Smart Sleeping Policies for Energy Efficient Tracking in Sensor Networks

Jason A. Fuemmeler, *Member, IEEE*, and Venugopal V. Veeravalli, *Fellow, IEEE*

***Abstract*—We study the problem of tracking an object that is moving randomly through a dense network of wireless sensors. We assume that each sensor has a limited range for detecting the presence of the object, and that the network is sufficiently dense so that the sensors cover the area of interest. In order to conserve energy the sensors may be put into a sleep mode with a timer that determines the sleep duration. We assume that a sensor that is asleep cannot be communicated with or woken up. Thus, the sleep duration needs to be determined at the time the sensor goes to sleep based on all the information available to the sensor. The objective is to track the location of the object to within the accuracy of the range of the sensor. However, having sleeping sensors in the network could result in tracking errors, and hence there is a tradeoff between the energy savings and the tracking errors that result from the sleeping actions at the sensors. We consider the design of sleeping policies that optimize this tradeoff.**

***Index Terms*—Dynamic programming, Markov models, POMDP.**

I. Introduction

ADVANCES in technology are enabling the deployment of vast sensor networks through the mass production of cheap wireless sensor units with small batteries. Such sensor networks can be used in a variety of application areas. Our focus in this paper is on applications of sensor networks that involve *tracking*, e.g., surveillance, wildlife studies, environmental control, and health care.

We study the problem of tracking an object that is moving through a network of wireless sensors as shown in Fig. 1. Each sensor has a limited range for detecting the presence of the object being tracked, and the objective is to track the location of the object to within the accuracy of the range of the sensor. For such a tracking problem to be well-posed, we need to assume that the sensor field is sufficiently dense so that the sensors cover the entire area of interest. The object follows a random path through the sensor field whose statistics are assumed to be either known *a priori* or estimated online.

Manuscript received October 25, 2006; revised September 13, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Venkatesh Saligrama. This work was funded in part by a grant from the Motorola corporation as well as by a NSF Graduate Research Fellowship and ARO MURI Grant W911NF-06-1-0094.

The authors are with the Department of Electrical and Computer Engineering and Coordinated Science Lab, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: vvv@uiuc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2007.912265

The sensor nodes typically need to operate on limited energy budgets. In order to conserve energy, the sensors may be put into a sleep mode. The use of sleeping sensors in sensor networks for tracking has been studied in the past. It appears that there have been two primary approaches. The first has been to assume that sleeping sensors can be woken up by external means on an as-needed basis (see, e.g., [1]–[6]). Either the method used for this wakeup is left unspecified or it is assumed that there is some low-power wakeup radio at each sensor dedicated to this function. The second approach has involved modifications to power-save functions in MAC protocols for wireless ad hoc networks (see, e.g., [7]–[9]).

In this work, we wish to examine the fundamental theory of sleeping in sensor networks for tracking, as opposed to the design of protocols for this sleeping. We will assume that the wakeup channel approach is impractical given current sensor technology. In other words, we assume it is not feasible to design a receiver that requires negligible power for operation. Thus, we must consider alternatives to the wakeup channel approach. A straightforward approach is to have each sensor enter and exit the sleep mode using a fixed or a random duty cycle. A more intelligent, albeit more complicated, approach is to use information about the object trajectory that is available to the sensor from the network to determine the sleeping strategy. In particular, it is easy to see that the location of the object (if known) at the time when the sensor is put to sleep would be useful in determining the sleep duration of the sensor; the closer the object, the shorter the sleep duration should be. We take this latter approach in this paper in designing sleeping strategies for the sensors.

It is clear that having sleeping sensors in the network that cannot be woken up could result in tracking errors, and hence there is a tradeoff between the energy savings and the tracking error that results from the sleeping at the sensors. The sleeping policies at the sensors should be designed to optimize this tradeoff. In order to simplify the optimization problem, we assume that there is a central unit that controls the sensor network. All information about the object trajectory is stored at this central unit and is used to determine sleep times of sensors that come awake.

The main contributions of this paper are threefold. First, we develop a framework for optimizing the tradeoff between energy cost and tracking error without assuming the use of wakeup radios. The framework we develop is an example of a partially observable Markov decision process (POMDP). Second, although we are unable to find an optimal solution, we identify two suboptimal sleeping policies and also derive a lower bound on optimal performance. Third, we provide simulation results that characterize the performance of our suboptimal policies. The results

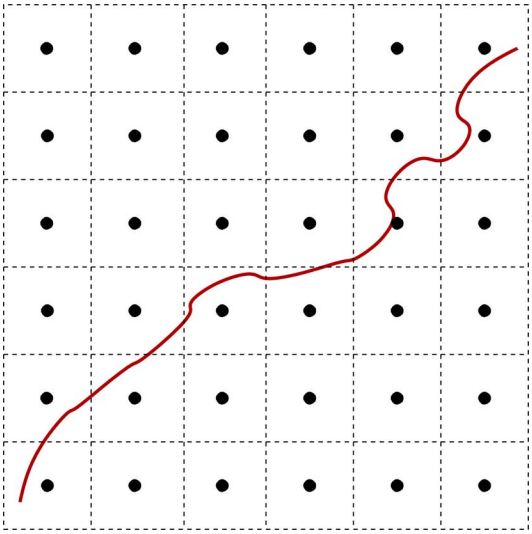

Fig. 1. Object tracking in a field of sensors.

indicate that our suboptimal policies perform well relative to optimal performance. Furthermore, our policies significantly outperform more naive policies that do not make use of information about the location of the object.

