Managing the Mobility of a Mobile Sensor Network Using Network Dynamics

Ke Ma, Yanyong Zhang, *Member*, *IEEE*, and Wade Trappe, *Member*, *IEEE*

Abstract—It has been discussed in the literature that the mobility of a mobile sensor network (MSN) can be used to improve its sensing coverage. How the mobility can efficiently be managed toward a better coverage, however, remains unanswered. In this paper, motivated by classical dynamics that study the movement of objects, we propose the concept of network dynamics and define the associated potential functions that capture the operational goals, as well as the environment of an MSN. We find that in managing the mobility of an MSN, Newton's laws of motion in classical dynamics are insufficient, for they introduce oscillations into the movement of sensor nodes. Instead, in network dynamics, the laws of motion are formulated using the steepest descent method in optimization. Based on the network dynamics model, we first devise a parallel and distributed algorithm (Parallel and Distributed Network Dynamics (PDND)) that runs on each sensor node to guide its movement. PDND then turns sensor nodes into autonomous entities that are capable of adjusting their locations according to the operational goals and environmental changes. After that, we formally prove the convergence of PDND. Finally, we apply PDND in three applications to demonstrate its effectiveness.

Index Terms—Mobile sensor networks, coverage, potential fields, jamming attack.

1 Introduction

Sensor networks usually consist of stationary sensor nodes. Sometimes, deploying such a stationary sensor network and maintaining its sensing coverage could be a daunting task. To illustrate the point, imagine deploying a stationary sensor network over a battlefield. Even though advanced tools like airplanes are available to make the deployment safer and easier, various factors such as winds and obstacles are very likely to introduce coverage holes, regardless of how many sensor nodes are dropped. Taking one step further, even if a perfect coverage can be achieved initially, events such as node failures and malicious attacks will certainly degrade the network coverage as time evolves. As a result, there is an urgent need for sensor nodes to be equipped with mobility so that they can autonomously discover and repair coverage holes. Another scenario in which mobile sensor nodes are desired is the application of monitoring a moving object that travels over a large area. In the two applications mentioned above, as well as many application scenarios that are looming the horizon, engaging a mobile sensor network (MSN) that is capable of changing its layout and position is more desirable. As a matter of fact, such an MSN has already been put into use [1], where a fleet of undersea mobile sensor nodes coordinate and collect measurements of the ocean without human intervention.

Since the mobility of sensor nodes is of great importance, it is crucial to formulate laws that govern the mobility. Inspired by classical dynamics that study the movement of objects, in [2], the concept of artificial potential fields was first introduced to guide the movement of a robot. This method was employed in [3], in which a robot navigated itself to a particular location through a field filled with obstacles. The robot first builds an artificial potential field, assuming that it receives an attractive force from the destination and a repulsive force from each obstacle. After that, the robot uses Newton's laws of motion in classical dynamics to determine its movement. Later in the literature, the artificial-potential-field approach was used to manage the mobility of an MSN to improve its sensing coverage in [4], [5], [6], [7], [8], [9], [10], [11], and [12]. In these studies, sensor nodes not only receive forces from the surrounding environment, but also receive forces from one another. In most studies, however, Newton's laws of motion were adopted for the simple reason that the stopping of a node's movement can be easily guaranteed by introducing friction forces. Despite the simplicity, Newton's laws suffer inefficiency, for they may cause oscillations. To visualize this, simply imagine how a pendulum comes to a stop. In the case of sensor mobility management, oscillations should be avoided, as they unnecessarily slow down deployment and waste energy. However, as long as Newton's laws of motion are used, oscillations are inherent and hard to be eliminated.

To address this challenge, we propose the concept of network dynamics in this paper, in which artificial potential functions are defined to capture the operational goals and the environmental constraints of an MSN, and the laws of motion are formulated using the steepest descent method in optimization. Using network dynamics, we can transform a deployment problem into an optimization problem with the help of potential energy: the movement of sensor nodes should be governed to minimize the potential energy of an MSN as quickly as possible. In addition, once the potential energy reaches its minimum, all sensor nodes should cease movement immediately. To the best of our knowledge, this is the first attempt to treat such a deployment problem in such a manner. Based on the model of network dynamics, we propose a parallel and distributed algorithm (Parallel and Distributed Network Dynamics (PDND)), in which

• *The authors are with the Wireless Information Network Laboratory (WINLAB), Rutgers University, 73 Brett Rd., Piscataway, NJ 08854. E-mail: {kemaru, yyzhang, trappe}@winlab.rutgers.edu.*

Manuscript received 27 Nov. 2006; revised 5 Mar. 2007; accepted 10 Apr. 2007; published online 9 May 2007.
Recommended for acceptance by Y. Pan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0382-1106.
Digital Object Identifier no. 10.1109/TPDS.2007.1113.

sensor nodes periodically broadcast their location information, calculate the virtual forces that are applied to them, and relocate themselves accordingly. In addition, we formally prove the convergence of PDND under realistic assumptions.

To demonstrate the generality and effectiveness of PDND, we study its applications in three important domains. The first application deals with deploying sensor nodes to monitor an area. We propose a practical deployment strategy here: We first dump all sensor nodes within a small area, or randomly drop them all over the whole area, and then let them autonomously adjust their locations to improve the overall sensing coverage by following PDND. Our simulation results show that PDND fulfills this task efficiently under a range of circumstances. Besides, we also propose a variation of PDND (PDND2) that can shorten the deployment time and total moving distance, without noticeably sacrificing the overall coverage. Finally, we compare our algorithms with the Lloyd algorithm, as proposed in [13], and show that our algorithms exhibit better performance. The second application migrates an MSN to chase a moving target. Here, the target, once detected, constantly exerts an attractive force to sensor nodes, and as a consequence of the attractive force, sensor nodes can follow the target while it moves. Last, the third application repairs network holes that are caused by malicious attacks like jamming. A jammer can cause communication holes in the network by blocking the wireless channel. In the presence of jamming, we first propose an effective retreat strategy so that all jammed nodes can escape the jammed area. After escaping, these nodes will follow PDND to uniformly disperse into the rest of the network. More importantly, PDND prevents the network from being partitioned by a jammer that sweeps through the network, as nodes will automatically fill the holes after the jammer leaves.

This paper is organized as follows: We begin the discussion of our proposed model the network dynamics in Section 2. In Section 3, we present a parallel and distributed algorithm that can be used to implement network dynamics in practice, and its convergence is formally proved in Section 4. In Sections 5, 6, and 7, we examine three applications of network dynamics. The first application, presented in Section 5, focuses on using network dynamics to control the sensing coverage of an MSN over a particular region. The second application in Section 6 studies the feasibility of applying network dynamics to enable an MSN to chase a moving target. In Section 7, we apply network dynamics to achieve a robust spatial retreat strategy that maintains desirable network connectivity in the presence of jamming attacks. Finally, in Sections 8 and 9, we present related works and concluding remarks, respectively.

2 Network Dynamics

An MSN in this paper is a collection of sensor nodes that have mobility, such as unmanned vehicles equipped with various sensors. These sensor nodes are assumed to run on powerful batteries. For example, the unmanned undersea vehicles in [14] have the ability to travel hundreds of miles. An MSN may be viewed as a dynamical system: the positions of sensor nodes change as time goes by. The dynamics of classical mechanical systems are described via underlying laws of motion and laws of force between objects [15]. We propose to apply the concept of forces, the corresponding notion of potential energy, and the laws of motion to manage the movement of an MSN. Appropriately defining potential functions and laws of motion yields a general framework that allows people to optimally use mobility to govern the operations of an MSN.

2.1 Classical Dynamics and MSNs

We shall look at an MSN as a dynamical system of N devices subject to the laws of classical mechanics. Each device will be able to communicate with other devices in its radio range through some wireless communication protocol. Since the devices are mobile, we will associate with each device i a position vector $\mathbf{p}_i$ and a momentum vector $\mathbf{q}_i$. We will, without loss of generality, assume that all devices are located in two dimensions and that for each device, both $\mathbf{p}_i$ and $\mathbf{q}_i$ are 2D vectors. For now, since we are looking at the system as a mechanical system, we will arbitrarily assign each network device a mass of 1; thus, momentum and velocity are equivalent. Later in the discussion of network dynamics, the concept of mass is no longer useful and, hence, removed.

N-body dynamical systems appear commonly in physics, and the dynamical relationship between the position and momentum of these N bodies evolve based upon Newton's second law, which describes the motion of a body in the presence of a field of force.

The forces acting upon a conservative dynamical system arise as the negative gradient of the potential energy function U. This potential function may be comprised of two components: *external* U_{ext} and *internal* U_{int}. External potentials arise from externally applied forces, whereas internal potentials correspond to the attractive or repulsive forces between the bodies of the system. Typically, the internal potentials are restricted to two-body interactions such as the gravitational pull between two objects.

We may collectively refer to the position vector of each of the N bodies by a $2N$-dimensional position vector $\mathbf{p} = [\mathbf{p}_1, \cdots, \mathbf{p}_N]$ and similarly for the momentum vector $\mathbf{q}$. Hence, the potential energy may be viewed as

$$U(\mathbf{p}) = \sum_i U_{ext}(\mathbf{p}_i) + \sum_i \sum_{j>i} U_{int}(\mathbf{p}_i, \mathbf{p}_j). \tag{1}$$

Newton's classical equations of motions give

$$\frac{d}{dt}\mathbf{q}_i = \mathbf{f}_i, \qquad \mathbf{f}_i = -\nabla_i U(\mathbf{p}), \tag{2}$$

where ∇_i is the gradient at the ith body's position $\mathbf{p}_i$. Applying the relationship between position and velocity $\frac{d}{dt}\mathbf{p}_i = \mathbf{q}_i$ yields a set of coupled differential equations describing the dynamics of the N bodies.

2.2 Potential Functions for Network Dynamics

The mobility of objects is governed by the description of the potential function $U(\mathbf{p})$, which, in turn, yields force and causes motion. We therefore need to define potential functions that suitably capture the need for causing mobile devices to move. We will do this in two parts: we first describe possible external potential functions $U_{ext}(\mathbf{p})$ and then describe internal (that is, pairwise) potential functions $U_{int}(\mathbf{p}_i, \mathbf{p}_j)$.

