Chapter 1

INTENTIONAL MOBILITY IN WIRELESS SENSOR NETWORKS

You-Chiun Wang and Yu-Chee Tseng*
Department of Computer Science
National Chiao-Tung University, Hsin-Chu, 30010, Taiwan

Abstract

The emerging wireless sensor networks (WSNs) offer a convenient way to monitor physical environments. In the past, WSNs are deployed by static nodes to continuously collect information from the environment. Today, by introducing the concept of controllable mobility to WSNs, we can further improve the network capability on many aspects, for example, automatic WSN deployment, flexible topology adjustment, and rapid reaction to events. In this chapter, we provide a comprehensive survey of recent researches on intentional mobility of WSNs. The discussion contains four parts. First, we survey some papers related to mobility-assisted network deployment. Second, we introduce functional enhancement schemes by mobile sensors. Third, we discuss the dispatch issue of mobile sensors in a hybrid WSN. Finally, we present the design of mobile platforms to support mobility for sensors and give several applications by mobile WSNs.

Keywords: coverage, deployment, dispatch, mobile sensors, sensor applications, wireless sensor networks.

1. Introduction

The rapid growth of micro-sensing MEMS and wireless communication technologies has promoted the development of wireless sensor networks (WSNs). Such a network is composed of one or multiple remote sinks and many tiny, low-power sensor nodes, each equipped with some actuators, sensing devices, and a wireless transceiver [1]. These nodes are massively deployed in a region of interest to collect information from their surroundings, and continuously report back to the remote sinks. Thus, WSNs can provide a convenient

*E-mail address: {wangyc, yctseng}@cs.nctu.edu.tw
way to monitor physical environments. In recent years, a large amount of WSN-related applications such as object tracking, health monitoring, security surveillance, and intelligent transportation have been proposed.

A WSN is usually deployed with static sensor nodes to perform monitoring missions in the region of interest. However, due to the dynamic changes of events and hostile environment, a pure static WSN could face the following severe problems:

- The initial deployment of a WSN may not guarantee complete coverage of the sensing field and connectivity of the whole network. Usually, sensor nodes may be scattered in a hostile region from the aircraft or by robots [2]. However, these randomly deployed sensors could not guarantee to cover the whole area and may be partitioned into several non-connected subnetworks, even though we scatter a huge amount of nodes. Moreover, the dynamic change of regions of interest and the existence of obstacles could make the problem become more difficult.

- Sensor nodes are usually battery-powered and prone to errors. As some nodes die due to the exhaustion of their energy, there could exist holes in the WSN’s coverage. In addition, these death nodes may break the network connectivity. However, in many scenarios, it is quite difficult to recharge sensor nodes or deploy new nodes to replace these death nodes.

- The WSN may be required to support multiple missions under various conditions [3]. For example, in an object tracking application, sufficient sensor nodes should be deployed along the track of the target, while in a boundary detection mission, there should be adequate nodes along the pre-described perimeter. These different requirements cannot be easily satisfied by deploying a large amount of sensor nodes, since provisioning for all possible combinations of mission requirements could not be economically feasible.

- Some applications may need sophisticated (and thus expensive) sensors to involve in. For example, one can image that in a military application, pressure sensors may be deployed along the boundary to detect whether any enemy intrudes in. However, these sensors can only report something passing but cannot describe what passes through them. In this case, more sophisticated sensing devices like cameras should be required to obtain more information. Nevertheless, it is infeasible to equip camera on each node because of their large number.

By introducing mobility to some or all nodes in a WSN, we can enhance its capability and flexibility to support multiple missions and to handle the aforementioned problems. Although a WSN is usually considered as an ad hoc network in which nodes are extended with sensing capability, a mobile WSN and a mobile ad hoc network (MANET) are essentially different. Mobility in a MANET is often arbitrary, whereas mobility in a mobile WSN should be “intentional”. In other words, we can control the movement of mobile sensors to conduct different missions.

In this chapter, we give a comprehensive survey of recent researches on intentional mobility in WSNs. The discussion includes four parts:
• **Mobility-assisted network deployment**: We will introduce several deployment schemes that investigate how to reorganize a randomly deployed WSN into a more regular topology, such that the WSN can cover the most area of the sensing field.

• **Functional enhancement by mobile sensors**: We will present some relocation methods of mobile sensors to improve the functionalities of a WSN, including area coverage, network connectivity, and detection ability.

• **Dispatch of mobile sensors in a hybrid WSN**: We will discuss the dispatch issue of mobile sensors in a hybrid WSN that consists of both static and mobile nodes.

• **Design of mobile platforms and applications**: We will survey the current design of mobile platforms to support mobility for sensor nodes, and give some interesting applications on mobile WSNs.

2. Mobility-Assisted Network Deployment

Sensor deployment is one important issue in WSNs because it directly affects the detection capability of the network. Several researches have proposed different mobility-assisted deployment schemes to help form a WSN. They consider that there are a large amount of mobile sensors randomly dropped in a sensing field. Then, different moving strategies are conducted to make these sensors to automatically organize a connected network that can cover the maximal target area. In this section, three categories of mobility-assisted deployment schemes are introduced. The force-directed deployment schemes image that there exist attractive and repulsive forces between sensors and these forces will drive sensors to move. The Voronoi-based deployment schemes use a graph-based method to estimate potential coverage holes and move sensors to cover these holes. Finally, the predetermined deployment schemes first calculate the locations to place sensors and then dispatch sensors to these locations in an energy-efficient manner.