The remainder of this paper is organized as follows. In Section II, we describe the tracking problem in mathematical terms and define the optimization problem. We discuss the expected benefits of our approach in Section III. In Section IV, we outline a dynamic programming approach to finding the optimal solution. However, we find that the size of state space renders the optimization intractable for networks of more than a handful of sensors. We therefore propose some practical suboptimal solutions in Section V. In the course of deriving these solutions, we also develop a lower bound on optimal performance that allows us to characterize sleeping policy performance. In Section VI, we provide some numerical results that illustrate the efficacy of the proposed sleeping policies. We summarize and conclude in Section VII.

II. Problem Formulation

Notation

In this paper, we will use the following notational conventions.

- Scalars are written in lower case (e.g., c).
- Matrices are written in upper case (e.g., P).
- All vectors are row vectors and are written in bold face (e.g., $\boldsymbol{p}$).
- The vector $\boldsymbol{e}_i$ is a vector with a one in the ith position and zeros elsewhere.
- The vector $\boldsymbol{e}$ is a vector with a one in every position.
- Let $\boldsymbol{p}$ be a probability vector of length n, let $\mathcal{S} \subseteq \{1 \dots n\}$ be a set of integers, and suppose $p_i > 0$ for some $i \in S$. Then define $[\boldsymbol{p}]_{\mathcal{S}}$ to be a probability vector formed by setting all components p_i such that $i \notin S$ to zero and then normalizing the vector so that the sum of the components is 1.
- The indicator function is written as $\mathbb{1}_{\{\cdot\}}$.

A. POMDP Formulation

We consider a sensor network with n sensors. For simplicity, we assume that the sensing ranges of the sensors completely cover the region of interest with no overlap. In other words, the region can be divided into n cells with each cell corresponding to the sensing range of a particular sensor. Each sensor can be in one of two states: awake or asleep. A sensor in the awake state consumes more energy than one in the asleep state. However, object sensing can be performed only in the awake state.

The movement of the object to be tracked is described by a Markov chain whose state is the current location of the object to within the accuracy of a cell. However, we also append a special terminal state, denoted as $\mathcal{T}$, that occurs when the object leaves the network. Thus, there are $n+1$ possible states for the object and we will refer to the terminal state as both $\mathcal{T}$ and $n+1$. The statistics for the object movement are described by a $(n+1) \times (n+1)$ probability transition matrix P such that P_{ij} is the probability of the object being in state j at the next time step given that it is currently in state i. Since the problem remains in the terminal state once the object leaves the network, we can write P as

$$P = \left[\begin{array}{c|c} Q & 1 - Q\boldsymbol{e}^T \\ \hline 00\dots 0 & 1 \end{array}\right] \tag{1}$$

where Q is a $n \times n$ matrix. We assume there is a path from every state to the terminal state, which is equivalent to having $\lim_{k\to\infty} Q^k = 0$. Let b_k denote the location of the object at time k. We can describe the evolution of the object location stochastically as

$$b_{k+1} \sim \boldsymbol{e}_{b_k} P. \tag{2}$$

Our model for the object movement is simplistic, but does allow us to investigate sleeping policy design. The principles we develop later in this paper should extend to more complicated object movement models.

To provide a means for centralized control, we assume the presence of an extra node called the central controller. The central controller keeps track of the state of the network and assigns sleep times to sensors that are awake. In particular, each sensor that wakes up remains awake for one time unit during which the following actions are taken: i) if the object is within its range, the sensor detects the object and sends this information to the central unit, and ii) the sensor receives a new sleep time (which may equal zero) from the central controller. The sleep time input is used to initialize a timer at the sensor that is decremented by one time unit each time step. When this timer expires, the sensor wakes up. Since we assume that wakeup signals are impractical, this timer expiration is the only mechanism for waking a sensor. Let $r_{k,\ell}$ denote the value of the sleep timer of sensor ℓ at time k. We call the n-vector $\boldsymbol{r}_k$ the residual sleep times of the sensors at time k. Also, let $u_{k,\ell}$ denote the sleep time input supplied to sensor ℓ at time k. We can describe the evolution of the residual sleep times as

$$r_{k+1,\ell} = (r_{k,\ell} - 1)\mathbb{1}_{\{r_{k,\ell}>0\}} + u_{k,\ell}\mathbb{1}_{\{r_{k,\ell}=0\}}. \tag{3}$$

The first term on the right-hand side of this equation expresses that if the sensor is currently asleep (the sleep timer for the sensor is not zero), the sleep timer is decremented by 1. The second term expresses that if the sensor is currently awake (the sleep timer is zero), the sleep timer is reset to the current sleep time input for that sensor.