External potential functions may be viewed as representing factors coming from the environment that should

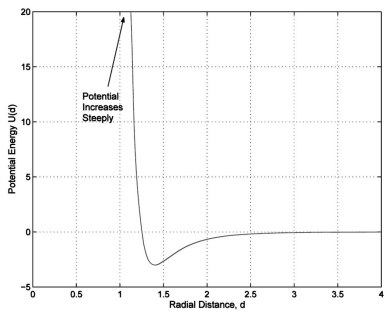

Fig. 1. The Lennard-Jones potential.

influence the motion of an MSN. For example, suppose that an MSN has been deployed to perform monitoring functions. It may be that there are regions of the network that deserve more attention than other regions and, therefore, we might wish for mobile devices to be attracted to these regions. As another example, it might be desirable for a set of monitoring devices to migrate, perhaps to monitor a moving asset or perhaps to sweep through a coverage area.

Internal potentials typically correspond to either attractive or repulsive pairwise interactions between entities. In general, we desire potential functions that are capable of dispersing mobile sensors, without causing them to separate too greatly from each other. These potential functions, often known as dispersive potentials, involve repulsive and attractive components. An example of such a potential function in the physical world is the Lennard-Jones potential from thermophysics [16]:

$$U(\mathbf{p}_i, \mathbf{p}_j) = 4\epsilon \left[\left(\frac{\sigma}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right)^{12} - \left(\frac{\sigma}{\|\mathbf{p}_i - \mathbf{p}_j\|} \right)^{6} \right], \quad (3)$$

where σ describes the radial intercept and ϵ governs the *well depth*, as depicted in Fig. 1. Readers should keep in mind that in managing an MSN, people can choose or define whatever potential functions that they feel fit their needs best. As a result, when a potential function observed in the physical world is chosen, there is no need to bring its physical conditions and constraints in.

2.3 Ideal Simulation Framework and Convergence

In the following, we will examine the natural discretization of Newton's equations of motion and argue that such a formulation leads to mobile devices performing extra mobility. To address this concern, in network dynamics, we propose to formulate the equations of motion by using the steepest descent method in optimization.

Suppose we have a system of N objects and consider the evolution of the N objects' positions in time k. Here, we represent time discretely by breaking time into intervals of length Δt. A typical discretization involves updating the ith object's position via

$$\mathbf{p}_i(k+1) = \mathbf{p}_i(k) + \mu_p \mathbf{q}_i(k), \quad (4)$$

$$\mathbf{q}_i(k+1) = \mathbf{q}_i(k) + \mu_q \mathbf{f}_i(k), \quad (5)$$

where μ_p and μ_q are step sizes, and the force $\mathbf{f}_i(k) = -\nabla_i U(\mathbf{p}(k))$ may be determined at each time step. The time evolution of the MSN's motion is then determined by letting the above system evolve.

The classical equations of motion are second order in time, which means that the coupled equations tie in position, velocity, and acceleration. This is suitable for mechanics but results in undesirable properties from the point of view of network dynamics. Specifically, the coupling between position, velocity, and acceleration implies that it is quite likely that even when the system has reached a configuration with minimal potential energy, the N bodies will still have velocity and, hence, kinetic energy. As a result, the system will escape its desirable configuration and have to compensate later by applying forces in the opposite direction. To visualize this, simply consider the classical pendulum, in which the pendulum achieves its minimal potential at the base of the trajectory, and the momentum causes the pendulum to swing past and increase potential energy. The increased potential causes the pendulum to reverse direction and oscillate around the point of minimal potential.

From the viewpoint of network dynamics, this behavior is undesirable, as it means that the mobile device must waste movement and, hence, power and time, compensating for momentum. We, therefore, would like to modify the equations of motion to remove the effect of momentum. This may be accomplished by making the equations first order and have the force affect the distance traveled. For example, we may simply create an iterative system:

$$\mathbf{p}(k+1) = \mathbf{p}(k) - \gamma(k) \nabla U(\mathbf{p}(k)). \quad (6)$$

Examining this iteration shows that we are simply making a step of size $\gamma(k)$ in the direction of the steepest descent to minimize the potential function U. Now, once the devices have moved into a configuration with minimal potential U, they stop moving and stay stationary until a disruption is introduced, which necessitates their relocation. For a suitable choice of $\gamma(k)$, convergence will follow from the convergence of the steepest descent method in optimization. In the following sections, where we discuss distributed implementations of network dynamics, we shall use our steepest descent formulation of motion as the starting point and will discuss the selection of $\gamma(k)$.

3 Distributed Implementations of Network Dynamics

In this section, we discuss how we can implement network dynamics in an MSN in a completely distributed way with realistic constraints.

3.1 Overview of Distributed Network Dynamics

In Section 2, we discussed the discretization of network dynamics. A straightforward way to implement network dynamics is to have a central controller that has the knowledge of each device's position and the ability to send its directives to each device. In practice, however, such an approach is unrealistic. An MSN does not have a centralized infrastructure by its ad hoc nature. Consequently, we must devise a set of distributed algorithms so that each node makes decisions based on its local environment to achieve the global objective.

System model. In the effort of formulating and implementing the distributed network dynamics algorithm, we have made the following assumptions regarding the underlying MSN:

- *Two-dimensional deployment*. The network is deployed on a 2D plane, and as a result, the movement of the nodes are also constrained on the 2D plane.
- *Limited mobility*. Nodes can move, but the mobility is limited both by the maximum speed at which a node can move and by the total distance that a node can move because movement, in general, consumes a large amount of energy.
- *Location aware*. Every node knows its own location. This can be achieved by devices such as GPS [17] or through various wireless localization algorithms [18].

Performance metrics. We propose the following metrics to evaluate the proposed distributed algorithms:

- *Movement efficiency*. Sensor nodes will experience different movement trajectories following the movement algorithm. To measure the efficiency of these trajectories, we add up the total distance traveled by all the sensors: a shorter travel distance indicates a lower energy consumption and, hence, a better movement efficiency.
- *Convergence time*. When the network dynamics algorithm converges, every node within the network will have a force, which is below the specified threshold. At the same time, the system potential energy U will reach its minimum. In general, a shorter convergence time is preferred.

3.2 Distributed Network Dynamics Algorithm

The easiest way of implementing the distributed network dynamics algorithm is the sequential approach. While a node is moving, the nodes in its radio range must remain stationary. Though it is easy to implement it and prove its convergence, this approach suffers inefficiencies due to its sequential nature. More specifically, it limits the number of nodes that may move at any time, which, in turn, prolongs the convergence time and the overall moving distance. In order to address these inefficiencies, we propose a parallel and distributed algorithm named the PDND algorithm.

In PDND, any node that intends to move can start movement immediately and, therefore, all nodes may move simultaneously. Each node advertises its location information every Δt time and maintains a table (neighbor table) that records its neighbors' most recent location information. Every Δt time, based on the information saved in its neighbor table, a node i calculates the total force $\mathbf{f_i}$ applied to it by all its neighbors. If the magnitude of $\mathbf{f_i}$ is greater than a preset threshold δ, that is, $\|\mathbf{f_i}\| > \delta$, node i will move along the direction of $\mathbf{f_i}$ for a specific distance that is a function of $\|\mathbf{f_i}\|$ and Δt (details are provided in Section 4). Node i stops moving when $\mathbf{f_i}$ is below the threshold δ. All nodes repeat this process iteratively, and finally, they will all stop moving.

The PDND algorithm is summarized in Algorithm 1. For PDND, Δt is an important parameter, for a course Δt may make sensors oscillate. However, we argue that since the communication among sensors happens in the scale of microseconds, Δt can be easily set to a small value such that its effect on the performance of PDND is negligible.

Algorithm: *Parallel and Distributed Network Dynamics Algorithm*

```
while (1) do
    f = calculateForce (my_location, neighbor_location);
    if (‖f‖ > δ) then
        calculateNewPos (my_location, f);
        moveToNewPos;
        send(my_location);
    else
        wait(Δt);
    end
end
```

4 The Convergence of the PDND Algorithm

In this section, we formally prove the convergence of PDND. We first prove its convergence on a convex sensing field and then extend the proof to cases with nonconvex sensing fields. We assume that there is a total of N sensors, indexed by $1, \ldots, N$, and that their effective sensing ranges, as well as their communication ranges, are identical discs. In what follows, we use r_s to represent the common sensing radius and r_c to represent the common communication radius. We use $\mathbf{p_i} = [x_i \; y_i]^T$ to represent the coordinates of sensor i and use $\mathbf{p}$ to denote the coordinates of all the sensor nodes, with $\mathbf{p} = [\mathbf{p_1}^T \; \cdots \; \mathbf{p_N}^T]^T$. Further, $\mathbf{P}$ denotes all the feasible choices of $\mathbf{p}$. $\mathbf{f_{ij}} = [f_{ij,x} \; f_{ij,y}]^T$ denotes the virtual force placed on sensor i by sensor j $(j \neq i)$ and, therefore, the total force on sensor i can be formulated as $\mathbf{f_i} = \sum_{j=1, j\neq i}^{N} \mathbf{f_{ij}}$. Finally, we use r_f to denote the maximum distance at which two sensors have a force between them.

As nearby sensors have virtual forces with each other, the whole network possesses a virtual potential energy $U(\mathbf{p})$, and $-\nabla U(\mathbf{p}) = \mathbf{f} = [\mathbf{f_i}^T \; \cdots \; \mathbf{f_N}^T]^T$ according to physical laws. The problem of repositioning sensors with the help of $U(\mathbf{p})$ can be transformed into the following optimization problem:

$$\begin{aligned} \min \quad & U(\mathbf{p}) \\ \text{subject to} \quad & \mathbf{p} \in \mathbf{P}. \end{aligned}$$

It should be noticed that this optimization problem is not convex. Given any optimal solution $\mathbf{p}^*$, one can obtain another optimal solution $\mathbf{p}'$ by exchanging the positions of any two sensors in $\mathbf{p}^*$.