2.1. Force-Directed Deployment Schemes

In [4], the concept of virtual force is introduced to move sensors. Each sensor $s_i$ is assumed to be exerted by three types of forces: the attractive (positive) force $F_{iA}$ by the target area, the repulsive (negative) force $F_{iR}$ by all obstacles, and the force $F_{ij}$ between sensors $s_i$ and $s_j$. Therefore, the total force $F_i$ on sensor $s_i$ can be expressed as

$$F_i = F_{iA} + F_{iR} + \sum_{j=1,j\neq i}^{n} F_{ij},$$

where $n$ is the number of sensors. The force $F_{ij}$ is expressed by a polar coordinate notation $(r, \theta)$, where $r$ is the magnitude and $\theta$ is the orientation (angle) of the vector $F_{ij}$. Thus, $F_{ij}$ can be derived as

$$F_{ij} = \begin{cases} (w_A \cdot (d_{ij} - d_{ih}), \theta_{ij}) & \text{if } d_{ij} > d_{ih} \\ (w_R \cdot \frac{1}{d_{ij}}, \pi + \theta_{ij}) & \text{if } d_{ij} < d_{ih} \\ 0 & \text{otherwise} \end{cases}$$ (1)
You-Chiun Wang and Yu-Chee Tseng

Figure 1. An example of the virtual forces on sensor $s_1$ by three other sensors. (a) Attractive and repulsive forces in [4]. (b) Only repulsive forces in [5].

where $d_{ij}$ is the Euclidean distance between sensors $s_i$ and $s_j$, $d_{th}$ is a threshold distance, $\theta_{ij}$ is the orientation of a line segment from $s_i$ to $s_j$, and $w_A$ and $w_R$ are measures of the attractive and repulsive forces, respectively. Note that the threshold $d_{th}$ controls how close sensors get to each other. Fig. 1(a) shows an example, where there are four sensors and $d_{th} = d_{12}$. According to Eq. (1), $s_2$ exerts no force on $s_1$, $s_3$ exerts an attractive force $F_{13}$ on $s_1$, and $s_4$ exerts a repulsive force $F_{14}$ on $s_1$ because $d_{12} = d_{th}$, $d_{13} > d_{th}$, and $d_{14} < d_{th}$, respectively. Sensor $s_1$ is thus moved by the compound force $F_{1}$.

The work [5] considers only repulsive forces, where each sensor is treated as an electron and will be repulsed by other sensors. The force corresponding to a higher local density is greater than the force corresponding to a lower local density, and the force from a closer node is greater than the force from a farther node. In particular, [5] suggests that a force function $f(\cdot)$ should be defined to satisfy the following rules:

1. $f(d_{ij}) \geq f(d_{ik})$ if $d_{ij} \leq d_{ik}$. This implies that forces will be inversely proportional to sensors’ distances.

2. $f(0^+) = f_{\text{max}}$, where $f_{\text{max}}$ is the maximum force. This puts an upper bound on forces.

3. $f(d_{ij}) = 0$ if $d_{ij}$ is larger than the communication distance of a sensor. This means that only neighboring sensors are considered to generate the force.

Sensors will be moved step by step. In each step $k$, the force on sensor $i$ by another sensor $j$ in its communication range is calculated to be a repulsive force as

$$f_{k}^{i,j} = \frac{D_k^j}{\mu^2} \frac{r_c}{|p_k^j - p_k^i|} \frac{|p_k^j - p_k^i|}{|p_k^j - p_k^i|},$$

where $D_k^j$ is the local density of sensor $i$ at step $k$, $\mu$ is the expected density (after the final deployment), $r_c$ is the communication distance of a sensor, and $p_k^i$ is the location of
sensor $i$ at step $k$. Fig. 1(b) gives an example, where sensors $s_2$, $s_3$, and $s_4$ are inside $s_1$’s communication range. Sensor $s_1$ will be moved by the compound force $f_1^k$. Note that when a sensor moves less than a threshold distance for a specified time period, it is considered to have reached the stable status and the sensor thus stops moving.

### 2.2. Voronoi-Based Deployment Schemes

Reference [6] uses a Voronoi diagram to find out potential coverage holes and then moves sensors to cover these holes. Given a set of nodes on a 2D plane, the Voronoi diagram [7] is formed from perpendicular bisectors of lines that connect two neighboring nodes, as shown in Fig. 2(a). The Voronoi diagram can represent the proximity information about a set of geometric nodes. Every point inside a Voronoi polygon is closer to the node in this polygon than to any other node. With this property, if the sensing range of a sensor cannot completely cover the corresponding Voronoi polygon it located, there may exist coverage hole inside that polygon. Thus, each sensor can check coverage hole locally since it only needs to know the locations of its Voronoi neighbors to construct the Voronoi polygon.

In [6], after the initial deployment, each sensor first calculates the corresponding Voronoi polygon with the location information from its neighbors. Then, three strategies are proposed to move sensors:

1. **Vector-based (VEC) strategy**: The VEC strategy also adopts the idea from virtual force. Two sensors $s_i$ and $s_j$ will be pushed to move a distance of $\frac{1}{2}(d_{\text{avg}} - d_{ij})$ away from each other, where $d_{\text{avg}}$ is the average distance between two sensors when the sensors are evenly distributed in the sensing field. However, if one sensor can completely cover its Voronoi polygon, it should not be moved. In this case, the other sensor will be pushed away with a distance of $d_{\text{avg}} - d_{ij}$.

2. **Voronoi-based (VOR) strategy**: When a sensor detects the existence of a coverage hole, it will move toward its farthest Voronoi vertex. Fig. 2(b) gives an example, where point $u$ is the farthest Voronoi vertex. Sensor $s_i$ will move along $s_iu$ and stop at point $v_1$, where $|uv_1|$ is equal to the sensing distance $r_s$. Note that when a sensor cannot obtain location information from all its Voronoi neighbors, it will lead the sensor to calculate an incorrect Voronoi polygon. In this case, we should limit the maximum moving distance to $\frac{r_s}{2}$ to prevent the sensor from moving more than needed.

3. **Minimax strategy**: Like VOR, the sensor also moves toward its farthest Voronoi vertex, but stops at a minimax point $v_2$ whose distance to all Voronoi vertices can be minimized, as shown in Fig. 2(c). The basic concept of minimax strategy is that a sensor should not be too far away from any of its Voronoi vertices when all sensors are evenly distributed. To calculate the minimax point, we should check all circles that pass through any two and any three Voronoi vertices. Among these circles, the one with the minimum radius that covers all Voronoi vertices will be the minimax circle, whose center is thus the minimax point.
2.3. Predetermined Deployment Schemes

In [8], Wang et al. target at planned deployment in known fields. They address the deployment issue by considering two related problems: sensor placement and sensor dispatch. The placement problem determines how to place the least number of sensors in a field to guarantee complete coverage and network connectivity. Based on the sensor placement, the dispatch problem asks how to delegate mobile sensors to the target locations specified in the placement result such that their moving energy consumption can be minimized or their remaining energy can be maximized.

