Intentional Mobility in Wireless Sensor Networks

You-Chiun Wang and Yu-Chee Tseng

Abstract—The emerging wireless sensor networks (WSNs) offer a convenient way to monitor physical environments. In the past, WSNs are deployed with 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 of mobile WSNs.

Index Terms—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 way to monitor physical environments. In recent years, a large amount of WSN applications such as object tracking, health monitoring, security surveillance, and intelligent transportation have been proposed.

A WSN is usually deployed with static nodes to perform monitoring missions in the region of interest. However, due to the dynamic changes of events and hostile environment, a purely static WSN could face the following 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 large amount of nodes. Moreover, the dynamic change of environment and the existence of obstacles could complicate the problem.
- Sensor nodes are usually battery-powered and prone to errors. As some nodes die due to the exhaustion of their energy, there could be 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.
- 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 target's track, while in a boundary detection mission, there should be enough 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 missions' requirements could not be economically feasible.
- Some applications may need sophisticated (and thus expensive) sensors to involve in. For example, 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 such as cameras should be used to obtain more information. Nevertheless, it is infeasible to equip camera on each node because of their large number.

By introducing mobility to 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, so that the WSN can cover the most area of the sensing filed.
- **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 several designs of mobile platforms to support

The authors are with the Department of Computer Science, National Chiao-Tung University, Hsin-Chu, 30010, Taiwan.
E-mail: {wangyc, yctseng}@cs.nctu.edu.tw

mobility for sensor nodes, and give some interesting applications of mobile WSNs.

2 Mobility-Assisted Network Deployment

Sensor deployment is one important issue of WSNs because it directly affects the detection capability of the network. Several researches have proposed different mobility-assisted deployment schemes to help organize 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 covers the maximal target area. In this section, three categories of mobility-assisted deployment schemes are introduced. The *force-directed deployment schemes* image that there are 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 $\overrightarrow{F_{iA}}$ by the target area, the repulsive (negative) force $\overrightarrow{F_{iR}}$ by all obstacles, and the force $\overrightarrow{F_{ij}}$ between sensors s_i and s_j. Therefore, the total force $\overrightarrow{F_i}$ on sensor s_i can be expressed as $\overrightarrow{F_i} = \overrightarrow{F_{iA}} + \overrightarrow{F_{iR}} + \sum_{j=1,j\neq i}^{n} \overrightarrow{F_{ij}}$, where n is the number of sensors. The force $\overrightarrow{F_{ij}}$ is expressed by a polar coordinate notation (r, θ), where r is the magnitude and θ is the orientation (angle) of the vector $\overrightarrow{F_{ij}}$. Thus, $\overrightarrow{F_{ij}}$ can be derived as

$$\overrightarrow{F_{ij}} = \begin{cases} (w_A \cdot (d_{ij} - d_{\text{th}}), \theta_{ij}) & \text{if } d_{ij} > d_{\text{th}} \\ (w_R \cdot \frac{1}{d_{ij}}, \pi + \theta_{ij}) & \text{if } d_{ij} < d_{\text{th}} \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where d_{ij} is the Euclidean distance between sensors s_i and s_j, d_{th} is a threshold distance, θ_{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 $d_{\text{th}} = d_{12}$. By Eq. (1), s_2 exerts no force on s_1, s_3 exerts an attractive force $\overrightarrow{F_{13}}$ on s_1, and s_4 exerts a repulsive force $\overrightarrow{F_{14}}$ 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 $\overrightarrow{F_1}$.

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.

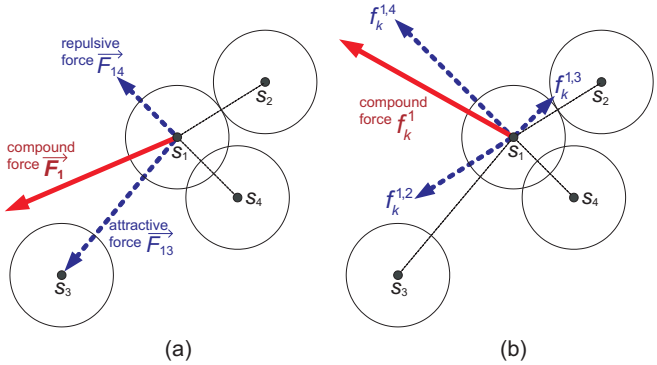


Fig. 1: An example of the virtual forces on sensor s_1 by three other sensors: (a) attractive and repulsive forces in [4] and (b) only repulsive forces in [5].

