocol Analysis

Our protocol, as described in Section 4, makes a group of n sensors act as one sensor whose lifetime is $N * F$ time units, where F is the lifetime of a regular sensor and N is some quantity, called *the effective number of the sensors* in the group. Clearly, we have $1 < N < n$. In this section, we analyze the protocol and estimate the value of N.

Figure 3 shows a time period T, consisting of a resolution period followed by one turn of tl time units. Since the average length of a resolution period is $ravg$ time units, we have $T = tl + ravg$ time units. During a turn, a sentry sends a sleep message every random period rp whose average is $ravg$ time units. Therefore, the average number of sleep messages sent by a sentry per turn is $tl/ravg$.

Let E_{slp} and E_{idl} be the amount of energy consumed by a sensor in the sleeping state and in the idling state per time unit respectively, and E_{snd} and E_{rcv} be the amount of energy consumed by a sensor to send a message and to receive a message respectively. There are two possible cases that can occur during the time period T:

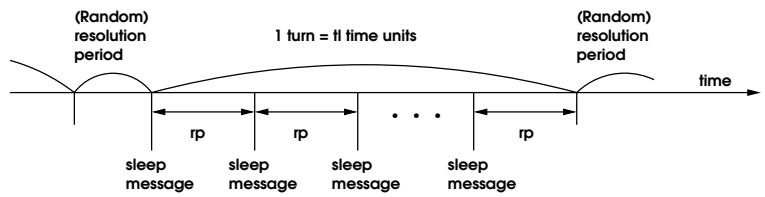

Fig. 3. A time period during protocol execution

i. *Case 1*: The sensors in the group do not execute the protocol, and remain in their idling states. The amount of energy consumed by n sensors in this case, E_{nop} is calculated as follows.

$$E_{nop} = E_{idl} * (tl + ravg) * n$$

ii. *Case 2*: The sensors in the group execute the protocol. The sentry stays in the idling state and sends $tl/ravg$ sleep messages for this time period. Each of the $(n-1)$ sleepers stays in the idling state for a resolution period, receives a sleep message, and sleeps for tl time units. Therefore, the amount of energy consumed by n sensors in this case, E_p is calculated as follows.

$$\begin{aligned} E_p = \quad & E_{idl} * (tl + ravg) + E_{snd} * (tl/ravg) \\ & +(n-1) * (E_{slp} * tl + E_{idl} * ravg + E_{rcv}) \end{aligned}$$

From the above analysis, we can estimate the effective number of the sensors N as follows:

$$N = \frac{E_{nop}}{E_p}$$

Table 1. Energy consumption of a sensor (in energy units)

E_{slp}	0.003 per time unit
E_{idl}	30 per time unit
E_{snd}	24.3 per message
E_{rcv}	9 per message

We present two figures, Figure 4 and 5, from the above formula for the four cases $tl = 30 * ravg$, $60 * ravg$, $90 * ravg$ and $120 * ravg$ in time units. In both of the figures, we use the values in Table 1 for E_{slp}, E_{idl}, E_{snd} and E_{rcv}. These values are computed using the energy consumption model in [10], under the assumption that a time unit is 100 milliseconds, and a time period taken for a sensor to send or receive a message is 30 milliseconds.

Figure 4 shows the relationship between the length of $ravg$ and N when n=4 and $5 \leq ravg \leq 1000$ in time units. If $ravg$ is 100 time units or more, then the value of N no longer depends on $ravg$. Similarly, when n=2 and 9, if $ravg$ is 100 time units or more, then the value of N no longer depends on $ravg$.

Figure 5 shows the relationship between n and N when $ravg$ = 100 time units. From Fig. 5, one can make two observations. First, tl does not have a strong effect on the value of N, especially when n is small. Second, N is closer to n when n is smaller. During a resolution period, all sensors in the group need to stay awake, and so the total amount of energy consumed by the sensors during this period is increased as n is increased. Thus, our protocol becomes more efficient as n is smaller.

In the real execution of the protocol, the current sentry can run out of the battery and die during its turn. Then there exists a time period where no sensor in the group is awake to perform the tasks assigned to the group. We call this time period a *gap*.