For sensor placement, [8] allows an arbitrary relationship between a sensor’s communication distance $r_c$ and its sensing distance $r_s$. Given an arbitrary-shaped sensing field with possibly arbitrary-shaped obstacles, [8] first partitions the field into single-row and multi-row regions. A single-row region is a belt-like area with a width not larger than $\sqrt{3}\min\{r_c, r_s\}$, where $r_{\text{min}} = \min\{r_c, r_s\}$, so one row of sensors are enough to cover the whole region while maintaining network connectivity. A multi-row region is perceivably larger and thus required several rows of sensors to cover it. To partition a field $\mathcal{A}$, we should first identify all single-row regions. Other regions are thus multi-row regions. This can be done by expanding $\mathcal{A}$’s boundaries inward and obstacles’ perimeters outward by a distance of $\sqrt{3}r_{\text{min}}$, as shown in Fig. 3(a). When the expanded line cuts off an obstacle with an area, say $O$, we can take a project from $O$ to obtain a single-row region. Fig. 3(a) shows an example, where four single-row regions (marked with alphabets) and four multi-row regions (marked with numbers) are found. Then, we can place sensors inside these regions by the following rules:
- **Single-row region:** For a single-row region, we can find its bisector and then place a sequence of sensors, each separated by a distance of \( r_{\text{min}} \), along the bisector, as shown in Fig. 3(b). This can ensure both coverage of the region and connectivity of the network. A bisector can be found by doing a triangulation on that region, and connecting the midpoints of all dotted lines. Note that we should add one sensor at the end of bisector to connect with neighboring regions.

- **Multi-row region:** Two cases are considered, as shown in Fig. 3(c). In the case of \( r_c < \sqrt{3} r_s \), sensors on each row are separated by a distance of \( r_c \), so the connectivity of each row is guaranteed. Adjacent row are separated by a distance not larger than \( r_c \), to connect them. On the other hand, in the case of \( r_c \geq \sqrt{3} r_s \), adjacent sensors are regularly separated by a distance of \( \sqrt{3} r_s \), so both coverage and connectivity are ensured. Note that we have to sequentially place sensors along the boundaries of multi-row region and the perimeters of obstacles to fill the uncovered space and to maintain connectivity with neighboring regions. These sensors should be separated by a distance of \( r_c \) and \( \sqrt{3} r_s \) when \( r_c \geq \sqrt{3} r_s \) and \( r_c < \sqrt{3} r_s \), respectively.

For sensor dispatch, [8] proposes a centralized scheme by converting the dispatch problem to a maximum-weight maximum-matching problem. This scheme involves the following steps:

1. Given a set of mobile sensors \( S \) and a set of locations \( L = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \) computed by the above placement scheme, where \(|S| \geq |L|\), we construct a weighted complete bipartite graph \( G = (S \cup L, S \times L) \). The weight of each edge \((s_i, (x_j, y_j))\), \( s_i \in S \), \((x_j, y_j) \in L \), can be defined either as
   \[
   w(s_i, (x_j, y_j)) = -(\Delta_m \times d(s_i, (x_j, y_j))),
   \]
   if we want to minimize the total energy consumption to move sensors, or as
   \[
   w(s_i, (x_j, y_j)) = e_i - (\Delta_m \times d(s_i, (x_j, y_j))),
   \]
   if we want to maximize the average remaining energy of sensors moved to target locations, where \( e_i \) is current energy of a sensor \( s_i \), \( \Delta_m \) is the unit energy cost to move a sensor in one step, and \( d(s_i, (x_j, y_j)) \) is the shortest distance from \( s_i \) to location \((x_j, y_j)\).

2. Find a maximum matching with the maximum weight on graph \( G \). In particular, we construct a new graph \( G' = (S \cup L \cup L', S \times \{L \cup L'\}) \) from \( G \), where \( L' \) is a set of \(|S| - |L|\) elements, each called a virtual location. The weight of each edge in \( G' \) that also appears in \( G \) remains the same as that in \( G \), and the weight of each edge from \( s_i \in S \) to \((x_j, y_j) \in L' \) is set to \( w_{\text{min}} \), where
   \[
   w_{\text{min}} = \min_{s_i \in S, (x_j, y_j) \in L} \{w(s_i, (x_j, y_j))\} - 1.
   \]
You-Chiun Wang and Yu-Chee Tseng

(a) partitioning a sensing field

(b) place sensors in a single-row region

(c) place sensors in a multi-row region

Figure 3. The sensor placement method proposed in [8].
Intuitively, a virtual location is a dummy one. Its purpose is to make the sizes of sets $S$ and $\{L \cup L'\}$ become equal. This allows us to transform the problem to the maximum-weight perfect-matching problem on $G'$, whose purpose is to find a perfect matching $M$ in $G'$ with the maximum edge weights. Then, we can adopt the Hungarian method [9] to find $M$.

3. For each edge $(s_i, (x_j, y_j))$ in $M$ such that $(x_j, y_j) \in L$, we move sensor $s_i$ to location $(x_j, y_j)$ through the shortest path. However, if there is any edge $(s_i, (x_j, y_j)) \in M$ such that $(x_j, y_j) \in L$ and $e_i - c(s_i, (x_j, y_j)) \leq 0$, it means that we do not have sufficient energy to move sensors to all locations in $L$, so the algorithm terminates.