Based on (2) and (3), we see that we have a discrete-time dynamical model that describes our system, with control input $\boldsymbol{u}_k$ and exogenous input w_k. The *state* of the system at time k is described by $x_k = (b_k, \boldsymbol{r}_k)$ and the state evolution is defined in (2) and (3). Unfortunately, not all of x_k is known to the central unit at time k since b_k is known only if the object is currently being tracked. Thus, we have a dynamical system with incomplete (or partially observed) state information. If we denote the observation available to the central unit at time k by z_k, then $z_k = (s_k, \boldsymbol{r}_k)$, with

$$s_k = \begin{cases} b_k & \text{if } b_k \neq \mathcal{T} \text{ and } r_{k,b_k} = 0 \\ \mathcal{E} & \text{if } b_k \neq \mathcal{T} \text{ and } r_{k,b_k} > 0 \\ \mathcal{T} & \text{if } b_k = \mathcal{T} \end{cases} \tag{4}$$

where $\mathcal{E}$ denotes an unknown or "erasure" value. The total information available to the control unit at time k is given by

$$I_k = (z_0, \ldots, z_k, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_{k-1}) \tag{5}$$

with $I_0 = z_0$ denoting the initial (known) state of the system. The control input for sensor ℓ at time k is allowed to be a function of I_k, i.e.,

$$\boldsymbol{u}_k = \mu_k(I_k). \tag{6}$$

The vector-valued function μ_k is the sleeping policy at time k.

We now identify the two costs present in our tracking problem. The first is an energy cost of $c \in (0, 1]$ for each sensor that is awake. The second is a tracking cost of 1 for each time unit that the object is not observed. Note that a drawback of this definition of tracking cost is that cost may be incurred even if the object location is known through a process of elimination. For example, if every sensor other than the one where the object is located comes awake, a cost is still incurred. Our definition of tracking cost is used so that we will later be able to easily separate the problem into a set of simpler subproblems. Note that not much is lost through this simplification since we require only one additional sensor awake per unit time to maintain zero tracking errors. If the object leaves the network (b_k enters the terminal state), we assume the problem terminates and no further cost is incurred. The alternative would be to consider the fact that the object might return to the network, causing the problem to never terminate. This would necessitate a discounted cost or average cost formulation. Since we would like to use a total cost formulation, we make our assumption of termination. Given the above, we can write the total cost at time k as

$$g(x_k) = \mathbb{1}_{\{b_k \neq \mathcal{T}\}} \left(\mathbb{1}_{\{r_{k,b_k} > 0\}} + \sum_{\ell=1}^{n} c \mathbb{1}_{\{r_{k,\ell}=0\}} \right). \tag{7}$$

Note that c is the parameter used to tradeoff energy consumption and tracking errors.

The total cost (over a possibly infinite horizon trajectory) for the system is given by

$$J(I_0, \mu_0, \mu_1, \ldots) = \mathsf{E}\left[\sum_{k=1}^{\infty} g(x_k) \middle| I_0 \right]. \tag{8}$$

Since g is bounded by $(cn + 1)$ and the expected time till the object leaves the network is finite, the cost function J well-defined. The goal is to compute the solution to

$$J^*(I_0) = \min_{\mu_0, \mu_1, \ldots} J(I_0, \mu_0, \mu_1, \ldots). \tag{9}$$

The solution to this optimization problem for each value of c yields an optimal sleeping policy. The optimization problem falls under the framework of partially observable Markov decision process (POMDP).

B. Dealing With Partial Observability

Partial observability presents a problem since the information for decision-making at time k given in (5) is unbounded in memory. To remedy this, we seek a sufficient statistic for optimization that is bounded in memory. We see from (4) that s_k depends only on x_k, which in turn depends only on x_{k-1}, u_{k-1}, and w_{k-1}. It is a standard argument (see, e.g., [10]) that for such an observation model, a sufficient statistic is given by the probability distribution of the state x_k given I_k. Such a sufficient statistic is referred to as a *belief state* in the POMDP literature (e.g., see [11] and [12]). Since the residual sleep times portion of our state is observable, the sufficient statistic can be written as $v_k = (\boldsymbol{p}_k, \boldsymbol{r}_k)$, where $\boldsymbol{p}_k$ is a row vector of length $n + 1$ that denotes the probability distribution of b_k given I_k. Mathematically, we have

$$p_{k,\ell} = \mathsf{P}(b_k = \ell | I_k). \tag{10}$$

We can write the evolution of $\boldsymbol{p}_k$ as

$$\boldsymbol{p}_{k+1} = \boldsymbol{e}_{\mathcal{T}} \mathbb{1}_{\{b_{k+1}=\mathcal{T}\}} + \boldsymbol{e}_{b_{k+1}} \mathbb{1}_{\{r_{k+1,b_{k+1}}=0\}} + [\boldsymbol{p}_k P]_{\{j: r_{k+1,j}>0\}} \mathbb{1}_{\{r_{k+1,b_{k+1}}>0\}} \tag{11}$$