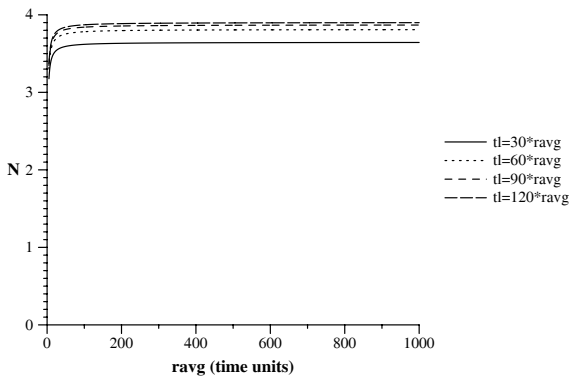

Fig. 4. N vs. $ravg$ when n=4

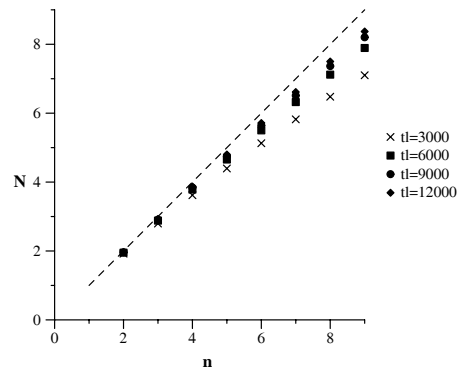

Fig. 5. N vs. n when $ravg$=100

We can estimate the total length of gaps over the lifetime of a group of n sensors from a simple formula. When the sentry dies during its turn, the average time period that no sensor in the group is awake is $tl/2$ (because the minimum time period is zero and the maximum time period is tl). Since $(n-1)$ sensors can die during their sentry turns, the total length of gaps over the lifetime is estimated as follows:

$$\frac{tl}{2} * (n-1)$$

The total length of gaps is relatively much smaller than the lifetime of the group. Note that the total length of gaps is related to the number of sensors in the group, not the lifetime of a regular sensor.

7 Simulation Results

We have developed a simulator that can simulate the execution of our sentry-sleeper protocol. This simulator simulates the behavior of a group of n sensors whose topology is fully connected and also allows us to configure the parameters of the protocol such as tl and $ravg$.

For the purpose of simulation, we have adopted the values in Table 1 as well as the following values:

- tl= 3000 time units
- $ravg$= 100 time units
- The amount of energy given to each sensor, at the beginning of simulation, is enough to keep that sensor in its idling state for 100000 time units.

We ran simulations of this protocol for the three cases n = 2, 4 and 9, and plotted the results in Figure 6 and 7. Figure 6 shows the effective number of the sensors N. Each circle mark represents the average effective number of the sensors over 100 simulations and each X mark represents the estimated effective number of the sensors. The effective number of the sensors in simulation is larger than that in estimation, because in simulation, sensors run out of their batteries and die over time and so the number of sensors in the group decreases over time. As discussed in Section 6, the protocol becomes more efficient as n is smaller.

Figure 7 shows the total length of gaps over the lifetime of a group of n sensors. Each circle mark represents the average total length of gaps over 100 simulations and each X mark represents the estimated total length of gaps.

From the simulation results, we show that the effective number of the sensors N is close to the number of sensors n in the group. That is, the group of n sensors can lengthen its lifetime around n times the lifetime of a regular sensor by adopting our protocol.

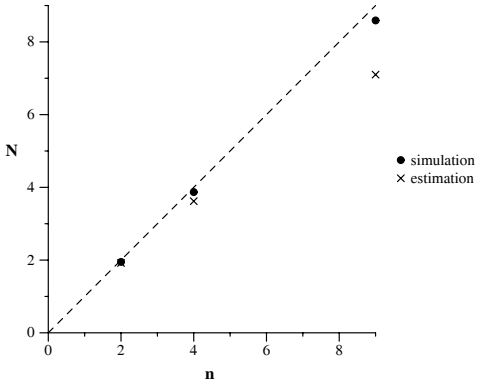

Fig. 6. The effective number of the sensors

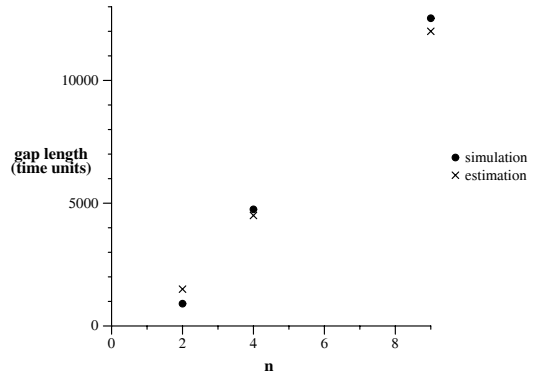

Fig. 7. The total length of gaps

8 Related Work

It is suggested in SBPM[8] to divide the sensors in a network into two sets, sentries and sleepers. Sentries stay awake, and provide basic communication services and coarse sensing services, while sleepers go to sleep to save their energy. When the sentries detect events, they can wake up the sleepers for more refined sensing. However, in SBPM, sentries are pre-selected and fixed. Moreover, a central computer decides when sleepers go to sleep. In GAF[18], all sensors that are equivalent in routing are identified using geographical location information. Then, only one sensor in a group of equivalent sensors stays awake and participates in routing, while the other sensors turn off their radios and go to sleep.