The remaining problem is how to find the shortest path from a sensor $s_i$ to a location $(x_j, y_j)$, without colliding with any obstacle. In [8], the authors propose a solution as shown in Fig. 4. Considering its physical size, $s_i$ is modeled as a circle with a radius $r$. Intuitively, $s_i$ has a collision-free motion if its center always keeps at least a distance of $r$ away from obstacles and $A$’s boundaries. This can be done by expanding the obstacles’ perimeters outward and $A$’s boundaries inward by a distance of $r$ and preventing $s_i$ from moving into these expanded areas. Then, the problem can be translated to one of finding the shortest path from $s_i$ to $(x_j, y_j)$ in a weighted graph $H = (s_i \cup (x_j, y_j) \cup V', E)$, where $V'$ contains all vertices $v$ of the polygons representing the expanded areas of obstacles and $A$’s boundaries such that $v$ is not inside other expanded areas, and $E$ contains all edges $(u, v)$ such that $u, v \in \{s_i \cup (x_j, y_j) \cup V'\}$ and $\overline{uv}$ does not pass any expanded area of obstacles or $A$’s boundaries. The weight of each edge $(u, v) \in E$ is the length of $\overline{uv}$. Fig. 4 gives an example. Nodes $e$ and $f$ are not vertices of $H$ because they are inside obstacles 2’s and 3’s expanded areas, respectively. Edges $(a, b)$, $(a, g)$, $(h, b)$, and $(h, g) \in E$, but $(h, c)$ and $(h, d) \notin E$ because they pass the expanded area of obstacle 2. After constructing such a graph $H$, we can use the Dijkstra’s algorithm [10] to calculate the shortest path from $s_i$ to $(x_j, y_j)$.
3. Functional Enhancements by Mobile Sensors

Mobile sensors can be also used to enhance the functionalities of a WSN. We can relocate some of mobile sensors to the “weakest” points of a WSN to strengthen it. In this section, we present three types of such enhancement methods. The coverage enhancement methods attempt to relocate sensors from high-density regions to low-density regions to improve the overall coverage of the network. The connectivity enhancement methods first find out potential weak points on network connectivity and then relocate some sensors to strengthen these points. Finally, the event-based motion methods relocate more sensors close to event locations to get a better detection of events.

3.1. Coverage Enhancement Methods

The work [11] uses a grid structure to perform sensor relocation. It partitions the sensing field into $N \times N$ grids and then moves sensors from high-density grids to low-density grids. This can help redistribute a more event network, especially when some nodes fail. In [11], the grid-quorum and cascaded movement schemes are proposed to fast find the redundant sensors and to relocate these sensors to the target grids, respectively. In the grid-quorum scheme, each grid will elect a grid head to maintain the grid’s information. The grid head of a high-density grid $g_i$ will send an advertisement (ADV) message in its row to announce that it has redundant sensors. On the other hand, the grid head of a low-density grid $g_j$ will send a request (REQ) message to its column to query redundant sensors. Because of the grid structure, there must be a grid $g_k$ that receives both the advertisement and request messages. Thus, grid $g_k$ can reply to grid $g_j$ that there are redundant sensors in grid $g_i$. Fig. 5(a) gives an example, where grid $(1,2)$ has redundant sensors and grid $(3,0)$ needs them. Grid $(3,2)$ will receive both advertisement and request messages from these two grids. Compared to the traditional flooding scheme, grid-quorum can reduce message overhead from $O(N)$ to $O(\sqrt{N})$.

After identifying redundant sensors and target grids, sensors are moved using the cascaded movement rather than the direct movement to prevent these sensors from wasting too much energy, as shown in Fig. 5(b) and (c). In the cascaded movement scheme, each sensor $s_i$ is associated with a tolerable delay $T_i$, during which its successor must move to its original location. Thus, a sensor $s_j$ can become a successor of $s_i$ if

$$\frac{d_{ij}}{v} - (t_i - t_j) \leq T_i,$$

where $d_{ij}$ is the distance between $s_i$ and $s_j$, $v$ is the moving speed of a sensor, and $t_i$ is the departure time of $s_i$. For example, in Fig. 5(c), $s_k$ can first move to the target location, and then $s_j$ can move to $s_k$’s original location. Finally, $s_j$ can move to $s_i$’s original location. Such a sequence $s_j \rightarrow s_i \rightarrow s_k$ is called a cascading schedule. Clearly, the cascading schedule is not unique and should minimize the total energy consumption and maximize the minimum remaining energy so that no individual sensor will waste too much energy. In [11], the schedule with the minimum difference between the total energy consumption and the minimum remaining energy is selected. In particular, suppose that there are two cascading schedules with $E_1$ and $E_2$ as their total energy consumption, and $E_1^{\text{min}}$ and $E_2^{\text{min}}$ as their minimum remaining energy. Schedule 1 will be selected if $E_1 - E_1^{\text{min}} \leq E_2 - E_2^{\text{min}}$. 
3.2. Connectivity Enhancement Methods

Reference [12] discusses how to move nodes to reorganize a stronger network. Specifically, the objective is to form a biconnected network such that the total moving distance of nodes can be minimized. A network is called biconnected if it is not partitioned after removing any of its nodes. Each such node is referred as a cutvertex. Fig. 6 shows an example, where node $d$ is a cutvertex and a biconnected network can be formed by moving node $f$ close to node $a$. Based on this observation, [12] proposes a block movement algorithm to achieve a biconnected network by removing all cutvertices from the original network. In particular, given a graph $G$ that describes the network topology, the biconnected components (also called blocks) of $G$ are first identified along with its cutvertices, and then the graph $G$ can be translated to a block tree [13]. Note that a block can have between 0 and $n$ nodes, where $n$ is the total number of nodes in the network. If two cutvertices are directly connected with an edge, the corresponding block contains no node. In the block tree, we can select the block with the maximum number of nodes as the root. Fig. 7(a) and (b) give an example, where five blocks (including the empty block $B_3$) and two cutvertices $C_1$ and $C_2$ are identified. Block $B_1$ is selected as the root and three blocks $B_2$, $B_4$, and $B_5$ are leaves. The block movement algorithm then executes the following iterations until the graph becomes biconnected:

- Move each leaf block toward the nearest node in its parent block, by the distance that exact one new edge will appear.

- If the parent block contains no node, we move the leaf block to the upstream cutvertex of its parent block.
Fig. 7(c) illustrates an example, where block $B_2$ will move toward the node $v$ in block $B_1$ and the blocks $B_4$ and $B_5$ will move toward the cutvertex $C_2$ because their parent block $B_3$ is empty. The final network topology is shown in Fig. 7(d).

Figure 6. Achieving a biconnected network by moving node $f$ toward node $a$.