The PDND algorithm proposed in Section 3 is similar to the standard gradient projection algorithm, which is formulated as

$$\mathbf{p}(k+1) = \Big[\mathbf{p}(k) - \gamma(k)\nabla U\big(\mathbf{p}(k)\big)\Big]^+, \tag{7}$$

where $\gamma(k)$ is a positive step size at the kth iteration of the algorithm, and $[\mathbf{p}]^+$ is the orthogonal projection (with respect to the euclidean norm) defined by

$$[\mathbf{p}]^+ = \arg\min_{\mathbf{p}' \in \mathbf{P}} \|\mathbf{p}' - \mathbf{p}\|_2. \tag{8}$$

Although the standard gradient projection algorithm is a suitable numerical method that can be used to find solutions for optimization problems, it is inappropriate for our problem due to two reasons. First, it needs global information to find out the appropriate step size by carrying out a line search algorithm in each iteration. Second, by adopting a single $\gamma(k)$

for all sensors, it does not take into account the speed limit of the sensors. As a result, during the time interval of the kth iteration, a sensor i may be asked to travel a distance far beyond its capability. To address the second shortcoming, one can either increase the locomotion capability of the sensors or extend the duration of each iteration. Both options, however, have drawbacks: the former is not always realistic, whereas the latter may significantly slow down the convergence of the algorithm. *In this paper, we propose a better choice, the PDND algorithm, which modifies the standard gradient projection algorithm by letting each sensor independently choose its own step size based on the local information*. The PDND algorithm is described by the following:

$$\mathbf{p}(t+\Delta t) = \Big[\mathbf{p}(t) - \mathbf{diag}(\gamma(t))\nabla U\big(\mathbf{p}(t)\big)\Big]^{+}, \tag{9}$$

where $\gamma(t) = [\gamma_1(t)\ \gamma_1(t)\ \ldots\ \gamma_i(t)\ \gamma_i(t)\ \ldots\ \gamma_N(t)\ \gamma_N(t)]^T \succeq 0$.

Before discussing the convergence of the PDND algorithm, we first present the assumptions that we have made for the considered systems. Note that these assumptions will not make the considered system less realistic.

Assumption 1. $r_f < r_c$.

This assumption eases the implementation of the proposed algorithm because sensor i only needs the position information of its communication neighbors to calculate $\mathbf{f_i}$.

Assumption 2 (Lipschitz continuity of $\mathbf{f_{ij}}$). *There exists a constant C such that*

$$\|\mathbf{f'_{ij}} - \mathbf{f_{ij}}\|_2 \le C\Big\|[\mathbf{p_i'}^T\ \mathbf{p_j'}^T]^T - [\mathbf{p}_i^T\ \mathbf{p_j}^T]^T\Big\|_2. \tag{10}$$

Under this assumption, the force between two sensors is a bounded continuous function of the distance between them. We now present some theoretical results describing the convergence of PDND.

Proposition 1. *$U(\mathbf{p})$ is bounded below for every feasible $\mathbf{p}$.*

Proof. Given that $-\nabla U(\mathbf{p}) = \mathbf{f}$ and according to Assumption 2, this is obvious. □

Proposition 2 (Lipschitz continuity of $\nabla U(\mathbf{p})$). *$U(\mathbf{p})$ is continuously differentiable, and there exists a constant K such that*

$$\|\nabla U(\mathbf{p}') - \nabla U(\mathbf{p})\|_2 \le K\|\mathbf{p}' - \mathbf{p}\|_2, \tag{11}$$

where $\mathbf{p}'$, $\mathbf{p} \in \mathbf{P}$.

Proof. Let $N(i)$ be the set consisting of every sensor j ($j \neq i$) that satisfies either $\|\mathbf{p_i'} - \mathbf{p_j'}\|_2 \le r_f$ or $\|\mathbf{p_i} - \mathbf{p_j}\|_2 \le r_f$. Let L be the maximum set size among all the sensors, that is, $L = \mathbf{max}_i\ |N(i)|$. The fact that $\mathbf{f_{ij}}$ is Lipschitz continuous indicates that there exists a constant C such that

$$\|\mathbf{f'_{ij}} - \mathbf{f_{ij}}\|_2 \le C\big\|[\mathbf{p_i'}^T\ \mathbf{p_j'}^T]^T - [\mathbf{p_i}^T\ \mathbf{p_j}^T]^T\big\|_2 \le C(\triangle d_i + \triangle d_j),$$

where $\triangle d_i = \|\mathbf{p_i'} - \mathbf{p_i}\|_2$. Therefore, the constant K can be found in the following way:

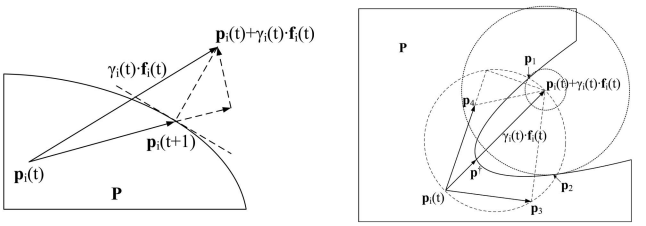

Fig. 2. (a) The orthogonal projection on a convex sensing field. (b) The sensor moving strategy on a nonconvex sensing field.

$$\begin{aligned}
&\|\nabla U(\mathbf{p}') - \nabla U(\mathbf{p})\|_2^2 \\
&= \sum_{i=1}^{N}\left[\Big(f'_{i,x} - f_{i,x}\Big)^2 + \Big(f'_{i,y} - f_{i,y}\Big)^2\right] \\
&= \sum_{i=1}^{N}\left[\left(\sum_{j\in N(i)}\Big(f'_{ij,x} - f_{ij,x}\Big)\right)^2 + \left(\sum_{j\in N(i)}\Big(f'_{ij,y} - f_{ij,y}\Big)\right)^2\right] \\
&\le L\sum_{i=1}^{N}\left[\sum_{j\in N(i)}\Big(f'_{ij,x} - f_{ij,x}\Big)^2 + \sum_{j\in N(i)}\Big(f'_{ij,y} - f_{ij,y}\Big)^2\right] \\
&\le LC^2\sum_{i=1}^{N}\sum_{j\in N(i)}(\triangle d_i + \triangle d_j)^2 \\
&\le 2LC^2\sum_{i=1}^{N}\sum_{j\in N(i)}(\triangle d_i^2 + \triangle d_j^2) \\
&\le 4L^2C^2\sum_{i=1}^{N}\triangle d_i^2 \\
&= 4L^2C^2\|\mathbf{p}' - \mathbf{p}\|_2^2.
\end{aligned}$$

As a result, $K = 2LC$. □

Lemma 1. $\forall \mathbf{p}'$, $\mathbf{p} \in \mathbf{P}$

$$U(\mathbf{p}') \le U(\mathbf{p}) + (\mathbf{p}' - \mathbf{p})^T\nabla U(\mathbf{p}) + \frac{K}{2}\|\mathbf{p}' - \mathbf{p}\|_2^2. \tag{12}$$

Proof. The proof is given in [19]. □

Lemma 2 (Properties of PDND on a convex set).

1. $$U\big(\mathbf{p}(t+\Delta t)\big) \le U\big(\mathbf{p}(t)\big) - \big(\mathbf{p}(t+\Delta t) - \mathbf{p}(t)\big)^T \mathbf{diag}\big(1/\gamma(t) - K/2\big)\big(\mathbf{p}(t+\Delta t) - \mathbf{p}(t)\big).$$
2. $\mathbf{p}(t+\Delta t) = \mathbf{p}(t)$ *if and only if* $\big(\mathbf{p}' - \mathbf{p}(t)\big)^T\nabla U\big(\mathbf{p}(t)\big) \ge 0$ *for all feasible* $\mathbf{p}'$.
3. *The mapping* $\big[\mathbf{p}(t) - \mathbf{diag}\big(\gamma(t)\big)\nabla U\big(\mathbf{p}(t)\big)\big]^+$ *is continuous.*

Proof. Only the proof of the first property is given here, and the rest is almost identical to that of the standard gradient project algorithm provided in [19].

From the definition of the projection method, we know that for each i (see Fig. 2a),

$$\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\gamma_i(t)\mathbf{f_i}(t) \geq \big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)$$

$$\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\mathbf{f_i}(t) \geq \frac{1}{\gamma_i(t)}\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big).$$

Therefore,

$$-\sum\nolimits_i\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\mathbf{f_i}(t) \leq -\sum\nolimits_i\frac{1}{\gamma_i(t)}\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)$$

$$\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)^T\nabla U\big(\mathbf{p}(t)\big) \leq -\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)^T\mathbf{diag}\big(1/\boldsymbol{\gamma}(t)\big)\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big).$$

Applying Lemma 1, we get

$$\begin{aligned}U\big(\mathbf{p}(t+\Delta t)\big) &\leq U\big(\mathbf{p}(t)\big)+\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)^T\nabla U\big(\mathbf{p}(t)\big)+\frac{K}{2}\|\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\|_2^2\\ &\leq U\big(\mathbf{p}(t)\big)-\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)^T\mathbf{diag}(1/\boldsymbol{\gamma}(t)-K/2)\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big).\end{aligned}$$

□

Theorem 3 (Convergence of PDND on a convex set). *If* $0<\max_i\gamma_i(t)<2/K$, *and* $\mathbf{p}^*$ *is a limit point of the sequence* $\{\mathbf{p}(t)\}$ *generated by the PDND algorithm,* $(\mathbf{p}-\mathbf{p}^*)^T\nabla U(\mathbf{p}^*)\geq 0$ *for all feasible* $\mathbf{p}$.

Proof. Let $\{\mathbf{p}(t)\}$ be the sequence generated by the PDND algorithm, where t can be $0,\Delta t,2\Delta t,\cdots$. We first consider the situation where $\gamma_i(t)>0$ for all i. The condition $0<\max_i\gamma_i(t)<2/K$ guarantees that the matrix $\mathbf{diag}\big(1/\gamma(t)-K/2\big)$ is positive definite. Applying this observation to Lemma 2.1, we get the conclusion that the sequence $\{U(\mathbf{p}(t))\}$ is strictly decreasing, unless $\mathbf{p}(t+\Delta t)=\mathbf{p}(t)$. Further, as $U(\mathbf{p})$ is bounded below, this sequence converges.

When at least one $\gamma_i(t)=0$, we may puncture both $\mathbf{diag}\big(1/\gamma(t)-K/2\big)$ by removing all zeros on the diagonal and $\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)$ by removing all corresponding zeros. Applying the same reasoning as before completes the proof. □

We have proven that under minor assumptions, the PDND algorithm converges on a convex sensing field. Next, we study its convergence in a nonconvex area. In such scenarios, the orthogonal projection method can no longer be used to calculate sensor locations. In order to address this difficulty, we propose a simple but efficient movement regulation strategy. When a sensor i approaches the boundary of the sensing field, it regulates its movement in the following way: if the target position $\big(\mathbf{p_i}(t)+\gamma_i(t)\mathbf{f_i}(t)\big)$ is outside the sensing field, sensor i moves within the sensing field and to the location that is the closest one in its vicinity to the target location. Specifically, sensor i's movement must be inside or on the circle centered at $\big(\mathbf{p_i}(t)+\gamma_i(t)\mathbf{f_i}(t)/2\big)$, with the radius $\gamma_i(t)\mathbf{f_i}(t)/2$. Fig. 2b illustrates this strategy. Sensor i first moves toward its target location $\big(\mathbf{p_i}(t)+\gamma_i(t)\mathbf{f_i}(t)\big)$ until it hits the boundary of the sensing field at $p^\dagger$, and then, it randomly picks a direction and moves along the boundary until it reaches a point that is closest to the target location in that particular direction, for example, $\mathbf{p_1}$ or $\mathbf{p_2}$ in Fig. 2b. Additionally, if it has a priori knowledge of the shape of the boundary, it can calculate the point that is closest to the target position, for example, $\mathbf{p_1}$ in Fig. 2b, and move to that point following the optimal path.