where $\boldsymbol{r}_{k+1}$ is defined through (3) and b_{k+1} (conditioned on $\boldsymbol{p}_k$) is distributed as

$$b_{k+1} \sim \boldsymbol{p}_k P. \tag{12}$$

To understand (11), note that if the object is observed at time $k + 1$, $\boldsymbol{p}_{k+1}$ becomes a point-mass distribution with all the probability mass concentrated at b_{k+1}. If the object is not observed, we eliminate all probability mass at sensors that are awake (since the object is known to not be at these locations) and renormalize. Thus, all information from observations is incorporated. One example of a distribution update is shown in Fig. 2.

We can now write our policy and cost function in terms of the sufficient statistic. The policy defined in (6) becomes

$$\boldsymbol{u}_k = \mu_k(v_k). \tag{13}$$

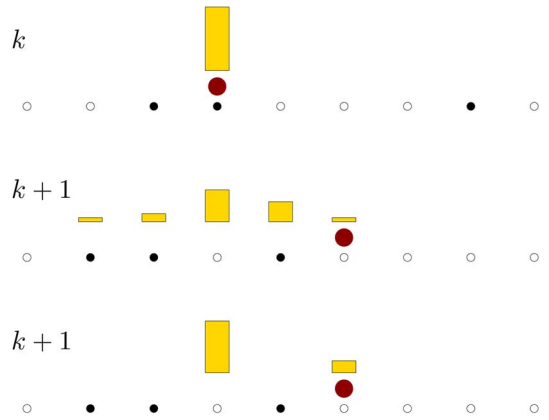

Fig. 2. Distribution update. In each of the three subfigures, the large circle represents the actual object location while the smaller circles represent sensors. A sensor whose circle has been darkened is a sensor that is awake, otherwise it is asleep. The bars in each subfigure represent the probability mass for each possible object location. The top subfigure give the probability distribution at time k, the middle subfigure the distribution at time $k+1$ before observations have been incorporated, and the bottom subfigure the distribution at time $k+1$ after the observations have been incorporated.

The total cost defined in (8) becomes

$$J(v_0, \mu_0, \mu_1, \ldots) = \mathsf{E}\left[\sum_{k=1}^{\infty} g(x_k) \middle| v_0\right] \tag{14}$$

and the optimal cost defined in (9) becomes

$$J^*(v_0) = \min_{\mu_0, \mu_1, \ldots} J(v_0, \mu_0, \mu_1, \ldots). \tag{15}$$

III. Performance Gains of our Approach

We comment now on what can be gained through our approach. For the purposes of comparison, we consider a sleeping policy that does not use information about the location of the object called the duty cycle scheme. In this sleeping policy, each sensor is awake for a fixed fraction π of the time slots. Whether the time slots where a particular sensor is awake are chosen deterministically or randomly is immaterial since the resultant performance is the same. For a duty cycle scheme, we see that in order to ensure zero tracking errors (an "Always Track" sleeping policy) we must use a value of $\pi = 1$, which means that every sensor is awake in every time slot. In contrast, since our scheme uses information about the location of the object, our Always Track policy will allow many sensors in the network to remain asleep since it will be known that the object could not be at those locations. We surmise that using location information in this fashion will result in a tradeoff curve between energy and tracking costs that is significantly better than those for a duty cycle scheme. These suppositions will be confirmed in Section VI.

We also note that our approach can result in better asymptotic behavior as the size of the network becomes large. Note that as n becomes large, the number of sensors awake per unit time for an Always Track duty cycle scheme grows as $O(n)$. In contrast, in the Appendix we show that when the movement of the object per time step is bounded, the number of sensors awake per unit time for our Always Track policy grows at most as $O(\log(n))$ for one-dimensional networks and at most as $O(\sqrt{n})$ for two-dimensional networks.

It may not be immediately obvious why the number of sensors awake per unit time might grow arbitrarily large with the size of the network, even if the number of possible locations an object can visit in the next time step is bounded. The reason is that when a sensor selects its sleep time it must make a worst-case assumption about the future movement of the object in order to ensure zero tracking errors. When this worst-case assumption is not realized, the sensor comes awake unnecessarily. This is a result of not allowing communication with sleeping sensors.

IV. Optimal Solution Via DP

Having formulated our optimization problem in terms of the sufficient statistic v_k, we seek a solution using the tools of dynamic programming. Because of the stationarity of the underlying Markov chain, it can be shown via standard arguments (see, e.g., [10]) that there exists a stationary optimal policy for our problem (i.e., $\mu_0 = \mu_1 = \cdots = \mu^*$). Such a policy and the optimal cost J^* can be found by solving the Bellman equation given as

$$J(v) = \min_{\mu} \mathsf{E}\left[g(x_1) + J(v_1) | v_0 = v, \boldsymbol{u}_0 = \mu(v)\right] \tag{16}$$

with μ^* being the minimizing value of μ in this equation. Note that there are multiple functions J^* that satisfy this equation since adding a constant to a particular J^* yields another solution. We are interested in J^* such that $J^*(v) = 0$ when $v = (\boldsymbol{e}_{\mathcal{T}}, \boldsymbol{r})$.