Figure 7. An example of the block movement algorithm propose in [12].
3.3. Event-Based Motion Methods

The work in [14] considers how to move more sensors close to event locations, while still maintaining complete coverage of the sensing field. Given a set of event locations, sensors should move in a way such that their final positions can eventually approximate the event distribution. In [14], two moving strategies are proposed to achieve this goal. In the history-free strategy, each sensor will react to an event by moving according to a function

\[ p_i^{k+1} = p_i^k + f(l_i^{k+1}, p_i^k, p_i^0), \]

where \( l_i^{k+1} \) is the location of event \( k+1 \) and \( p_i^k \) refers to the position of sensor \( i \) after event \( k \). The function \( f \) can be introduced with a dependency on the distance \( d \) between the sensor and the event, and should satisfy the following three criteria:

- \( 0 \leq f(d) \leq d, \forall d. \) This means that after an event occurs, the sensor should never move past that event.
- \( f(\infty) = 0. \) This indicates that the sensors’ motion should tend to zero as the event gets father away.
- \( f(d_1) - f(d_2) < d_1 - d_2, \forall d_1 > d_2. \) This implies that no sensor should move past another along the same vector in response to the same event.

In [14], the authors thus suggest two possible functions to satisfy the above criteria. The first one is \( f(d) = de^{-d} \), where \( e \approx 2.718 \). The other function is \( f(d) = \alpha d^\beta e^{-\gamma d} \) for values of parameters \( \alpha, \beta, \) and \( \gamma \) such that \( \alpha e^{-\gamma} (\beta d^{\beta-1} - \gamma d^\beta) > 1, \forall d. \) For example, we can set \( \alpha = 0.06, \beta = 3, \) and \( \gamma = 1. \) On the other hand, the history-based strategy requires sensors to maintain event history to get a better approximation of event distribution. The idea is to use the events’ cumulative distribution function (CDF) to calculate the final positions of sensors. Specifically, given the x-axis location \( l_x^k \) of an event \( k \), each sensor will conduct the following steps:

1. Increment CDF bins representing positions \( \geq l_x^k \).
2. Scale CDF by \( k/(k+1) \).
3. Find bins \( b_i \) and \( b_{i+1} \) with values \( b_i \leq x_0 \leq b_{i+1} \), where \( x_0 \) is the initial x-axis position of the sensor.
4. Compute the final x-axis position of the sensor by interpolating the values of \( b_i \) and \( b_{i+1} \).

since events occurs in a 2D plane, we should also calculate the final y-axis positions of sensors by the above steps.

By the aforementioned moving strategies, sensors will move close to event locations. However, to maintain complete coverage of the sensing field, [14] proposes to use a Voronoi diagram to determine whether the movement of a sensor will cause a coverage hole. As mentioned earlier in Section 2.2., if any part of the sensor’s Voronoi polygon is farther away than the sensing distance, the sensor can know that its leaving will cause a coverage hole. In this case, the sensor should stop moving to keep its original coverage.
4. Dispatch of Mobile Sensors in a Hybrid WSN

The above two sections assume a WSN containing only mobile sensors. Some works consider a hybrid WSN consisting of both static and mobile sensors. Static sensors are used to detect events occurred in the environment, while mobile sensors will be dispatched to certain locations to adjust the network topology or to conduct advanced data collections at event locations. In this section, we present three such works. First is topology adjustment by mobile sensors, where static sensors may form several isolated groups and mobile sensors are dispatched to connect these groups. Second is sensor navigation, where static sensors that detect events will invite and navigate nearby mobile sensors to their locations to conduct more in-depth analysis of events. Finally, given the locations of events and mobile sensors, the load-balanced dispatch scheme considers how to efficiently dispatch mobile sensors to visit events so that the system lifetime can be extended.

4.1. Topology Adjustment

The work in [15] considers that randomly deployed static sensors may not be distributed evenly so that these static sensors could form several isolated groups, each called an island. These islands cannot connect with each other so that we have to dispatch mobile sensors to connect them, as shown in Fig. 8. Thus, a three-step algorithm is proposed in [15] to dispatch mobile sensors to “repair” the partitioned network. The first step is to search isolated islands by grouping static sensors. This can be done by checking the neighboring relationships of static sensors. All directly and indirectly connected static sensors belong to the same group. The second step is to calculate the least number of mobile sensors $M_{A,B}$ needed to connect two islands $A$ and $B$:

$$M_{A,B} = \left\lceil \frac{\min_{s \in A, t \in B} \{\text{distance}(s,t)\}}{r_c} - 1 \right\rceil,$$

where $d_{A,B} = \min_{s \in A, t \in B} \{\text{distance}(s,t)\}$ is the shortest distance between two islands $A$ and $B$. The last step adopts a dynamic programming scheme to find the optimal set of islands to be connected. Specifically, let $C_G$ be the coverage of an island $G$ and $W(G,m)$ be the optimal island set that starts from island $G$, using $m$ mobile sensors to connect with. Then, we can obtain that

$$W(G,m) = \max\{C_{G,H} + W(G \cup H, m - M_{G,H})\},$$

where $H$ is an island to be connected with $G$. The above equation can be divided down and thus solved by the dynamic programming scheme. Note that for each island $G$, if the remaining $x$ mobile sensors cannot allow to connect with any other island, we set $W(G,x) = 0$. After identifying the optimal set of islands, mobile sensors can be placed along the lines that connect these islands and the concept of virtual force is adopted to make these mobile sensors to achieve maximum coverage. Fig. 8 gives an example.

4.2. Sensor Navigation

Reference [16] considers a hybrid WSN deploying in a region with obstacles. Static sensors are used to monitor the environment. When detecting an event, static sensors will
search nearby mobile sensors to move to their locations to conduct more in-depth analysis of events. Since static sensors have no idea of where mobile sensor locate, they will broadcast to the network to search mobile sensors. In particular, static sensors detecting the same event will elect a leader to broadcast a weight request (WREQ) packet to find mobile sensors. On receiving such a WREQ packet, a mobile sensor $m_j$ will reply its weight $w_j$ to the querying static sensor $s_i$. The weight is calculated as follows:

$$w_j = \frac{A_{m_j} \times d(s_i, m_j)}{E_{m_j}},$$

where $A_{m_j}$ is the size of coverage hole when $m_j$ leaves its current location, $d(s_i, m_j)$ is the shortest distance between $s_i$ and $m_j$, and $E_{m_j}$ is the remaining energy of $m_j$. The size $A_{m_j}$ can be measured by the area of Voronoi polygon of $m_j$. The distance $d(s_i, m_j)$ is represented by the minimum hop count between $s_i$ and $m_j$. After collecting multiple weight replies from mobile sensors, the leader static sensor will choose the mobile sensor with the smallest weight. In the case that two or more mobile sensors have the same weight, the mobile sensor that is closest to the event region is selected.