2. $f(0^+) = f_{\max}$, where $f_{\max}$ 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 s_i by another sensor s_j in its communication range is calculated to be a repulsive force as $f_k^{i,j} = \frac{D_k^i}{\mu^2}(r_c - |p_k^i - p_k^j|)\frac{p_k^j - p_k^i}{|p_k^j - p_k^i|}$, where D_k^i is the local density of sensor s_i at step k, μ 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 s_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_k^1. Note that when a sensor moves less than a threshold distance after a predefined period, it is considered to reach the stable status and thus the sensor 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 its Voronoi polygon, there could be coverage hole inside that polygon.

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

Vector-based (VEC) strategy: The VEC strategy adopts the idea of 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_{avg} is the average distance between two sensors when all 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 $d_{\text{avg}} - d_{ij}$.

Voronoi-based (VOR) strategy: When a sensor detects the existence of a coverage hole, it will move toward its

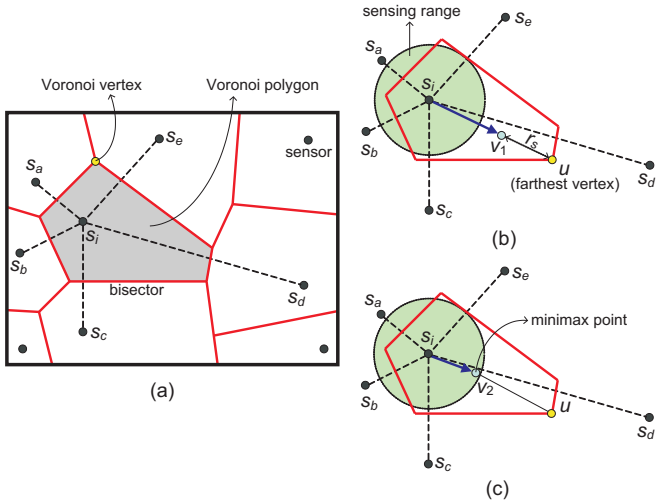


Fig. 2: The deployment method proposed in [6]: (a) a Voronoi diagram, (b) the VOR strategy, and (c) the minimax strategy.

farthest Voronoi vertex. Fig. 2(b) gives an example, where point u is the farthest Voronoi vertex. Sensor s_i will move along $\overline{s_i u}$ and stop at point v_1, where $|\overline{uv_1}|$ 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 $\frac{r_c}{2}$ to prevent the sensor from moving more than needed.

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 is minimized, as shown in Fig. 2(c). 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 two deployment-related problems: *sensor placement* and *sensor dispatch*. The placement problem determines how to place the least number of sensors in a sensing field to guarantee complete coverage and network connectivity. The dispatch problem asks how to delegate mobile sensors to the target locations calculated by 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 $\sqrt{3}r_{\min}$, where $r_{\min} = \min\{r_c, r_s\}$, so one row of sensors are enough to cover the whole region while maintaining network connectivity. A multi-row region requires several rows of sensors to cover it. To partition a sensing 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 $\sqrt{3}r_{\min}$, as shown in Fig. 3(a). When the expanded line cuts off an obstacle with an area,