Due to the complexity of the expressions involved, we are unable to find an analytical solution to (16). The remaining alternative is to solve the equation using an iterative technique such as value iteration or policy iteration. Let us consider value iteration (also known as successive approximation). In this method, we start with some initial estimate for J^* (e.g., $J^* = 0$) and repeatedly apply the transformation defined by the right-hand side of (16) until the sequence of cost functions we obtain converges. It is possible to show theoretically that value iteration converges for this problem.

However, there are some practical issues to consider. The state space for this problem consists of $\boldsymbol{p}_k$, which is uncountably infinite, and $\boldsymbol{r}_k$, which is countably infinite. Thus, we must either have an analytical solution for the cost function at each iteration (which we cannot find due to the aforementioned complexity of the problem) or we must quantize and truncate the state space so that there are a finite number of states. Of course, restricting the infinite state space to a finite state space will lead to some loss of optimality.

Unfortunately, even with the restriction to a finite state space, the complexity of value iteration remains intractable except for the most trivial cases. This is because the state space grows exponentially with the number of sensors. For example, even with seven sensors, a maximum sleep time of 10, and a probability mass function quantized to multiples of 0.1, there are about 10^9 possible states v_k. We conclude that we cannot find an optimal solution to our problem and that we need other approaches to finding near-optimal solutions. We identify other approaches in Sections V and VI.

V. Suboptimal Solutions

A. General Approach

Much of the complexity of our problem stems from the complicated evolution of $\boldsymbol{p}_k$ given in (11). In deriving suboptimal solutions to our problem, we will make assumptions about the observations that will be available in the future. These assumptions will allow us to simplify the evolution in (11) considerably. In fact, the evolution of $\boldsymbol{p}_k$ will no longer be affected by the sleeping actions of the sensors. Furthermore, each sensor will only be able to affect the energy and tracking costs that occur at its location. The result is that the optimization problem easily separates into n simpler problems, one for each sensor. In each of these simpler problems, we will be able to eliminate the residual sleep times $\boldsymbol{r}_k$ from the state since the only times of interest will be those when the sensor comes awake. It will then be possible to solve each of the n simpler problems to find a cost function and policy. The cost function for the original problem is then the sum of the per-sensor cost functions while the policy for the original problem is the per-sensor policies applied in parallel.

The assumptions we make will be inaccurate. However, the usefulness of our assumptions must be measured in terms of how well the resultant solutions approximate optimal performance. Of course, we have no idea as yet what optimal performance may be. Fortunately, in the course of our derivations we will obtain a lower bound on optimal performance that will be useful in later performance analysis.

We now define some additional notation. Under each assumption, we will refer to an optimal cost and policy as J^* and μ^*. These functions may written in terms of the state $(J^*(v), \mu^*(v))$ or the component parts of the state $(J^*(\boldsymbol{p},\boldsymbol{r}), \mu^*(\boldsymbol{p},\boldsymbol{r}))$. The optimal cost and policy for the simpler problem for sensor ℓ will be $J^{*(\ell)}$ and $\mu^{*(\ell)}$ respectively. These functions are written in terms of the $\boldsymbol{p}$ portion of the state $(J^{*(\ell)}(\boldsymbol{p}), \mu^{*(\ell)}(\boldsymbol{p}))$.

B. FCR Solution

To generate the first cost reduction (FCR) solution, we assume that we will have no future observations. In other words, we are replacing (11) with

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k P. \tag{17}$$

Note that this does not mean that it will be impossible to track the object; we are simply making an assumption about the future state evolution in order to generate a sleeping policy. As shown in the Appendix, we can solve the equation

$$J^{(\ell)}(\boldsymbol{p}) = \min_u \left(\sum_{j=1}^{u} \left[\boldsymbol{p}P^j\right]_\ell + \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i + J^{(\ell)}(\boldsymbol{p}P^{u+1}) \right) \tag{18}$$

to find the cost function and policy for sensor ℓ. Note that the three terms inside the minimization represent the tracking cost, the energy cost, and the future cost respectively given a sleep time of u. Thus, (18) is a Bellman equation for the per-sensor problem under the assumption of no future observations.