Suppose that the leader static sensor $s_i$ chooses the mobile sensor $m_j$. It will broadcast an advertisement (ADV) packet to build up a navigation field to guide $m_j$, as shown in Fig. 9(a). In particular, $s_i$ sets the highest credit value $C_1$ for itself and inserts $C_1$ into the ADV packet. Other static sensors receiving such an ADV packet will set their credit values as $C_2$, where $C_2 < C_1$. Then, only those static sensors that have ever relayed the weight reply message from $m_j$ will rebroadcast the ADV packet containing $C_2$. Other static sensors receiving such an ADV packet will set their credit values as $C_3$, where $C_3 < C_2$, and rebroadcast ADV packets if necessary. This procedure is repeated until the ADV packet reaches $m_j$. Then, the mobile sensor $m_j$ will move toward the static sensor with higher credit values. Fig. 9(b) shows an example. With credit values, the mobile sensor can eventually arrive to the location of querying static sensor.

### 4.3. Load-Balanced Dispatch

In [17], Wang et al. address how to efficiently dispatch mobile sensors to visit event locations, given different sets of event locations round by round and limited energy of mobile sensors. The goal is to maximize the system lifetime, which is defined as the number of rounds until some event locations cannot be reached by any mobile sensor due to lack of
energy. On intuitive solution is to minimize the total moving energy of mobile sensors in each round. However, it fails to achieve the goal because mobile sensors are assigned with unfair loads. Some mobile sensors may die fast and other remaining ones will be burdened with heavy loads. Therefore, Wang et al. propose a dispatch solution by balancing the loads of mobile sensors. In [17], a general dispatch problem is considered, where the numbers of mobile sensors and event locations can be arbitrary.

Given a set of event locations $L = \{l_1, l_2, \ldots, l_m\}$ in each round and a set of mobile sensors $S = \{s_1, s_2, \ldots, s_n\}$, where $|L| \leq |S|$, we can transform the dispatch problem to the problem of finding a maximum matching. Specifically, we construct a weighted bipartite graph $G = (S \cup L, S \times L)$, where the weight $w(s_i, l_j)$ of each edge $(s_i, l_j) \in S \times L$ is the energy cost for the mobile sensor $s_i \in S$ to move to the event location $l_j \in L$. The goal is to find a maximum matching $M$ on $G$ such that the total edge weight in $M$ is minimized and the standard deviation of edge weights in $M$ is minimized. Clearly, the first objective means that the total moving energy of mobile sensors should be minimized, while the second objective wants to balance the loads of mobile sensors. Finding such a maximum matching $M$ involves the following steps:

1. For each $s_i \in S$, we associate it with a preference list $Plist(s_i)$, which ranks each location $l_j \in L$ by its weight $w(s_i, l_j)$ in an increasing order. Similarly, for each $l_j \in L$, we associate it with a preference list $Plist(l_j)$, which ranks each $s_i \in S$ by its weight $w(s_i, l_j)$ in an increasing order.

2. To satisfy the second objective, we use a bound $B_{l_j}$ for each $l_j \in L$ to restrict the candidate mobile sensors that $l_j$ can match with. That is, $l_j$ can choose a mobile...
sensor \( s_i \) only if \( w(s_i, l_j) \leq B_{l_j} \). The initial value of each \( B_{l_j} \) is set as

\[
\frac{1}{m} \sum_{j=1}^{m} \min_{\forall i, (s_i, l_j) \in S \times L} \{w(s_i, l_j)\}.
\]

3. For each unmatched location \( l_j \in L \), we find a mobile sensor \( s_i \) in \( Plist(l_j) \) such that \( w(s_i, l_j) \) is minimized and \( w(s_i, l_j) \leq B_{l_j} \) to match with. If \( s_i \) is also unmatched, we add the pair \((s_i, l_j)\) in \( M \). Otherwise, \( s_i \) must be matched with another location, say \( l_o \). In this case, \( l_j \) will compete with \( l_o \) for \( s_i \). In particular, \( l_j \) can win if one of the following cases is satisfied:

a) \( B_{l_j} > B_{l_o} \). Since enlarging the bound will increase the risk of including an edge with an extreme weight into \( M \), we prefer matching \( s_i \) with \( l_j \) to expanding the larger bound \( B_{l_j} \).

b) \( B_{l_j} = B_{l_o} \) and \( l_j \) is prior to \( l_o \) in \( Plist(s_i) \). As \( s_i \) prefers \( l_j \), we match \( s_i \) with \( l_j \) to reduce the total edge weight of \( M \).

c) \( B_{l_j} = B_{l_o} \) and \( s_i \) is the last candidate in \( Plist(l_j) \) but not in \( Plist(l_o) \). Since \( l_j \) cannot have another candidate to pick in \( Plist(l_j) \), \( s_i \) should be matched with \( l_j \).

Once \( s_i \) is matched with \( l_j \), the pair \((s_i, l_o)\) should be replaced by the new pair \((s_i, l_j)\) in \( M \), and \( l_o \) becomes unmatched. If \( l_j \) fails to find a match, it examines the remaining candidates in \( Plist(l_j) \) by the above rule, until there is no candidate.

4. If \( l_j \) cannot match with any mobile sensor in step 3, we increase the bound \( B_{l_j} \) by an amount of \( \Delta_B \) and go back to step 3, until a match can be found for \( l_j \), where

\[
\Delta_B = \frac{\delta}{mn} \times \left( \sum_{j=1}^{m} \max_{\forall i, (s_i, l_j) \in S \times L} \{w(s_i, l_j)\} - \sum_{j=1}^{m} \min_{\forall i, (s_i, l_j) \in S \times L} \{w(s_i, l_j)\} \right),
\]

and \( \delta \) is the system parameter.