Lemma 3 (Properties of PDND on a nonconvex set).

$$U\big(\mathbf{p}(t+\Delta t)\big)\leq U\big(\mathbf{p}(t)\big)-\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big)^T\mathbf{diag}\big(1/\boldsymbol{\gamma}(t)-K/2\big)\big(\mathbf{p}(t+\Delta t)-\mathbf{p}(t)\big). \quad (13)$$

Proof. Any position that is inside or on the circle centered at $\big(\mathbf{p_i}(t)+\gamma_i(t)\mathbf{f_i}(t)/2\big)$, with the radius $\gamma_i(t)\mathbf{f_i}(t)/2$, for example, $\mathbf{p_3}$ or $\mathbf{p_4}$ in Fig. 2b, satisfies the following inequality:

$$\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\gamma_i(t)\mathbf{f_i}(t)\geq\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big)^T\big(\mathbf{p}_i(t+\Delta t)-\mathbf{p}_i(t)\big).$$

According to the aforementioned movement regulation strategy, $\mathbf{p_i}(t+\Delta t)$ is such a position. The rest of the proof is the same as that of Lemma 2.1 and is hence omitted. □

Theorem 4 (Convergence of PDND on a nonconvex set). *If* $0<\max_i\gamma_i(t)<2/K$, *the PDND algorithm converges.*

Proof. Let $\{\mathbf{p}(t)\}$ be the sequence generated by the PDND algorithm. When all $\gamma_i(t)$'s are greater than zero, from the condition $0<\max_i\gamma_i(t)<2/K$, we find that $\mathbf{diag}\big(1/\gamma(t)-K/2\big)$ is positive definite. Applying this observation to Lemma 3, we can conclude that the sequence $\{U(\mathbf{p}(t))\}$ is strictly decreasing, and since $U(\mathbf{p})$ is bounded below, this sequence converges. When there exists at least one $\gamma_i(t)$ that is equal to zero, the convergence can be proven using the same strategy in the proof of Theorem 3. □

5 Case Study 1: Spatial Coverage

In the first case study, we explore the applicability of network dynamics to the problem of maximizing the spatial coverage of an MSN.

5.1 Problem Statement

To set up the problem, let us consider the scenario in which an MSN is deployed to monitor a particular region (referred to as sensing field). It is sometimes difficult to initially deploy the sensors in such a way that the maximum coverage is achieved. On the other hand, the ease of dropping sensors over a smaller region or randomly over the whole sensing field suggests that we adopt a two-phase strategy that involves first randomly dumping the sensor nodes and then letting the sensors adjust their positions to better cover the area. Furthermore, even if it is possible to

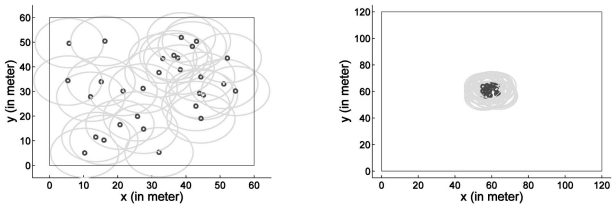

Fig. 3. Initial deployments. (a) Case 1. (b) Case 2.

initially place sensors to achieve the maximum coverage, it is still desirable to reconfigure their positions on the fly because sensors may fail during the operation.

The proposed PDND algorithm aims to make an MSN an autonomous entity that always tries to maximize its sensing coverage.

Coverage degree. The most important metric for this problem domain is the coverage degree, which measures the ratio of the entire sensing field that is covered. PDND intends to maximize the coverage degree of a given network by coordinating the movement of the sensors, and as a final result, sensors should be sufficiently separated from each other to maximize coverage but, at the same time, sufficiently close to each other to stay connected. Before presenting how we can measure the coverage degree, we first introduce the two terms that are associated with each sensor: coverage range and Voronoi cell. In this study, we assume a simple disc coverage model, in which a sensor can cover a circular area (referred to as coverage range), with the radius r_s centered at the sensor itself. After all sensors are deployed on the field, each sensor has a Voronoi cell, a generalized polygon whose interior consists of all points in the plane, which are closer to that sensor than to any other. In order to calculate the overall coverage degree, we adopt a divide-and-conquer strategy. We first divide the whole sensing field into Voronoi cells based on the positions of the sensors, and then, we let each sensor calculate what fraction of its own Voronoi cell is covered (by comparing the area of the Voronoi cell and the coverage range), which is a simple geometric problem, and the details can be found in [20].

Force model. Although any force model that satisfies Assumption 2 can be used, we choose the following one because of its simplicity:

$$\mathbf{f}_{ij} = \left[(r^* - d_{ij}) I_t(d_{ij}) + \frac{(r^* - r_t)(r_f - d_{ij})}{r_f - r_t} \left(I_f(d_{ij}) - I_t(d_{ij}) \right) \right] \mathbf{u}_{ij}. \tag{14}$$

The notations in the foregoing equation are explained as follows:

- r^* is the distance between two sensors when the force between them is zero. As the distance becomes shorter (or longer), they start to repel (or attract) each other.
- r_f ($> r^*$) is the distance beyond which the attractive force between two sensors vanishes.

TABLE 1
Default PDND Parameter Values

parameters	default values
sensing radius r_s	$10m$
communication radius r_c	$30m$
sensor speed	$1m/s$
force model parameter r^*	$22m$
force model parameter r_t	$24m$
force model parameter r_f	$26m$
time interval	$1s$
stopping criterion	$0.1m$

- r_t ($r^* < r_t < r_f$) is the distance at which two sensors attract each other most. The attractive force drops off as the distance decreases or increases.
- $d_{ij} = \|\mathbf{p_i} - \mathbf{p_j}\|_2$.
- $\mathbf{u_{ij}} = (\mathbf{p_i} - \mathbf{p_j})/d_{ij}$.
- $I_t(d) = \begin{cases} 1 & \text{if } d \le r_t \\ 0 & \text{otherwise} \end{cases}$, $I_f(d) = \begin{cases} 1 & \text{if } d \le r_f \\ 0 & \text{otherwise.} \end{cases}$

The constant C in Assumption 2 of this force model can be easily verified to be $\sqrt{2}$.

5.2 Simulation Results

In this exercise, we conduct detailed simulation studies to examine the effectiveness and efficiency of PDND in improving the sensor network's spatial coverage. In addition, we develop a variation of PDND that utilizes a more relaxed convergence criterion. Finally, we compare the performance of the two PDND algorithms with an Voronoi-diagram-based movement algorithm.

In our simulation studies, we consider two random initial deployments involving 30 sensors, one over a $60 \times 60\ \mathrm{m}^2$ area and the other over a $120 \times 120\ \mathrm{m}^2$ area, which are illustrated in Figs. 3a and 3b. We use these two cases to represent two typical random deployment strategies: case 1 represents the deployments where the sensors are randomly thrown over the whole area, and case 2 represents the deployments where the sensors are randomly dumped within a very small area (in this case, the sensors are placed within a $10 \times 10\ \mathrm{m}^2$ area, whereas the intended deployment area is $120 \times 120\ \mathrm{m}^2$). Additionally, case 1 represents a dense network, whereas case 2 represents a sparse one. The initial topologies shown in Fig. 3 have coverage degrees of 0.8709 and 0.0306, respectively.

PDND has several important parameters, and their default values are summarized in Table 1. Most of the parameters are self explainable, except for the stopping criterion. The stopping criterion works in the following way. At the beginning of each time interval, every sensor calculates the position at which its force will be zero based on the local information. However, it only starts moving if the distance between the target position and its current position is larger than the stopping criterion. Setting a threshold on the force magnitude, as discussed in Section 3, would be a more natural choice for PDND, but we chose to use a distance-based threshold because the Voronoi-diagram-based algorithm that we are going to compare with uses such a threshold. In some experiments, we adopt different values for these parameters. We will explicitly specify them when presenting those results.

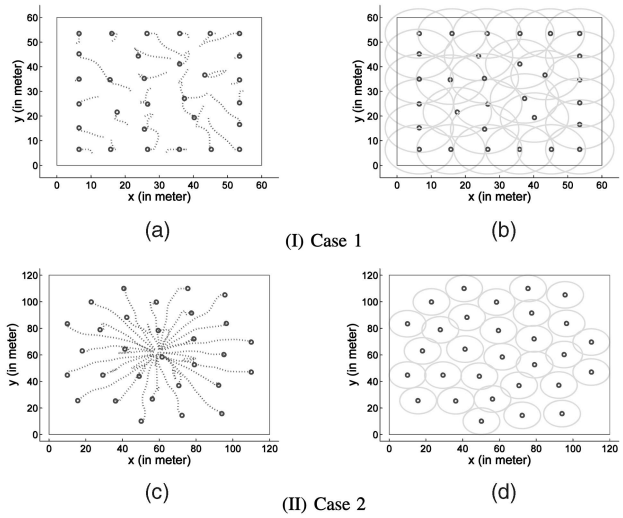

Fig. 4. Sensor trajectories and final deployments under PDND. (a) and (c) Trajectories. (b) and (d) Final deployment.

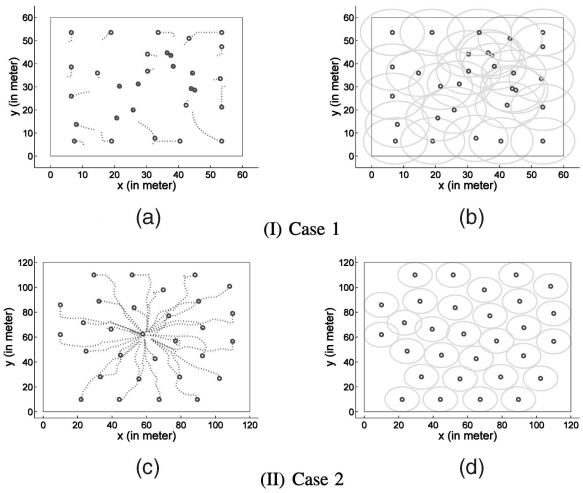

Fig. 5. Sensor trajectories and final deployments under PDND2. (a) and (c) Trajectories. (b) and (d) Final deployment.