It is easy to verify that

$$J^{*(\ell)}(\boldsymbol{p}) = \sum_{j=1}^{\infty} \min \left\{ \left[\boldsymbol{p}P^j\right]_\ell, \sum_{i=1}^{n} c \left[\boldsymbol{p}P^j\right]_i \right\} \tag{19}$$

is indeed a solution to (18). In other words, at each time step we incur a cost that is the minimum of the expected tracking cost at sensor ℓ and the expected energy cost at sensor ℓ. Define the set $\mathcal{U}(\boldsymbol{p})$ as

$$\mathcal{U}(\boldsymbol{p}) = \left\{ u : \left[\boldsymbol{p}P^{u+1}\right]_\ell \geq \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i \right\}. \tag{20}$$

It is then easily verified that the per-sensor policy $\mu^{*(\ell)}(\boldsymbol{p})$ has the form

$$\mu^{*(\ell)}(\boldsymbol{p}) = \min_{u \in \mathcal{U}(\boldsymbol{p})} u. \tag{21}$$

More simply, the policy is to come awake at the first time such that the expected tracking cost exceeds the expected energy cost. This is why this solution is called the first cost reduction solution.

C. Q_{MDP} Solution

In the POMDP literature (e.g., see [11] and [12]), a Q_{MDP} solution is one in which it is assumed that the partially observed state becomes fully known after a control input has been chosen. In our problem, this means assuming that we will have perfect future observations, i.e., the location of the object will be known in the future. In other words, we are replacing (11) with

$$\boldsymbol{p}_{k+1} = \boldsymbol{e}_{b_{k+1}}. \tag{22}$$

Note that this does not mean that it will be impossible to incur tracking errors; we are simply making an assumption about the future state evolution in order to generate a sleeping policy. As shown in the Appendix, we can solve the equation

$$J^{(\ell)}(\boldsymbol{p}) = \min_u \left(\sum_{j=1}^{u} \left[\boldsymbol{p}P^j\right]_\ell + \sum_{i=1}^{n} c \left[\boldsymbol{p}P^{u+1}\right]_i + \sum_{i=1}^{n} \left[\boldsymbol{p}P^{u+1}\right]_i J^{(\ell)}(\boldsymbol{e}_i) \right) \tag{23}$$

to find the cost function and policy for sensor ℓ. Note that the three terms inside the minimization represent the tracking cost, the energy cost, and the future cost respectively given a sleep time of u. Thus, (23) is a Bellman equation for the per-sensor problem under the assumption of perfect future observations.

Unfortunately, we are unable to find an analytical solution to (23). However, note that if we can solve (23) for $\boldsymbol{p} = \boldsymbol{e}_b$ for all $b \in \{1, \ldots, n\}$, then it is straightforward to find the solution for all other values of $\boldsymbol{p}$. We therefore concern ourselves with finding values of $J^{*(\ell)}(\boldsymbol{e}_b)$ and $\mu^{*(\ell)}(\boldsymbol{e}_b)$ that satisfy (23) for all $b \in \{1, \ldots, n\}$. This can be achieved through policy iteration. Policy iteration proceeds as follows.

1) Set $\mu^{(\ell)}(\boldsymbol{e}_b) = \infty$ and $J^{(\ell)}(\boldsymbol{e}_b) = \sum_{j=1}^{\infty} \left[\boldsymbol{p}P^j\right]_\ell$ for all $b \in \{1,\ldots,n\}$.
2) Compute a new value for $\mu^{(\ell)}(\boldsymbol{e}_b)$ as

$$\mu^{(\ell)}(\boldsymbol{e}_b) = \arg\min_u \left(\sum_{j=1}^{u} \left[\boldsymbol{e}_b P^j\right]_\ell + \sum_{i=1}^{n} c\left[\boldsymbol{e}_b P^{u+1}\right]_i + \sum_{i=1}^{n} \left[\boldsymbol{e}_b P^{u+1}\right]_i J^{(\ell)}(\boldsymbol{e}_i) \right) \quad (24)$$

for all $b \in \{1,\ldots,n\}$, with the additional caveat that if there are multiple minimizing values of u, the smallest should be chosen.
3) Solve a set of linear equations to find new values for $J^{(\ell)}(\boldsymbol{e}_b)$ for all $b \in \{1,\ldots,n\}$. Using the shorthand $u_b = \mu^{(\ell)}(\boldsymbol{e}_b)$, the linear equations are given as

$$J^{(\ell)}(\boldsymbol{e}_b) = \sum_{j=1}^{u_b} \left[\boldsymbol{e}_b P^j\right]_\ell + \sum_{i=1}^{n} c\left[\boldsymbol{e}_b P^{u_b}\right]_i + \sum_{i=1}^{n} \left[\boldsymbol{e}_b P^{u_b}\right]_i J^{(\ell)}(\boldsymbol{e}_i) \quad (25)$$

for all $b \in \{1,\ldots,n\}$.
4) If $\mu^{(\ell)}(\boldsymbol{e}_b)$ is different from the previous value for $\mu^{(\ell)}(\boldsymbol{e}_b)$ for at least one value of b, return to Step 1. Otherwise, terminate and set $J^{*(\ell)} = J^{(\ell)}$ (of course, we will then have that $\mu^{*(\ell)} = \mu^{(\ell)}$).