5. Repeat steps 3 and 4, until every \( l_j \in L \) can find a mobile sensor to match with.

Fig. 10 gives an example, where \( \delta = 2 \). The initial bound is \( \frac{1}{4} \times (99 + 78 + 93) = 90 \) and \( \Delta_B = \frac{2}{3 \times 4} \times ((215 + 240 + 235) - (99 + 78 + 93)) = 70 \). We start with \( l_1 \). Since there is no candidate in \( Plist(l_1) \) with the initial bound \( B_{l_1} \), we expand \( B_{l_1} = 90 + 70 = 160 \). Thus, we have three candidates \( s_1, s_2, \) and \( s_3 \). Since \( s_1 \) is the first candidate in \( Plist(l_1) \) and \( s_1 \) is unmatched, we add \((s_1, l_1)\) to the matching \( M \). Following the same operation, pair \((s_3, l_2)\) is added in \( M \), as shown in Fig. 10(b). However, after expanding \( B_{l_3} \), \( l_3 \) finds that the first candidate \( s_1 \) has been matched with \( l_1 \), so it competes with \( l_1 \) for \( s_1 \). Since \( B_{l_3} = B_{l_1} \) and \( l_3 \) is prior to \( l_1 \) in \( Plist(s_1) \), \((s_1, l_1)\) is replaced by \((s_1, l_3)\) in Fig. 10(c). Similarly, \( l_1 \) obtains \( s_3 \) from \( l_2 \) and thus \( l_2 \) has to find an unmatched mobile sensor \( s_4 \) to match with. Fig. 10(e) shows the final result.

When \(|S| < |L|\), we can group event locations into \( n \) clusters \( \hat{C} = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_n\} \) and then apply the aforementioned matching solution. In particular, we construct a weighted
An example to show how to find the maximum matching $M$.

bipartite graph $G' = (S \cup \hat{C}, S \times \hat{C})$, where the weight of each edge $(s_i, \hat{c}_j) \in S \times \hat{C}$ is formulated as

$$w(s_i, \hat{c}_j) = \Delta_{\text{move}} \times (d(s_i, l_k) + \phi(\hat{c}_j)),$$

where $S \cup \hat{C}$, $d(s_i, l_k) \in \hat{C}$ is the closest location to $s_i$, and $\phi(\hat{c}_j)$ is the total moving distance to travel all locations inside $\hat{c}_j$ (this can be also treated as the cost of $\hat{c}_j$). To reduce the computation complexity, $\phi(\hat{c}_j)$ is approximated by the total edge weights of the minimum spanning tree in $\hat{c}_j$. By applying the previous matching solution, each mobile sensor $s_i$ can be assigned with a cluster $\hat{c}_j$ of event locations. Then, $s_i$ will first move to the closest location in $\hat{c}_j$ and adopt the traveling salesman approximation algorithm [18] to visit other locations in $\hat{c}_j$. The remaining problem is how to group event locations into clusters. To achieve this, [17] proposes a clustering scheme, as shown in Fig. 11. The goal is to minimize the total cost of clusters. Specifically, we first adopt K-means [19] to group event locations into $n$ clusters. Then, we find the maximum edge weight $w_{\text{intra}}^{\text{max}}$ among edges in all clusters and the minimum inter-cluster distance $w_{\text{inter}}^{\text{min}}$, where the distance between two clusters $\hat{c}_a$ and $\hat{c}_b$ is the distance of the two closest locations $l_i \in \hat{c}_a$ and $l_j \in \hat{c}_b$. If $w_{\text{intra}}^{\text{max}} > w_{\text{inter}}^{\text{min}}$, we can split the cluster with the edge whose weight is $w_{\text{intra}}^{\text{max}}$ (by removing that edge) and then merge two clusters.
Intentional Mobility in Wireless Sensor Networks

whose distance is $w_{\text{inter}}^{\text{min}}$ (by connecting them with the shortest edge). We can repeat the above procedure until $w_{\text{intra}}^{\text{max}} \leq w_{\text{inter}}^{\text{min}}$. Fig. 11 gives an example. In Fig. 11(b), $w_{\text{intra}}^{\text{max}}$ is 60 (in cluster A) and $w_{\text{inter}}^{\text{min}}$ is 16 (between clusters C and D). We thus split cluster A into two clusters $A_1$ and $A_2$, and then merge clusters C and D into the same one, as shown in Fig. 11(c). Similarly, we can further split cluster D and then merge clusters $A_2$ and B to reduce the total cost of clusters. The final result is shown in Fig. 11(d).

5. Design of Mobile Platforms and Applications

Till now, we focus on the algorithmic designs of mobile WSNs. Several studies have implemented mobile platforms to provide mobile capability for sensors. They combine their mobile sensors and a static WSN to develop many interesting applications. In this section, we introduce four such implementations and corresponding applications. First is the pursuer-evader game, where a mobile sensor will capture a moving object by the assistance of a static WSN. Second, we introduce Mobile Emulab, a robotic wireless and sensor network testbed. Third, we present the design of a signal-based mobile sensor, where the mobile sensor can move to the target locations according to the received signal strengths from static sensors. Finally, we introduce an integrated mobile surveillance and wireless sensor
(iMouse) system, which combines video-based surveillance systems and mobile WSNs to provide security surveillance in a home/office environment.

5.1. Pursuer-Evader Game

In [20], Sharp et al. implement the pursuer-evader game [21] by using a hybrid WSN. In such a game, there are two mobile sensors, called pursuer and evader, in the sensing field. The evader will arbitrarily roam inside the sensing field, and the pursuer attempts to intercept the evader according to the information reported by static sensors. The critical issue is how to fast tell the pursuer where the evader is. To achieve this, static sensors that detect the evader will form a group and elect a leader to report where the evader currently locates. Such a report is sent to the moving pursuer via a multi-hop tree routing. After data filtering and interception planning, the pursuer can chase the evader.