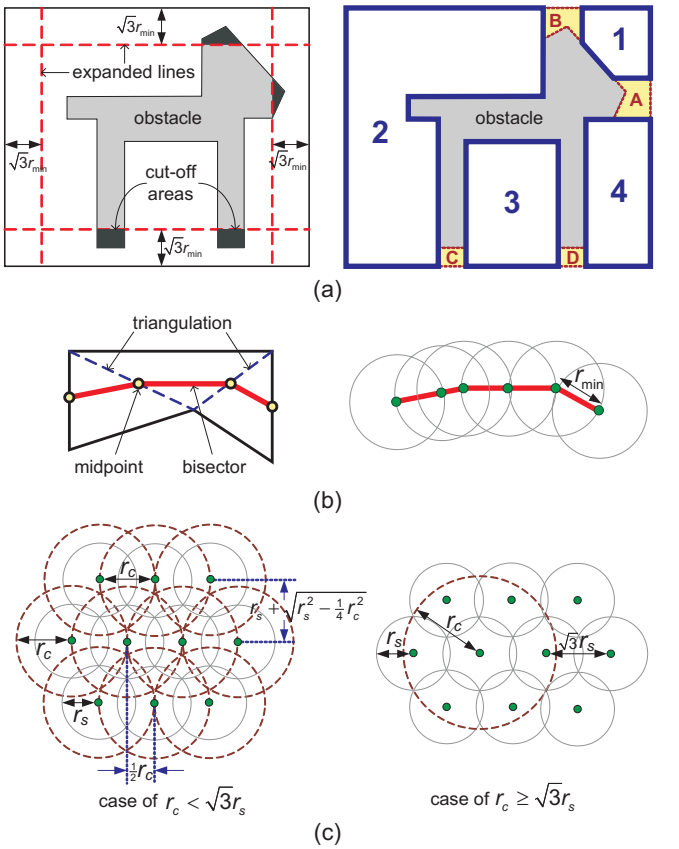


Fig. 3: The sensor placement method proposed in [8]: (a) partition a sensing field, (b) place sensors in a single-row region, and (c) place sensors in a multi-row region.

say $\mathcal{O}$, we can take a project from $\mathcal{O}$ 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: We find the region's bisector and place a sequence of sensors, each separated by a distance of $r_{\min}$, along the bisector, as shown in Fig. 3(b). This can guarantee both coverage and connectivity. 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 of $r_s + \sqrt{r_s^2 - \frac{1}{4}r_c^2}$ and shifted by a distance of $\frac{1}{2}r_c$. With such an arrangement, the coverage of the whole area is guaranteed. In this case, we have to add a column of sensors between two adjacent rows, each 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 guaranteed. 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 $r_c < \sqrt{3}r_s$ and $r_c \geq \sqrt{3}r_s$, respectively.

For sensor dispatch, [8] proposes a centralized scheme by converting the dispatch problem to a *maximum-weight maximum-matching problem*:

1. Given a set of mobile sensors $\mathcal{S}$ and a set of locations $\mathcal{L} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$ computed by the placement scheme, where $|\mathcal{S}| \geq |\mathcal{L}|$, we construct a weighted complete bipartite graph $\mathcal{G} = (\mathcal{S} \cup \mathcal{L}, \mathcal{S} \times \mathcal{L})$. The weight of each edge $(s_i, (x_j, y_j))$, $s_i \in \mathcal{S}$, $(x_j, y_j) \in \mathcal{L}$, 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 after they move to target locations, where e_i is the energy of a sensor s_i, Δ_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 $\mathcal{G}$. In particular, we construct a new graph $\mathcal{G}' = (\mathcal{S} \cup \mathcal{L} \cup \mathcal{L}', \mathcal{S} \times \{\mathcal{L} \cup \mathcal{L}'\})$ from $\mathcal{G}$, where $\mathcal{L}'$ is a set of $|\mathcal{S}| - |\mathcal{L}|$ elements, each called a *virtual location*. The weight of each edge in $\mathcal{G}'$ that also appears in $\mathcal{G}$ remains the same as that in $\mathcal{G}$, and the weight of each edge from $s_i \in \mathcal{S}$ to $(x_j, y_j) \in \mathcal{L}'$ is set to $w_{\min}$, where $w_{\min} = \min_{s_i \in \mathcal{S},\ (x_j, y_j) \in \mathcal{L}} \{w(s_i, (x_j, y_j))\} - 1$. Intuitively, a virtual location is a dummy one. Its purpose is to make the sizes of sets $\mathcal{S}$ and $\{\mathcal{L} \cup \mathcal{L}'\}$ become equal. This allows us to transform the problem to the *maximum-weight perfect-matching problem* on $\mathcal{G}'$, whose purpose is to find a perfect matching $\mathcal{M}$ on $\mathcal{G}'$ with the maximum edge weights. Then, we can adopt the Hungarian method [9] to find $\mathcal{M}$.
3. For each edge $(s_i, (x_j, y_j))$ in $\mathcal{M}$ such that $(x_j, y_j) \in \mathcal{L}$, we move sensor s_i to location (x_j, y_j) through the shortest path. However, if there is any edge $(s_i, (x_j, y_j)) \in \mathcal{M}$ such that $(x_j, y_j) \in \mathcal{L}$ 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 $\mathcal{L}$, 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 $\mathcal{A}$'s boundaries. This can be done by expanding the obstacles' perimeters outward and $\mathcal{A}$'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 $\mathcal{H} = (s_i \cup (x_j, y_j) \cup \mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ contains all vertices v of the polygons representing the expanded areas of obstacles and $\mathcal{A}$'s boundaries such that v is not inside other expanded areas, and $\mathcal{E}$ contains all edges (u, v) such that $u, v \in \{s_i \cup (x_j, y_j) \cup \mathcal{V}\}$ and $\overline{uv}$ does not pass any expanded area of obstacles or $\mathcal{A}$'s boundaries. The weight of each edge $(u, v) \in \mathcal{E}$ is the length of $\overline{uv}$. Fig. 4 gives an example. Nodes e and f are not vertices of $\mathcal{H}$ because they are inside obstacles 2's and 3's expanded areas, respectively. Edges (a, b), (a, g), (h, b), and $(h, g) \in \mathcal{E}$, but (h, c) and $(h, d) \notin \mathcal{E}$ because they pass the expanded area of obstacle 2. After constructing such a graph $\mathcal{H}$, we can use the Dijkstra's algorithm [10] to calculate the shortest path from s_i to (x_j, y_j).

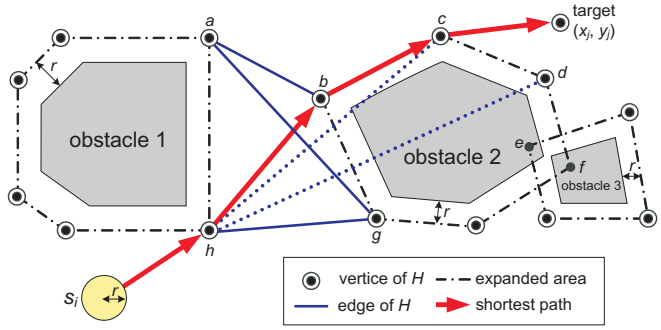


Fig. 4: Find a collision-free path from s_i to (x_j, y_j).

3 Functional Enhancements by Mobile Sensors

Mobile sensors can be used to enhance the functionalities of a WSN. We can relocate some 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 grids and then moves sensors from high-density grids to low-density grids. This can help redistribute a more even network, especially when some nodes fail. In [11], the *grid-quorum* and *cascaded movement* schemes are proposed to quickly find out 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, each sensor s_i is associated with a *tolerable delay* T_i, during which its successor must

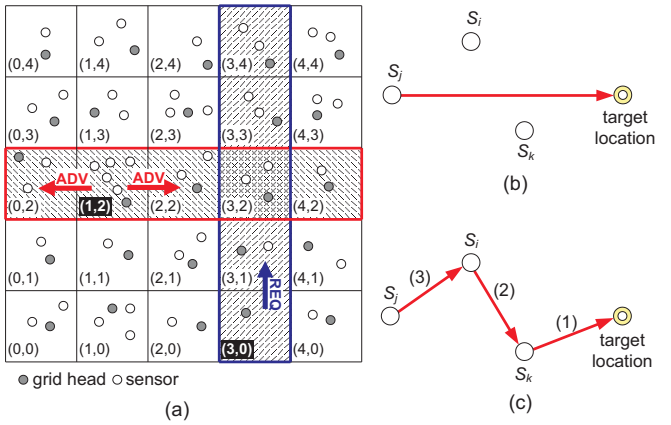


Fig. 5: The coverage enhancement method proposed in [11]: (a) grid-quorum, (b) direct movement, and (c) cascaded movement.