5.2.1 Effectiveness of PDND

After applying PDND to the two deployments, we present the resulting sensor trajectories and final topologies in Fig. 4. For both deployments, PDND can significantly increase the coverage degree by guiding the movement of the sensors: the coverage degree in case 1 is boosted from 0.8709 to 1, and in case 2, from 0.0306 to 0.6449. We would like to point out that in case 2, the coverage degree can be at most 0.6545 because 30 nodes are not enough to fully cover the sensing field. The final topology for case 2 shown in Fig. 4d still includes a small overlap among sensing areas, and this is because the stopping criterion is not strict enough. The relatively straight trajectories in Figs. 4a and 4c reveal that PDND can not only achieve better coverage degree, but also lead to efficient sensor movements.

5.2.2 Approximate PDND

The rationale behind PDND is that sensors should keep moving until the potential energy of the network is minimized, and when the algorithm converges, the sensors are usually well separated from each other, and the coverage degree is maximized. However, if the network has an enough number of sensors, it is often unnecessary to evenly distribute them to fully cover the sensing field. As a result, we propose the *approximate PDND* algorithm (PDND2) that employs a more relaxed convergence criterion. During a time interval, a sensor moves if and only if the following conditions are true: 1) its Voronoi cell is not fully covered and 2) its intended movement distance is longer than the stopping criterion. This way, sensor nodes will move much less compared with the original PDND algorithm.

After applying PDND2 to the two initial deployments shown in Fig. 3, we present the resulting sensor trajectories and final topologies in Fig. 5. The results confirm that PDND2 can significantly reduce the overhead for dense deployments. For example, in case 1, the approximate PDND reduces the total travel distance by 59.65 percent while still maintaining a coverage degree of 1. Furthermore, it reduces the convergence time by 50 percent. On the other hand, its performance is comparable to that of the original PDND algorithm in case 2, where the sensor density is low.

5.2.3 Voronoi-Diagram-Based Mobility Management

Although PDNDs use the concept of potential energy and virtual forces to govern the movement of the sensors, another set of popular mobility control algorithms are built upon the concept of Voronoi tessellation, such as the technique discussed in [12], [13]. In order to study the difference between these two sets of algorithms, in this exercise, we implement a Voronoi-diagram-based algorithm (referred to as the Lloyd algorithm) and compare its performance with that of PDNDs'. Like PDNDs, Lloyd also partitions the time axis into discrete intervals. During an interval, each sensor moves toward the centroid of its current Voronoi cell. Details of the Lloyd algorithm can be found in [13].

When we apply Lloyd to both initial deployments in Fig. 3, we notice that when the network does not have enough sensors to fully coverage its sensing field as in case 2, sensors at the edge of the network may keep oscillating. In such a sparse network, a sensor at the edge may have a large Voronoi cell and, thus, the centroid of its Voronoi cell is very likely out of the communication ranges of all other sensors. This loss of connection with the rest of the network will, in turn, lead to errors in calculating Voronoi cells. In order to cope with such oscillations, we manually stop the Lloyd algorithm when it hits the preset upper bound of the convergence time. As an example, in case 2, such oscillations occur, and we terminate Lloyd manually after 500 seconds. We never observe this phenomenon in dense networks, as in case 1. Finally, the final topologies and sensor trajectories are shown in Fig. 6.

5.2.4 Performance Comparison

In order to take a closer look at the execution of these three algorithms, we present the time series for the coverage degree and potential energy of the entire network in Figs. 7 and 8. After carefully examining the spatial coverage time series, we have the following observations. First, for dense

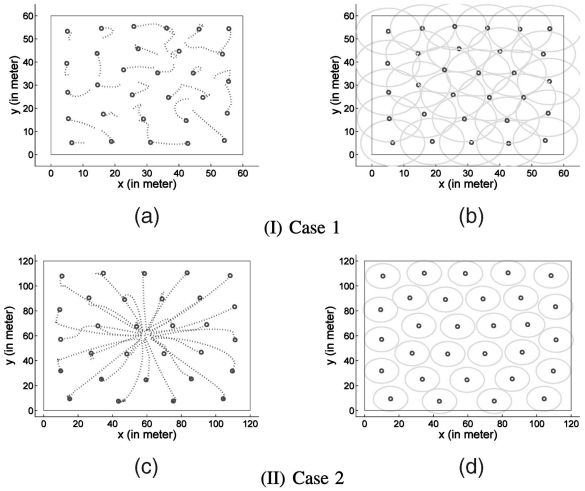

Fig. 6. Sensor trajectories and final deployments under the Lloyd algorithm. (a) and (c) Trajectories. (b) and (d) Final deployment.

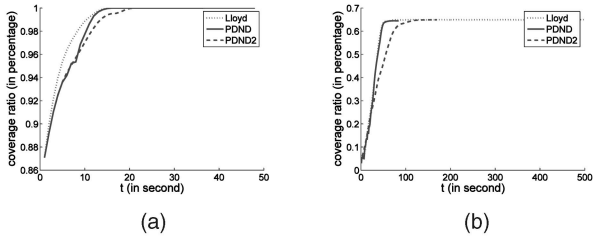

Fig. 7. Coverage degree time series. (a) Case 1. (b) Case 2.

networks (refer to Fig. 7a), the convergence time of the Lloyd algorithm is comparable with PDND's. For sparse networks, on the other hand, the Lloyd algorithm converges very slowly (or it has to be forced to stop), as discussed above. Second, both PDND and Lloyd can maximize the coverage degree quickly and then spend a long time fine tuning the position of each sensor. On the contrary, the PDND2 algorithm can efficiently reduce the fine-tuning overhead, without sacrificing the overall coverage degree. Third, PDND2 takes a longer time to reach the maximum coverage degree than the other two, as the sensors move at

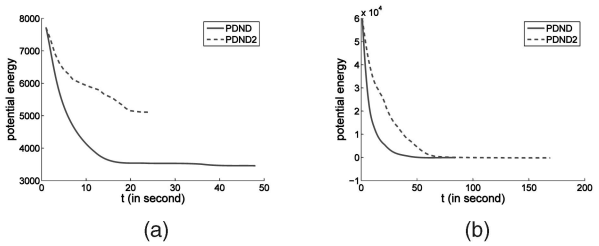

Fig. 8. Potential energy time series. (a) Case 1. (b) Case 2.

a slower pace due to the adopted movement criterion. Fourth, moving toward centroids or moving along the directions of virtual forces does not necessarily lead to a strict coverage degree increasing in the middle of algorithms' running and, therefore, the time series may exhibit zigzag-like behaviors.

The system potential energy time series (refer to Fig. 8) shows similar trends. However, we would like to emphasize two issues. First, we observe that the potential energy of the network strictly decreases, as we proved before. Second, in case 1, PDND2 stops when the potential energy is still high because each sensor has already fully covered its Voronoi cell.

In the next set of experiments, we study how these three algorithms perform when we vary some of the parameters, that is, the time interval duration and the stop criterion value. The detailed results are presented in Tables 2 and 3. In general, a smaller time interval may speed up the convergence, as each node will have more up-to-date knowledge about other nodes, whereas a smaller stop criterion may lead to a slower convergence time and longer travel distance. In order to compensate for the randomness of the results, each entry in the table is the average of 100 different initial deployments. Further, all the 100 initial deployments that belong to case 1 are generated by uniformly randomly throwing sensors over the entire $60 \times 60\ \mathrm{m}^2$ field, whereas all the 100 initial deployments that belong to case 2 are generated by uniformly randomly throwing sensors over the central $10 \times 10\ \mathrm{m}^2$ area of the $120 \times 120\ \mathrm{m}^2$ sensing field.

TABLE 2
Numerical Results for Case 1

Time Interval (in s)		1						0.1					
Stop Criteria (in m)		0.2			0.05			0.02			0.005		
Algorithm		Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2
Final Coverage Degree	Mean	1	1	1	1	1	1	1	1	1	1	1	1
	Var (1e-9)	2.2	6.8	1097	0	0.2	822	0	42	1801	0	3.3	391
Convergence Time (in s)	Mean	33.04	37	23.42	67.57	67.11	34.18	20.52	27.02	21.05	27.12	50	30.24
	Var	50.79	83.27	51.16	365.34	402.1	255.83	14.59	41.69	47.69	43.12	207.65	310.99
Total Travel Distance (in m)	Mean	247.65	254.47	109.37	288.87	286.96	105.31	335.22	275.54	96.25	348.16	306.56	99.01
	Var	1233.8	937.7	551	1988.9	1618.4	672.81	2174.5	1404.6	548.46	2179.4	1643.6	593.41

TABLE 3
Numerical Results for Case 2

Time Interval (in s)		1						0.1					
Stop Criteria (in m)		0.2			0.05			0.02			0.005		
Algorithm		Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2	Lloyd	PDND	PDND2
Final Coverage Degree	Mean	0.6498	0.6433	0.6412	0.6516	0.6455	0.6452	0.652	0.6462	0.6452	0.652	0.6461	0.6451
	Var (1e-6)	5.3	9.6	15	3.2	4	4.4	3.8	3.2	4.2	1.5	3.4	3.7
Convergence Time (in s)	Mean	475.87	82.66	119.4	519.86	141.45	186.08	367.88	73.51	97.43	431.78	159.41	156.17
	Var	193170	138.13	237.39	170120	951.08	1105.3	39256	102.87	276.04	24936	7734.5	5646.4
Total Travel Distance (in m)	Mean	2805.4	1296.9	1192.6	3006.3	1397.8	1311.3	2201.8	1584.2	1418.7	2658.8	1713.6	1511.2
	Var	2754000	454.39	878.32	3603800	1063.2	1529	391800	3927.5	2912.6	562770	16225	8464

For network topologies that belong to case 1, with the time interval of 1 sec, PDND and Lloyd deliver very similar performances, regardless of the stop criteria. With the same configuration, PDND2 incurs much lower overheads while achieving the same coverage degree: compared to the other two algorithms, it can reduce the convergence time by 40 percent and the total travel distance by 60 percent. As the time interval becomes smaller, the performances of PDND and Lloyd start to differ: PDND results in a shorter travel distance, whereas Lloyd can converge faster. More specifically, the performance gain of PDND in terms of the total travel distance is more significant with a slightly larger stop criteria, whereas the performance gain of Lloyd in terms of convergence time is more significant with a slightly smaller stop criteria. On the other hand, under a smaller time interval, PDND2 can reduce the total travel distance even further, but its convergence time becomes longer relative to that of Lloyd.