In [20], the pursuer and evader are implemented by ground robots, which are essentially mobile off-road laptops equipped with GPS (global position system) devices. Each robot executes the Linux operating system on a 266 MHz Pentium2 CPU with 128 MB of RAM, and is equipped with an IEEE 802.11 wireless radio, all-terrain off-road tires, a motor-controller subsystem, and a high-precision differential GPS device. The GPS device reports the current position of the robot every 0.1 seconds, with an accuracy of approximately 0.02 meters. The maximum speed of the robot is approximate 0.5 meters per second. In addition, a static sensor network is deployed in the sensing field to detect where the evader is. Each static sensor is a MICA2 mote [22] and run on the TinyOS operating system [23]. More implementation details can be found in [20].

5.2. Mobile Emulab

Mobile Emulab [24], a robotic testbed for wireless and sensor networks, is proposed for researchers to evaluate their proposed mobility-related network protocols, applications, and systems. In this testbed, robots carry single-board computers and sensors (so that they can be treated as “mobile sensors”) through a fixed indoor field, all running the user’s selected software. In real-time, interactively or driven by a script, remote users can position and control these robots. Mobile Emulab is composed of three components, including video cameras, robots (or mobile sensors), and static sensors. The cameras are mounted on the ceiling to overlook the sensing field and to track robots. The mobile sensors, which use Acroname Garcia robots [25] as their mobile platforms, carry an XScale-based Stargate [26] small computer running the Linux operating system, a MICA2 mote, and an IEEE 802.11b WLAN card for computing, communicating, and controlling purposes. Finally, the static sensors are used to detect events in the sensing field.

The software architecture of Mobile Emulab is illustrated in Fig. 12. It consists of three subsystems: robot backend, robotd, and visiond. Users can control the robots by sending motion requests or query sensing data from them by sending event requests through the robot backend subsystem. When a motion request is received, the robotd subsystem will translate it into low-level motion commands to control the robots. However, it requires the position data of robots to conduct such translation. Thus, the visiond subsystem will periodically track robots’ positions through the cameras and feedback this information to the robotd subsystem. More implementation details can be refer to [24].
5.3. Signal-Based Mobile Sensors

The above systems require GPS devices or ceiling-mounted cameras to obtain positions of mobile sensors. Reference [27] releases such limitations and uses received signal strength (RSS) from static sensors to navigate mobile sensors. In particular, given a static WSN, the sink first constructs a routing path to each static sensor by flooding a message to the WSN. These routing paths are used as navigation paths of mobile sensors. Specifically, when a mobile sensor is dispatched to visit a static sensor $s$, the sensor $s$ will send a packet containing its ID to the sink. Static sensors on the routing path will also add their IDs to the packet. In this way, the inverse sequence of sensors’ IDs will be the navigation path.

After receiving the packet replied from sensor $s$, the sink will dispatch a mobile sensor to the target location. The mobile sensor will move to the next static sensor by continuously monitoring the signal strength of the beacons sent from the static sensor at the next hop. To approach a specified static sensor, the mobile sensor will go forward and detect the change of RSS. If RSS increases, it means that the mobile sensor is approaching a turning point, which is defined as the midpoint of a line segment that the mobile sensor can receive the strongest signal strength, as shown in Fig. 13(a). Otherwise, the mobile sensor is moving away from the target location and thus it will immediately reverse its current moving direction. When the mobile sensor arrives at the turning point, it knows that the target location is either in the right or left side of its moving direction. In this case, the mobile sensor will first turn right to seek for the target. However, if the target location is in the left side, the signal strength received by the mobile sensor will become weaker as it moves away from the target location. In this case, the mobile sensor will reverse its moving direction. By repeating the above procedure, the mobile sensor can eventually arrive to the target location. Fig. 13(b) gives an example, where the mobile sensors takes totally 15 steps to arrive to the target location.
22 You-Chiun Wang and Yu-Chee Tseng

(b) moving steps of the mobile sensor

Figure 13. Navigation of a mobile sensor by the received signal strength.

5.4. iMouse System

Traditional video-based surveillance systems typically collect a large volume of image data from wall-board cameras, which require huge computation or even manpower to analyze. Introducing the intelligence of mobile WSNs can help reduce such overheads and provide more advanced, context-rich services. Motivated by this observation, iMouse [28] is proposed to integrate the sensing capability of mobile WSNs into surveillance systems. The iMouse system consists of many static sensors and a small number of more powerful mobile sensors, as shown in Fig. 14. The former is used to monitor the environment, while the latter can move to the potential emergency sites (reported by static sensors) and take snapshots at event locations. In this way, the iMouse system can be event-driven, in the sense that only when an event occurs should a mobile sensor be dispatched to capture the images of that event. Thus, iMouse can avoid recording unnecessary images when nothing happens.

In iMouse, each mobile sensor has the following functionalities: moving to event locations, exchanging messages with other sensors, taking snapshots at event locations, and transmitting images to the server. To achieve this, each mobile sensor is equipped with a Stargate processing board, which is connected to a LEGO car [29], a MICAz Mote [22], a webcam, and an IEEE 802.11 WLAN card, as shown in Fig. 15(a). The LEGO car supports mobility. The Mote can communicate with static sensors. The webcam is used to take snapshots. The WLAN card is to support high-speed, long-distance communications, such as transmitting images. The Stargate controls the actions of the LEGO car and the webcam. The light detector below the LEGO car is used to navigate the mobile sensor. This is realized by different colors of tapes stuck on the ground, as shown in Fig. 15(b). Specifically, black tapes represent roads and golden tapes represent intersections. The mobile sensor will move along the black tapes and can make a 90-degree turn when it arrives at an intersection.
(1) send commands to WSN
(2) report events
(3) dispatch
(4) move to event locations and take snapshots
(5) report images through WLAN

Figure 14. System Architecture of iMouse.

(a) mobile sensor  (b) a grid-like sensing field

Figure 15. Implementation of the iMouse system.

6. Conclusions

Traditional static WSNs have limitations on supporting multiple missions and handling different situations when network conditions change. Introducing mobility to WSNs can significantly improve the network capability and thus release the above limitations. This chapter provides a comprehensive survey of current researches on intentional mobility in WSNs. Various moving strategies of mobile sensors for network deployment, functional enhancement, and sensor dispatch have been discussed. Also, the design of mobile platforms and potential applications of mobile WSNs have been presented in this chapter.
References