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_i 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 $E_1^{\min}$ and $E_2^{\min}$ as their minimum remaining energy. Schedule 1 will be selected if $E_1 - E_1^{\min} \leq E_2 - E_2^{\min}$.

3.2 Connectivity Enhancement Methods

Reference [12] discusses how to move nodes to reorganize a stronger network. 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 $\mathcal{G}$ that describes the network topology, the biconnected components (also called *blocks*) of $\mathcal{G}$ are first identified along with its cutvertices, and then the graph $\mathcal{G}$ 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 to the nearest node in its parent block, by the distance that one edge appears.
- 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).

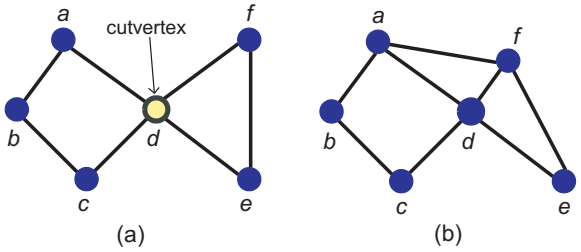


Fig. 6: Achieve a biconnected network by moving node f toward node a: (a) 1-connected network and (b) biconnected network.

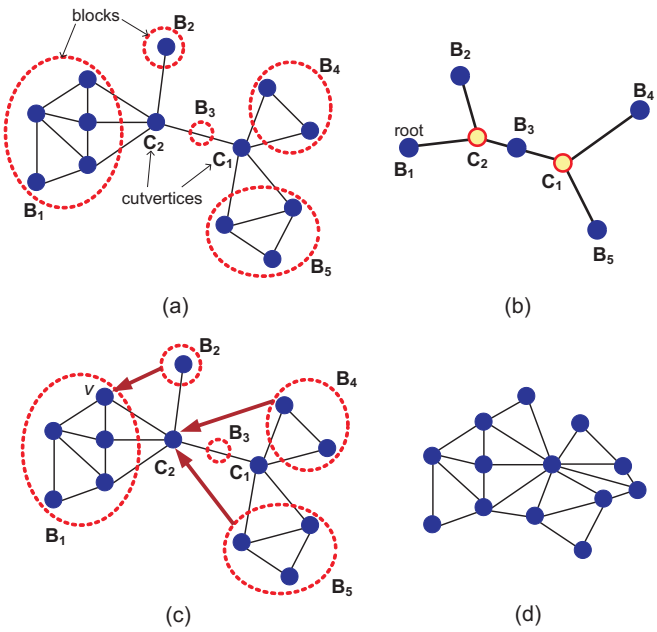


Fig. 7: An example of the block movement algorithm propose in [12]: (a) initial network, (b) block tree, (c) block movement, and (d) final network.

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 $p_i^{k+1} = p_i^k + f(l^{k+1}, p_i^k, p_i^0)$, where l^{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 sensor's 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], two possible functions are suggested to satisfy the above criteria. The first one is $f(d) = de^{-d}$, where $e \approx 2.718$. The other one is $f(d) = \alpha d^{\beta} e^{-\gamma d}$, where $\alpha e^{-\gamma d}(\beta d^{\beta-1} - \gamma d^{\beta}) > 1$, $\forall d$. (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. Increase CDF bins representing positions $\geq l_x^k$.
2. Scale CDF by $\frac{k}{k+1}$.
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 above moving strategies, sensors will move close to event locations. However, to maintain complete coverage of the sensing field, [14] propose using a Voronoi diagram to determine whether the movement of a sensor will cause a coverage hole. Specifically, if a sensor finds that its sensing range can completely cover its Voronoi polygon, it should stop moving.