We observe very different trends for sparse network topologies that belong to case 2. As discussed earlier, Lloyd may cause oscillations in such cases and, thus, either converge very slowly or never converge. As a result, both PDNDs significantly outperform Lloyd. In the simulations, we stop Lloyd at 1,000 sec when the time interval is 1 or 0.5 seconds and at 500 seconds when the time interval is 0.1 seconds. As a result, the true performance of the Lloyd algorithm can be much worse than what is shown in Table 3. Also, as we discussed before, the advantage of PDND2 is less pronounced for sparse networks.

6 Case Study 2: Spatial Migration

In our second case study, we examine the possibility of using network dynamics to realize MSNs that can migrate themselves to track moving objects.

6.1 Problem Setup

Many sensor applications aim at monitoring mobile objects, such as the one presented in [21]. There are three general ways of implementing such applications. First, we can attach sensor nodes to the mobile targets and then let the sensors ship the data back to the base station in an opportunistic manner. Second, we can place a sufficiently large number of sensors to cover all the possible (or important) areas that the target may visit frequently. Third, we can deploy mobile sensor nodes that can autonomously follow the moving objects. There are pros and cons associated with each of these approaches, and no single approach can suit all different application scenarios. For instance, though the first approach is cost effective, attaching sensor nodes to a mobile target might not always be possible. As another example, if the target's movement is predictable and is within a limited range, the second approach is viable. However, it will be too costly an option if the target covers a large area. On the other hand, as more sensor nodes are equipped with mobility and as the mobility-management techniques advance, the third approach will become more prevalent.

Migration metric. In this case study, we adopt the third approach, and the most important metric is the sensor network's capability of chasing the target. If the network of sensor nodes can follow the target at its movement, the mobility management algorithms are considered successful.

Force model and application model. Since the focus of this study is on governing sensor mobility to chase mobile targets, we do not intend to elaborate on how sensor nodes can detect the movement or location of the target. Instead, we assume that sensors whose sensing ranges cover the target can obtain the target's location information. Further, we assume that each sensor can operate in two modes: normal mode and chasing model. A sensor node switches to the chasing mode when it detects that the target is moving. In some cases, it may be more desirable to prevent sensor nodes from chasing the target when the target moves about within the network. To address this concern, we can let boundary nodes switch to the chasing mode first because they can detect whether the target is leaving the network.

Sensor nodes in the normal mode employ the same force model, as discussed in the previous case study. In addition to the forces that exist in the normal mode, a sensor in the chasing mode is also influenced by an attractive force $\mathbf{f}'$ pointing to the position of the target. In this study, we choose to set $\mathbf{f}'$ to a constant value for all sensors in the chasing mode, regardless of their distances to the target. The larger this constant value, the tighter the sensors follow the target.

Now, let us look at how the entire network migrates following the moving target. As soon as the target moves, and the nodes that can detect its movement decide to chase, these sensor nodes switch to the chasing mode. As for the rest of the sensor nodes, we can adopt two strategies:

- *Rapid chase*. In this case, nodes in the chasing mode immediately flood the entire network with a *Switch2-Chase* REQ message, and a sensor node in the normal mode reacts to the REQ message by switching to the chasing mode.
- *Slow chase*. In this case, only the nodes that can detect the movement of the target will stay in the chasing mode. Other nodes, though in the normal mode, will also move due to the movement of their neighbors.

As mentioned before, nodes in the chasing mode are affected by an attractive force pointing to the target, whose magnitude is assigned a constant value. The calculation of the force requires each sensor node to have the knowledge of the target's location. For this purpose, we assume that the target's position can only be sensed within a certain range (referred to as *detection range*). Sensor nodes within the detection range of the target can obtain the target's position, and if under the rapid chase strategy, they will broadcast the information to the rest of the network periodically. Sensors may lose the target if all of them are outside the detection range. This is the scenario that we try to avoid.

6.2 Simulation Results

6.2.1 Rapid Chase Strategy

We set up simulation experiments to demonstrate the effectiveness of PDND in enabling MSNs to track mobile targets. Meanwhile, to examine the scalability of the PDND algorithm, we test two networks: a smaller one with 30 sensors and a larger one with 60 sensors. At the

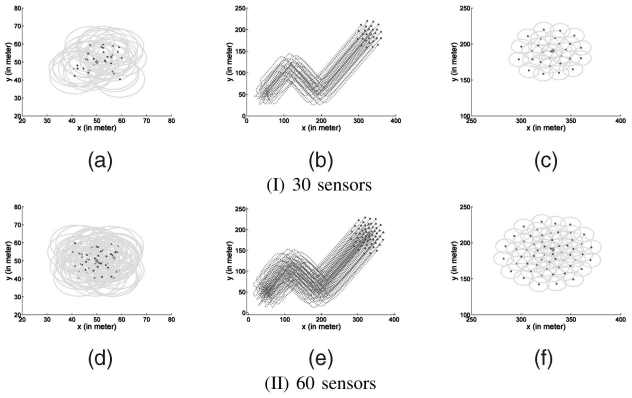

Fig. 9. Rapid chase results. (a) and (d) Initial deployment. (b) and (e) Trajectories. (c) and (g) Final deployment.

beginning of the experiments, the target is located at (50, 50), and sensors are randomly deployed around it, as shown in Fig. 9a for the 30-node network and in Fig. 9d for the 60-node network.

Once the experiment starts, the target moves following the trajectory, as shown in Figs. 9b and 9e, respectively (the red lines) for 400 seconds. To make the chasing task challenging, we let the target move at a speed of 1 m/s, which is the highest speed that a sensor can move at. The detection range of the target is 10 m, and the nodes in the detection range broadcast the target's location once every 1 second. The sensors have the same parameters as in the previous case study, that is, communication radius $r_c = 30m$ and sensing radius $r_s = 10m$. The time interval is 1 second. The stop criteria is set to 0.1 m. To reduce the energy consumption while maintaining the chasing ability, a sensor stays still whenever it is in the normal mode. In addition to the target's trajectory, Figs. 9b and 9e also present the trajectory for all the sensor nodes, which demonstrate that though the target moves at a fast speed, sensors can closely chase the target, regardless of the network size. The final topologies after the target stops are presented in Figs. 9c and 9f. One interesting phenomenon that needs to be mentioned is that in the final topology, all sensors tightly surround the target. Such a topology is advantageous because it can tolerate sensor failures. This is the result of having an attractive force pointing to the moving target for each sensor.

6.2.2 Slow Chase Strategy

Unlike the rapid chase strategy, in which the target applies an attractive force to every sensor node, the slow chase strategy only applies the attractive force to the sensor nodes that are within the target's detection range. Since nodes outside the target's detection range are only affected by the forces from their neighbors, the network's capability of chasing the target is thus determined by how closely those nodes follow the target, which, in turn, is determined by the relationship between the speed of the sensor nodes and that of the target, and how fast sensors get information about the target's location. In order to study the impact of this relationship on the chasing performance, we have looked at three scenarios. In all the three scenarios, we have 30 sensor nodes, and the sensors can move at the speed of 1 m/s. In

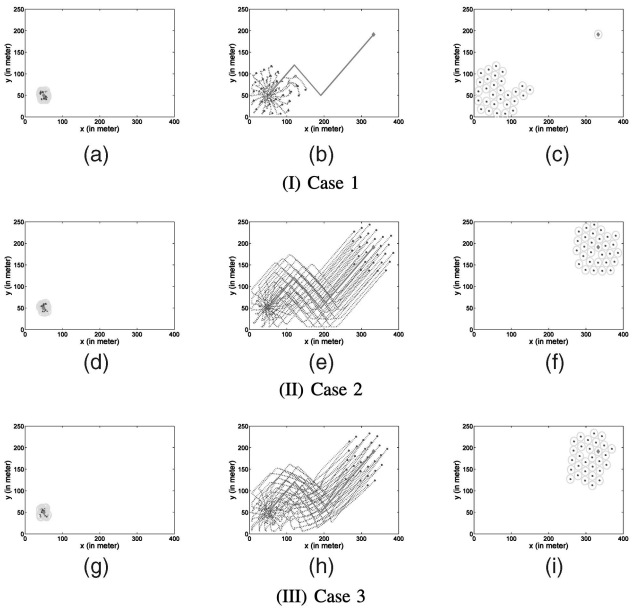

Fig. 10. Slow chase results. (a), (d), and (g) Initial deployment. (b), (e), and (h) Trajectories. (c), (f), and (i) Final deployment.

case 1, the time interval is 0.1 second, and the target moves at the speed of 0.5 m/s. In this case, the sensor network loses the target very soon, with the trajectory and the final topology shown in Figs. 10b and 10c. In case 2, the time interval is the same as that in case 1, but the target moves at a much slower speed: 0.2 m/s. The slower target in this case enables the rest of the network to follow it to the final destination, as shown in Figs. 10d, 10e, and 10f. In case 3, we keep the target speed the same as that in case 1, but we decrease the time interval to 0.02 seconds so that the sensors can get their neighbors' location information more often, and hence, the sensor network again successfully chases the target, as shown in Figs. 10g, 10h, and 10i.

Comparing the rapid chase strategy and slow chase strategy, the former is always able to follow the target, with an additional broadcast overhead. In the slow chase strategy, in order to successfully chase the moving target, either the sensors should be able to move much faster than the target does or the sensors should frequently exchange their position information.

7 Case Study 3: Spatial Retreats

In this section, we apply network dynamics to repair an ad hoc network that is subjected to attacks of radio interference.

7.1 Jamming Attacks and Spatial Retreats

The shared nature of the wireless medium makes wireless networks susceptible to a broad array of security threats. In particular, one class of powerful attacks that has received attention recently is jamming [22], [23], [24], [25], [26]. Adversaries may easily jam legitimate wireless communications either by continuously emitting radio frequency (RF) signals to occupy the wireless channel or by interrupting the transmission and reception of legitimate packets [26]. Either way, the net result is that legitimate traffic will be interfered

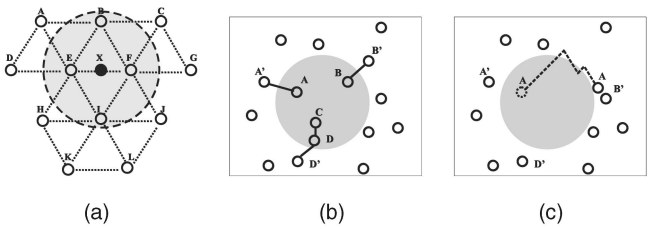

Fig. 11. (a) Jamming attacks in a network. (b) and (c) How a node escapes from a jammed area, where (b) illustrates the network topology when the jamming attack occurs with the jammed area highlighted by the gray area, and (c) uses the dashed line to mark the trace through which node A escapes from the jammed area and reconnects to the rest of the network.