There are portions of this algorithm that warrant further discussion. Note that the minimization in Step 2, although well defined, is nontrivial since we are minimizing a non-convex function over a countably infinite set. Although we could restrict the set of sleep times to a finite set, this could lead to loss in optimality. A better approach is to start with an initial guess of $u = \infty$ for the minimizing u and a minimum value equal to the limit of the function to be minimized as $u \to \infty$. We then start at $u = 0$ and search for a minimum by repeatedly increasing u by 1. At each step, we can compute a lower bound on the function to be minimized over all values of u' such that $u' \geq u$. If the minimum found so far is less than or equal to this lower bound, then a global minimum has been found and the search terminates. This procedure will work as long as the lower bound we compute becomes appropriately tight as $u \to \infty$. It is frequently possible to find such lower bounds, so this is an attractive approach. Turning our attention to Step 3 of the policy iteration algorithm, it is easy to establish that the set of linear equations described does have a unique and nonnegative solution. It is also clear that if the algorithm terminates, a solution to the Bellman equation has been found. Although we can apply policy iteration to any particular tracking problem and hope for termination, we would like to know if there are any conditions under which termination is assured. It can be shown that one such condition is for Q (the previously defined submatrix of P) to be primitive, i.e., a square matrix with nonnegative elements that has a unique maximal eigenvalue (see [13]). Note that constructing the lower bounds discussed for Step 2 of the policy iteration algorithm is made relatively simple if Q is primitive because the cost function to be minimized becomes an exponential function asymptotically.

Note that for the Q_{MDP} solution, we are assuming more information than is actually available. Thus, the cost function obtained under the Q_{MDP} is a lower bound on optimal performance. We will use this lower bound when we present our numerical results.

D. Point Mass Approximations

The suboptimal policies derived in the preceding sections are considerably easier to compute than the optimal policy and can be computed online after some initial offline computation has been completed. However, such online computation requires sufficient processing power and could introduce delays. It would be convenient if the suboptimal μ^* could be precomputed and stored either at the central controller or distributed across the sensors themselves. The latter option is particularly attractive since it allows for distributed implementation. But the set of possible distributions $\boldsymbol{p}$ is potentially quite large—even if quantization is performed—and could make the storage requirements prohibitive.

To make the storage requirements feasible, we consider approximating $\boldsymbol{p}$ with a point mass distribution. The number of sleep times to be stored is then only n per sensor. We consider two options for the placement of the unit point mass when computing the sleep time for sensor ℓ: i) the centroid of $\boldsymbol{p}$, and ii) the nearest point to sensor ℓ on the support of $\boldsymbol{p}$. Note that the latter option allows for the implementation of policies without detailed information about the statistics of the random walk—only the support of the random walk is required.

VI. Numerical Results

In this section, we give some simulation results that illustrate the performance of the policies we derived in previous sections. We begin with simulation results for one-dimensional sensor networks. In each simulation run, the object was initially placed at the center of the network and the location of the object was made known to each sensor. A simulation run concluded when the object left the network. The results of many simulation runs were then averaged to compute an average tracking cost and an average energy cost. To allow for easier interpretation of our results, we then normalized our costs by dividing by the *expected* time the object spends in the network. We refer to these normalized costs as costs per unit time, even though the true costs per unit time would use the *actual* times the object spent in the network (the difference between the two was found to be small).

We will give results for two different one-dimensional networks. Network A is a one-dimensional network with 41 sensors where the object moves with equal probability either one to the left or one to the right in each time step. Thus, the Q matrix for Network A can be written as a 41×41 matrix with zeros everywhere except for values of $(1/2)$ on the diagonals just above and below the main diagonal. Network B is a one-dimensional network with 61 sensors where the object can move anywhere from three positions to the left to three positions to the right at each time step, with the object movement being uniformly distributed on these seven positions. Thus, the Q matrix for Network B can be written as a 61×61 matrix with zeros everywhere except for values of $(1/7)$ on the seven diagonals in the middle of the matrix.

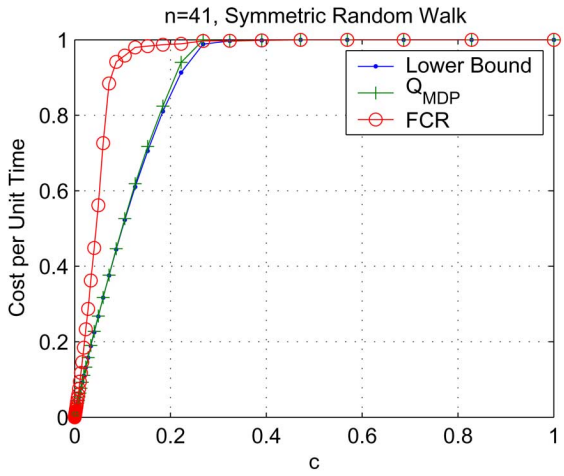

Fig. 3. Cost per unit time comparisons as a function c for Network A.