In Span[6] and TMPO[4], each sensor exchanges its neighbor information to compute which sensor joins a connected backbone in a network. Only sensors in the backbone participate in routing, while other sensors can go to sleep to save energy. In ASCENT[5], a sensor in a network keeps track of the number of its active neighbors and message loss rate, and joins a network topology only if the sensor becomes helpful. However, once a sensor enters the active state, it continues to be awake until it dies.

Other approaches to save energy in a sensor network have been proposed in [7], [13], [19], [12], [17], [20]. In LEACH[7], to reduce communication cost, each cluster-head collects data messages from the sensors in the cluster, and then compresses and forwards the messages to a base station. In STEM[12], a sensor in a monitoring state turns off the radio. If the sensor detects an event, it turns on the radio and wakes up other sensors if necessary.

Leader election protocols have been studied for single-hop single-channel radio networks in [11], [15], [16] and [14]. However, in general, these protocols assume that a station sending a message can simultaneously listen [11], [15], [14]. These protocols may not be useful in a sensor network, since generally a sensor cannot listen while the sensor is sending a message as in IEEE 802.11.

9 Concluding Remarks

In this paper, we described a sentry-sleeper protocol that can be used by a group of sensors to elect a new sentry at the beginning of each time period. Our protocol is based on the assumption that the sensors in a group have identical identities, and so the sensors compete to become a sentry purely based on randomization. We also showed that our protocol is self-stabilizing. The simulation results showed that a group of n sensors can lengthen its lifetime $1.95 * F$ for $n = 2$, $3.87 * F$ for $n = 4$ and $8.59 * F$ for $n = 9$, where F is the lifetime of a regular sensor.

Our protocol can be applied to a cluster-head election algorithm that balances energy load evenly among the nodes in a cluster. By adopting our protocol, the nodes in the cluster can elect a cluster-head

based on randomization without resorting to any identifiers, so some nodes with the highest identifier or the lowest identifier do not have an advantage over others.

Acknowledgment

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) Contract F33615-01-C-1901, and in part by three IBM Faculty Partnership Awards for the academic years of 2000-2003, and in part by the Texas Advanced Research Program, Texas Higher Education Coordinating Board, under Grant TARP 14-970823.

References

1. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks, Elsevier Science*, 38(4):393–422, 2002.
2. A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, and et al. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks (Elsevier), Special Issue on Military Communications Systems and Technologies*, 46(5):605–634, December 2004.
3. A. Arora and M. Gouda. Closure and Convergence: A Foundation of Fault-Tolerant Computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.
4. L. Bao and J. Garcia-Luna-Aceves. Topology Management in Ad Hoc Networks. In *Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, Annapolis, Maryland, June 2003.
5. A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In *Proceedings of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, June 23–27 2002.
6. B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom)*, pages 85–96, Rome, Italy, July 2001.
7. W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. In *Proceedings of Hawaiian Int'l Conf. on Systems Science*, January 2000.
8. Z. R. J. W. Hui and B. Krogh. Sentry-Based Power Management in Wireless Sensor Networks. In *The International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003.
9. A. Mainwaring, J. Polastre, R. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.
10. M. J. Miller and N. H. Vaidya. Minimizing Energy Consumption in Sensor Networks Using A Wakeup Radio. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'04)*, March 2004.
11. K. Nakano and S. Olariu. Randomized Leader Election Protocols in Radio Networks with No Collision Detection. In *International Symposium on Algorithms and Computation*, pages 362–373, 2000.
12. C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology Management for Sensor Networks: Exploiting Latency and Density. In *The ACM Symposium on Mobile Adhoc Networking and Computing (MOBIHOC 2002)*, Lausanne, Switzerland, June 9–11 2002.
13. S. Singh, M. Woo, and C. S. Raghavendra. Power-aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 181–190. ACM Press, 1998.
14. K. N. T. Hayashi and S. Olariu. Randomized Initialization Protocols for Packet Radio Networks. In *International Parallel Processing Symposium (IPPS) 1999, IEEE*, pages 544–548.
15. M. K. T. Jurdzinski and J. Zatopianski. Efficient Algorithms for Leader Election in Radio Networks. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 51–57. ACM Press, 2002.
16. M. K. T. Jurdzinski and J. Zatopianski. Weak Communication in Radio Networks. In *Euro-Par2002, Lecture Notes in Computer Science 2400, Springer-Verlag*, pages 965–972, 2002.
17. Y. Xu, J. Heidemann, and D. Estrin. Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Research Report 527, USC/Information Sciences Institute, October 2000.
18. Y. Xu, J. Heidemann, and D. Estrin. Geography-informed Energy Conservation for Ad-hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom)*, Rome, Italy, July 2001.
19. W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA, June 2002. IEEE.
20. M. Younis, M. Youssef, and K. Arisha. Energy-aware Management for Cluster-based Sensor Networks. *Computer Networks*, 43(5):649–668, 2003.