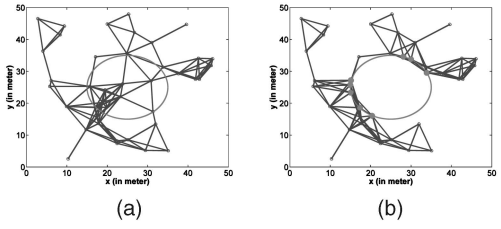

Fig. 12. Illustration of the escape strategy. (a) Initial topology. (b) Final topology.

with. Jamming attacks can have a particularly deleterious effect on MSNs, as the presence of a jammer may block whole regions of the network, as depicted in Fig. 11a.

To defend against jamming attacks, two strategies were recently proposed in [25], namely, channel surfing and spatial retreats. The underlying idea behind each strategy is to evade the interferer, either in the spectral sense or in the physical sense. In this paper, we are interested in spatial retreats, in which jammed nodes try to evacuate from jammed regions. As such, spatial retreats are suitable for MSNs. In the remainder of this section, we discuss our proposed spatial retreat strategies for MSNs and show how network dynamics may be incorporated into spatial retreats to achieve robustness in network communications.

The rationale behind spatial retreats is that when mobile nodes are interfered with, they should simply move to a safe location. Assuming that each mobile node can detect jamming attacks in a timely fashion [26], the key to the success of this strategy is to decide where the mobile nodes should escape to and how these nodes should coordinate their escapes. Merely escaping from a jammed region is not a sufficient defense mechanism, however, as a mobile adversary can move through the coverage area and cause large swaths of the MSN to relocate. By doing so, an adversary can cause the network to become unevenly distributed, or even partitioned, thereby severing network communications.

Therefore, spatial retreat strategies should be robust to mobile jammers. In order to achieve this robustness, our spatial retreat strategy has two phases:

- *Escape phase*. In this phase, the nodes that are located within the jammed area move to "safe" regions and stay connected with the rest of the network.
- *Reconstruction phase*. After the jammed nodes escape from the jammed area, a distributed network dynamics algorithm is applied to readjust the network deployment to be more uniformly covered. In particular, after the jammer has moved on, the jammed area will leave a hole in the network coverage, and the network dynamics will serve to quickly fill in the hole and restore the network coverage.

We begin by discussing the escape strategy that we have developed. Suppose the network is connected before the jamming attack; that is, every node within the jammed area is connected with nodes outside via one hop or through multiple hops. In the example shown in Fig. 11b, before the jamming attack, node A was directly connected with A', node B was directly connected with B', node D was directly connected with D', and C was connected with D' via D. After the jamming attack is detected, the nodes within the jammed area choose a random direction to evacuate. While moving, each node continuously runs the jamming detection algorithm until it leaves the jammed area. As soon as it leaves the jammed area, it tests whether there are some nodes within its radio range. If not, it moves along the boundary of the jammed area until it reconnects to the rest of the network. In Fig. 11b, if node A moves along the boundary, it will eventually arrive at a location that is between the location of A' and the original location of A, where it can reconnect to A'.

In order to make sure that a node moves along the boundary of the jammed area, the node must continually run the jamming detection algorithm. Following the hull-tracing strategy in [25], it can use the simple strategy: whenever it feels that the jammer's power is increasing, it makes a 90-degree left turn, and whenever it feels that the jammer's power is decreasing, it makes a 90-degree right turn. After it has moved a total r distance, where r is its radio range, the node will probe to see whether there is another node in its radio range (probing can also occur at a finer granularity). If not, it will continue moving along the boundary. Fig. 11c illustrates how node A chooses a random direction to escape from the jammed area and how it reconnects to the rest of the network by using the simple policy.

Figs. 12a and 12b present a set of simulation results for the escape strategy. In this set of experiments, we generate a random network topology, with 40 nodes deployed over a $50 \times 50\ \mathrm{m}^2$ network field. Each node's radio range is 10 m. The jammed area is a circle centered at location (25, 25), with a radius of 10 m. The topology before the victim's escape is shown in Fig. 12a. Fig. 12b shows the topology after these nodes escape to the boundary of the jammed area and reconnect to the rest of the network. These results show that our simple escape strategy is effective at escaping the jammed area while ensuring that evacuated nodes reconnect with the network.

We next turn to the reconstruction phase. Although our simple escape strategy guarantees that every jammed node can escape from the jammed area and successfully connect to the rest of the network, a serious problem remains. As we noted earlier, if the jammer is mobile, its movement may cause the network to become severely unbalanced, or even

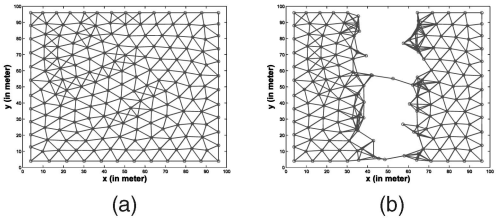

Fig. 13. A mobile jammer may partition the network. (a) The topology before the jamming attack. (b) The topology after the jammer moves from the bottom to the top.

partitioned. As an example, in Fig. 13, we depict an initial network configuration (Fig. 13a) and then introduce a jammer that moves in the y-direction through the middle of the network. The result is a network that is severely partitioned (Fig. 13b).

In order to address this problem, we propose to apply the network dynamics to continuously repair the network topology, regardless of the jammer movement. In this scenario, there are three types of forces in the network field: the forces between the nodes, the force from the boundary of the region, and the force from the boundary of the jammed area. We used the same model for the internode forces and region boundary forces, as we used in Section 5. The force that we used for the boundary of the jammed area is similar to the force for the boundary of the network field. Nonetheless, we cannot preprogram the jammed area boundary, as in the case of network field boundary. Instead, jammed area mapping techniques such as the one proposed in [22] should be incorporated here. In [22], upon detecting the presence of a jammer, nodes will report their situations to their neighbors, and based on these reports, nodes on the boundary of the jammed area can collectively map out the jammed area.

We now examine the behavior of our robust spatial retreat strategy by looking at an experiment involving a mobile jammer cutting across the network coverage area. Fig. 14 illustrates the evolution of the MSN's topology as the jammer moves through the network, and the robust spatial retreat algorithm not only evacuates the jammed area but also repairs regions left empty by the mobile jammer. In this experiment, we used PDND. Overall, the benefit of applying distributed network dynamics is twofold: 1) as soon as the victim nodes escape from the jammed area, instead of gathering around the boundary, the nodes redistribute to cover the rest of the network field more evenly and 2) as soon as the jammer leaves the current location, the resulting "hole" can be quickly repaired.

8 Related Literature

Mobility has become a recent trend in the area of mobile communications to look for scenarios where mobility can improve network performance. In [27], Grossglauser and Tse show that random mobility, in conjunction with multiuser diversity routing, can enhance the capacity of a communication network. Goldenberg et al. [28] examined cases where mobility improves the network performance. Their distributed algorithm adjusts the locations of mobile devices according to the local information to achieve global communication objectives.

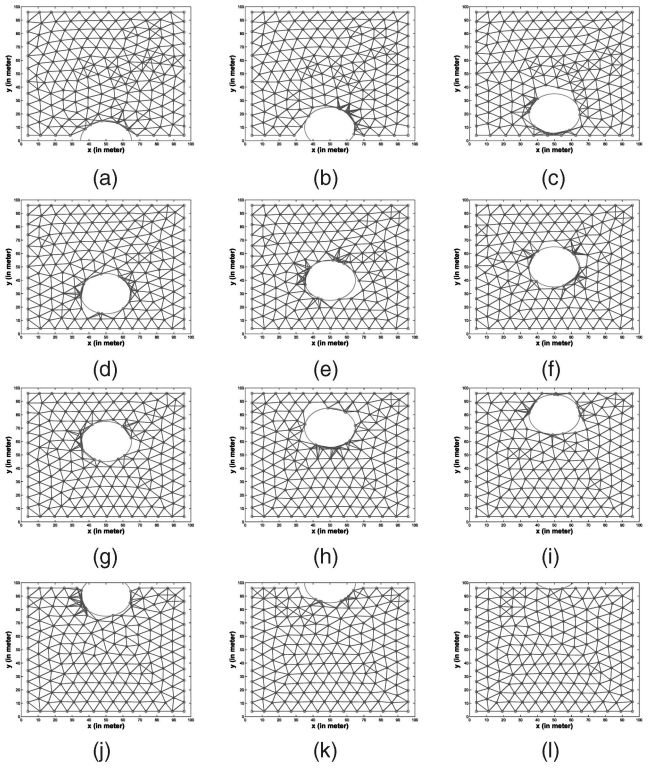

Fig. 14. The subfigures depict the ability of the robust spatial retreat algorithm to fight a jammer passing through the sensing field by using the PDND algorithm.

The mobility of an MSN can be used to improve the network's sensing coverage, following either a nonideal initial deployment of sensors or the presence of sensor failures [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [29], [30], [31]. The majority of these papers adopt the potential field (or virtual force) approach, which, as far as we know, was first proposed to navigate a robot in [2]. As its name suggests, the potential field approach builds a potential field based on the environment and the objective and guides the movement accordingly.

Regarding how we can perform the deployment, the aforementioned potential-field-based works can be classified into centralized ([4]) and distributed ([5], [6], [7], [8], [9], [10], [11], and [12]) approaches. In a centralized approach, the movement of all sensors is guided by a central entity that possesses the global information. In a distributed approach, each sensor directs its own movement based on the local information. The distributed approaches can be further divided into parallel ([5], [6], [7], [8], [11], and [12]) and sequential ([9] and [10]) approaches. In a parallel approach, all sensors can move simultaneously, whereas in a sequential approach, only one sensor is allowed to move at each moment.