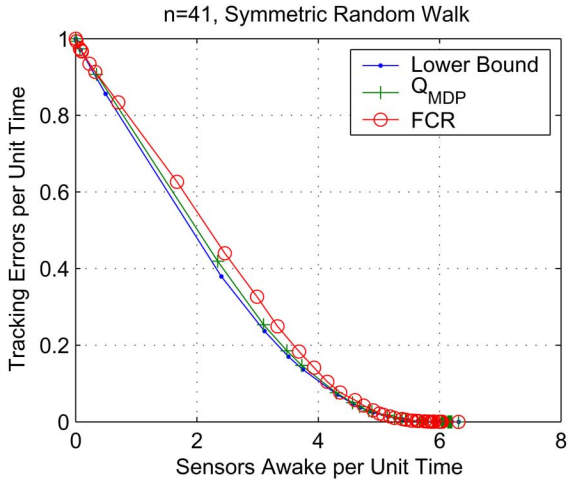

Fig. 5. Tradeoff curves for Network A.

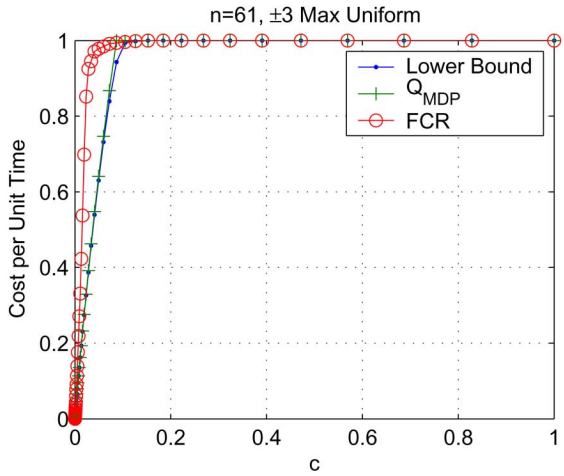

Fig. 4. Cost per unit time comparisons as a function c for Network B.

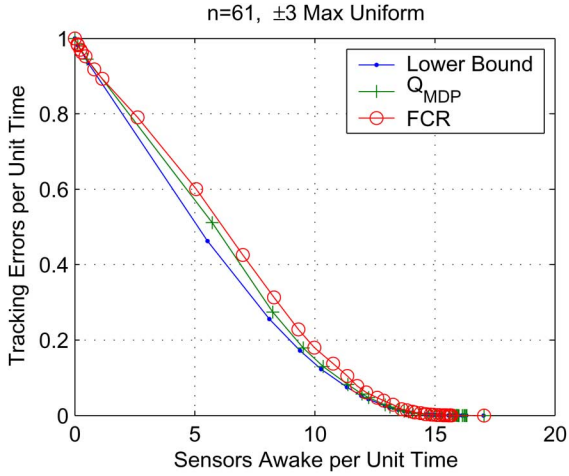

Fig. 6. Tradeoff curves for Network B.

In Figs. 3 and 4 we plot cost curves as a function of c for Networks A and B, respectively. In each figure three curves are shown. The first curve is the lower bound on optimal performance discussed in Section V-C. The second and third curves are the costs for the Q_{MDP} and FCR policies, respectively. From these data we can see that the Q_{MDP} policy consistently outperforms the FCR policy. Moreover, the cost for the Q_{MDP} policy is extremely close to the lower bound on optimal performance except at a few data points. We therefore suspect that the Q_{MDP} policy is a near-optimal policy.

In Figs. 5 and 6, we now examine the tradeoff curves between energy cost and tracking cost for Networks A and B respectively. From these data, we again see that the Q_{MDP} policy outperforms the FCR policy, although the difference does not appear as large. This does not contradict our previous results since it is possible that if one policy achieves certain values of energy and tracking costs at a particular value of c, it is possible for another policy to achieve these same energy and tracking costs at a somewhat different value of c. Note that the difference between the performance of the Q_{MDP} policy and the lower bound on optimal performance becomes small as the number of tracking errors becomes small. This makes sense since when there are few tracking errors, the Q_{MDP} assumption (that we will know the object location in the future) becomes realistic.

For the moment, consider the duty cycle scheme discussed earlier where each sensor is awake in a fraction π of the time slots. As π is varied, we achieve a tradeoff curve that is a straight line between the points (0, 1) and $(n, 0)$ (where n is the appropriate number of sensors) in the coordinate systems used in Figs. 5 and 6. When compared with this policy, the schemes we have proposed result in significant improvement.

In Fig. 7, we explore the impact of using the point mass approximations discussed in Section V-D on the performance of the Q_{MDP} policy for Network A. Four curves are shown in the figure. The first two are the lower bound and Q_{MDP} tradeoff curves already seen. The third and fourth curves are the tradeoff curves for the Q_{MDP} policy using the centroid and nearest point point mass approximations, respectively. It can be seen that there is indeed some loss in performance when using point mass approximations, but this loss becomes small as the number of tracking errors becomes small. This makes sense since when tracking errors are infrequent, the object location is