Although some distributed approaches ([4], [11], and [12]) adopt heuristic methods with respect to how the deployment process stops, others ([5], [6], [7], [8], [9], [10]) obey Newton's laws of motion. If Newton's laws of motion are adopted (implicitly or explicitly), sensors are assumed to possess both potential and kinetic energy, and a friction force (damping factor) is introduced for each moving sensor. Because the friction force drains energy out, the

total energy possessed by a sensor is a Lyapunov function, which guarantees that the deployment process stops sooner or later.

As discussed in Section 2, adopting Newton's laws of motion has an obvious disadvantage: a sensor oscillates before it stops. An example can be found in [3]. Before the robot in [3] settles down, it moves around the objective position for a while. Oscillations should be avoided because they waste energy and prolong the deployment process.

The issue of jamming detection was briefly studied by Wood et al. in [22] in the context of sensor networks. Several different models for jammers, along with multimodal detection mechanisms, were presented in [26]. Countermeasures for coping with jammed regions in wireless networks has been studied in [24] and [25]. In [25], two countermeasures are presented for coping with jamming. The first method channel surfing involves a form of on-demand link-layer frequency hopping, where valid participants change the channel that they are communicating on when a denial of service attack occurs. The second method spatial retreats focused on the case of a stationary jammer and did not deal with the pernicious effects of a mobile jammer.

9 Concluding Remarks

In this paper, we have presented the framework for managing the mobility of an MSN, namely, network dynamics. More specifically, we have defined suitable potential functions that capture the operational goals and the environment of an MSN, and they are used to guide the movement of sensor nodes. Instead of following Newton's laws of motion, we argue that the equations of motion should follow the steepest descent formulation to minimize potential energy. Here, we devise a parallel and distributed algorithm (that is, PDND), under which devices make movement decisions based on the local knowledge. We have formally proven the convergence of PDND and explored its effectiveness and efficiency in three applications. In the first application, we have demonstrated that PDND can successfully maximize the sensing coverage of an MSN. Besides, we have also proposed an enhanced version of PDND, which shortens the convergence time by terminating the movement of a node as soon as it fully covers its Voronoi cell. We have also shown that PDND can work in scenarios that are impossible for a popular Voronoi-cell-based mobility management algorithm. In the second application, we have discussed how we can apply PDND to an MSN that tracks a moving target. We have demonstrated that if the speed of sensor nodes is comparable to that of the moving target, the network can successfully chase the target. Finally, we have examined the performance of network dynamics in managing the topology of an MSN in the presence of a jamming attack. By employing PDND in conjunction with a jamming escape algorithm, we have developed a robust strategy that is capable of repairing network partitioning as a jamming device cuts through the network. By conducting these three case studies, we show that the model of network dynamics and the PDND algorithm can efficiently utilize node mobility toward robust and efficient sensor network operations.

References

[1] Adaptive Sampling and Prediction, http://engineering.princeton.edu/gallery/asap/index.htm, 2006.

[2] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '85),* pp. 500-505, 1985.

[3] B.Q. Nguyen, Y.-L. Chuang, D. Tung, C. Hsieh, Z. Jin, L. Shi, D. Marthaler, A. Bertozzi, and R.M. Murray, "Virtual Attractive-Repulsive Potentials for Cooperative Control of Second Order dynamic Vehicles on the Caltech MVWT," *Proc. 2005 Am. Control Conf. (ACC '05),* pp. 1084-1089, 2005.

[4] Y. Zou and K. Chakrabarty, "Sensor Deployment and Target Localization Based on Virtual Forces," *Proc. IEEE INFOCOM '03,* pp. 1293-1303, Mar. 2003.

[5] A. Howard, M. Mataric, and G. Sukhatme, "Mobile Sensor Network Deployment Using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem," *Proc. Sixth Int'l Symp. Distributed Autonomous Robotics Systems (DARS '02),* June 2002.

[6] S. Poduri and G.S. Sukhatme, "Constrained Coverage for Mobile Sensor Networks," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '04),* pp. 165-171, 2004.

[7] P. Ögren, E. Fiorelli, and N.E. Leonard, "Cooperative Control of Mobile Sensor Networks: Adaptive Gradient Climbing in a Distributed Environment," *IEEE Trans. Automatic Control,* vol. 49, no. 8, pp. 1292-1302, 2004.

[8] N.E. Leonard and E. Fiorelli, "Virtual Leaders, Artificial Potentials and Coordinated Control of Groups," *Proc. 40th IEEE Conf. Decision and Control (CDC '01),* pp. 2968-2973, 2001.

[9] J. Tan and O.M. Lozano, "A Distributed Graph Model for the Coordination and Formation of Multi-Robot Systems," *Int'l J. Robotics Research,* to be published.

[10] J. Tan and N. Xi, "Peer-to-Peer Model for the Area Coverage and Cooperative Control of Mobile Sensor Networks," *Proc. SPIE Symp. Defense and Security,* pp. 439-450, 2004.

[11] N. Heo and P.K. Varshney, "Energy-Efficient Deployment of Intelligent Mobile Sensor Networks," *IEEE Trans. Systems, Man and Cybernetics,* vol. 35, no. 1, pp. 78-92, 2005.

[12] G. Wang, G. Cao, and T. Porta, "Movement-Assisted Sensor Deployment," *Proc. IEEE INFOCOM '04,* pp. 2469-2479, Mar. 2004.

[13] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage Control for Mobile Sensing Networks," *IEEE Trans. Robotics and Automation,* vol. 20, no. 2, pp. 243-255, 2004.

[14] P.G. Fernandes, P. Stevenson, A.S. Brierley, F. Armstrong, and E.J. Simmonds, "Autonomous Underwater Vehicles: Future Platforms for Fisheries Acoustics," *ICES J. Marine Science,* vol. 60, no. 3, pp. 684-691, 2003.

[15] R.P. Feynman, *Feynman Lectures on Physics.* Addison Wesley, 1970.

[16] C.G. Gray and K.E. Gubbins, *The Theory of Molecular Fluids 1: Fundamentals.* Clarence Press, 1984.

[17] P. Enge and P. Misra, *Global Positioning System: Signals, Measurements and Performance.* Ganga-Jamuna Pr, 2001.

[18] K. Langendoen and N. Reijers, "Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison," *Computer Networks,* vol. 43, no. 4, pp. 499-518, 2003.

[19] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, 1997.

[20] A. Ghosh, "Estimating Coverage Holes and Enhancing Coverage in Mixed Sensor Networks," *Proc. 29th Ann. IEEE Int'l Conf. Local Computer Networks (LCN '04),* pp. 68-76, 2004.

[21] T. Liu, C.M. Sadler, P. Zhang, and M. Martonosi, "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and Zebranet," *Proc. Second Int'l Conf. Mobile Systems, Applications, and Services (MobiSys '04),* pp. 256-269, June 2004.

[22] A. Wood, J. Stankovic, and S. Son, "JAM: A Jammed-Area Mapping Service for Sensor Networks," *Proc. 24th IEEE Real-Time Systems Symp. (RTSS '03),* pp. 286-297, 2003.

[23] A. Wood and J. Stankovic, "Denial of Service in Sensor Networks," *Computer,* vol. 35, no. 10, pp. 54-62, Oct. 2002.

[24] G. Noubir and G. Lin, "Low-Power DoS Attacks in Data Wireless LANs and Countermeasures," *ACM SIGMOBILE Mobile Computing and Comm. Rev.,* vol. 7, no. 3, pp. 29-30, 2003.

[25] W. Xu, T. Wood, W. Trappe, and Y. Zhang, "Channel Surfing and Spatial Retreats: Defenses against Wireless Denial of Service," *Proc. 2004 ACM Workshop Wireless Security (WiSe '04),* pp. 80-89, 2004.

[26] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The Feasibility of Launching and Detecting Jamming Attacks in Wireless Networks," *Proc. ACM MobiHoc '05,* pp. 46-57, 2005.

[27] M. Grossglauser and D. Tse, "Mobility Increases the Capacity of Ad Hoc Wireless Networks," *Proc. IEEE INFOCOM '01*, pp. 1360-1369, Mar. 2001.
[28] D. Goldenberg, J. Lin, and A. Morse, "Towards Mobility as a Network Control Primitive," *Proc. ACM MobiHoc '04*, pp. 163-174, May 2004.
[29] G. Wang, G. Cao, T. Porta, and W. Zhang, "Sensor Relocation in Mobile Sensor Networks," *Proc. IEEE INFOCOM '05*, pp. 2302-2312, Mar. 2005.
[30] S. Ganeriwal, A. Kansal, and M.B. Srivastava, "Self-Aware Actuation for Fault Repair in Sensor Networks," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '04)*, pp. 5244-5249, Apr. 2004.
[31] A. Howard, M. Mataric, and G. Sukhatme, "An Incremental Deployment Algorithm for Mobile Robot Teams," *Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems (IROS '02)*, pp. 2849-2854, Sept. 2002.

Ke Ma received the BS degree in information science from Beijing University of Posts and Telecommunications, Beijing, in 1998 and the MPhil degree in information engineering from the Chinese University of Hong Kong, Hong Kong, in 2003. He is currently working toward the PhD degree at the Wireless Information Network Laboratory (WINLAB) and the Department of Electrical and Computer Engineering, Rutgers University. His research interests include mobility management in mobile sensor networks and quality of service (QoS) in wireless ad hoc networks.

Yanyong Zhang received the PhD degree in computer science and engineering from Pennsylvania State University in 2002. She is an assistant professor in the Department of Electrical and Computer Engineering, Rutgers University, and she is also a faculty member at Wireless Information Network Laboratory (WINLAB). Her current research interests include sensor networks, sensor network security and privacy, and fault-tolerant sensor networks. She has received several US National Science Foundation (NSF) grants on these topics, including an NSF Faculty Early Career Development (CAREER) Award. She is a member of the ACM, the IEEE, and the IEEE Computer Society.

Wade Trappe received his BA degree in mathematics from the University of Texas at Austin in 1994 and the PhD degree in applied mathematics and scientific computing from the University of Maryland in 2002. He is currently an assistant professor at the Wireless Information Network Laboratory (WINLAB) and the Department of Electrical and Computer Engineering, Rutgers University. His research interests include wireless security, wireless networking, multimedia security, and network security. He is a coauthor of *Introduction to Cryptography with Coding Theory* (Prentice Hall, 2001). He is a member of the IEEE, IEEE Signal Processing Society, IEEE Communications Society, and the ACM. While at the University of Maryland, he received the George Harhalakis Outstanding Systems Engineering Graduate Student Award. He is the recipient of the 2005 Best Paper Award from the IEEE Signal Processing Society.