4 Dispatch of Mobile Sensors in a Hybrid WSN

Some works consider a hybrid WSN consisting of both static and mobile sensors. Static sensors will detect events occurred in the sensing field, while mobile sensors will be dispatched to certain locations to adjust the network topology or to conduct advanced analysis of events. 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 static sensors may form several isolated groups, each called an *island*. These islands cannot communicate 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 $M_{A,B} = \left\lceil \frac{d_{A,B}}{r_c} - 1 \right\rceil$ needed to connect two islands A and B, 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 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 $W(G,m) = \max\{C_{G \cup H} + W(G \cup H, m - M_{G,H})\}$, where H is an island to be connected with G. The above equation can be divided down and thus solved by the dynamic programming. 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 island set, 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.

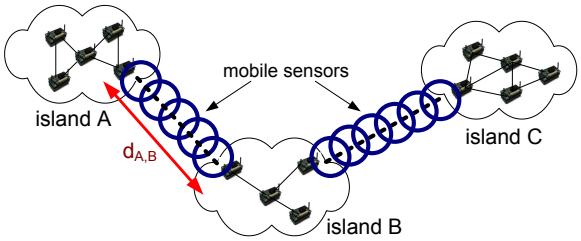


Fig. 8: The isolated islands are connected by mobile sensors.

4.2 Sensor Navigation

Reference [16] considers that a hybrid WSN is deployed 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 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 $w_j = \frac{A_{m_j} \times d(s_i, m_j)}{E_{m_j}}$, 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 weight replies, 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, until it arrives to the event region. Fig. 9(b) shows an example.

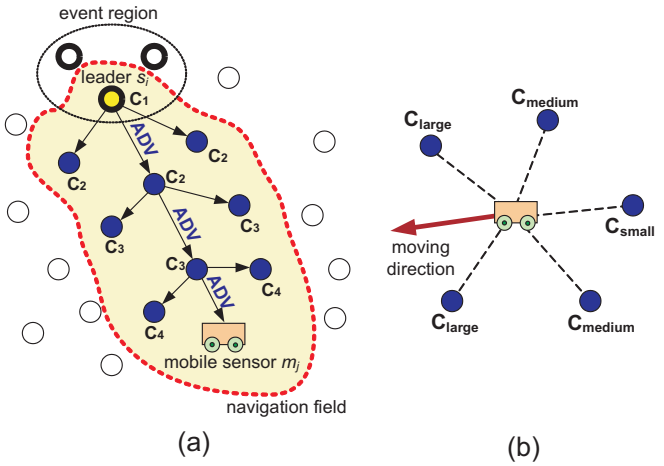


Fig. 9: Navigate a mobile sensor: (a) build up the navigation field and (b) calculate the moving direction of the mobile sensor.

4.3 Load-Balanced Dispatch

Given different sets of event locations round by round, [17] discusses how to efficiently dispatch mobile sensors to visit event locations such that the number of rounds can be maximized. 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 quickly die and other remaining ones will be burdened with heavy loads. Therefore, [17] proposes a dispatch solution by balancing the loads of mobile sensors.

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

1. For each $s_i \in \mathcal{S}$, we associate it with a preference list $Plist(s_i)$, which ranks each $l_j \in \mathcal{L}$ by its weight $w(s_i, l_j)$ in an increasing order. Similarly, for each $l_j \in \mathcal{L}$, we associate it with a preference list $Plist(l_j)$, which ranks each $s_i \in \mathcal{S}$ 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 $l_j \in \mathcal{L}$ 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 $\frac{1}{m}\sum_{j=1}^{m} \min_{\forall i,(s_i,l_j)\in\mathcal{S}\times\mathcal{L}} \{w(s_i, l_j)\}$.
3. For each unmatched location $l_j \in \mathcal{L}$, 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 $\mathcal{M}$. Otherwise, s_i must have been 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 $\mathcal{M}$, we prefer matching s_i with l_j to avoid 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 $\mathcal{M}$.
   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 $\mathcal{M}$, 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 Δ_B and go back to step 3, until a match can be found for l_j.
5. Repeat steps 3 and 4, until every $l_j \in \mathcal{L}$ can find a mobile sensor to match with.

Fig. 10 gives an example, where $\delta = 2$ and $\Delta_B = 70$. The initial bound is $\frac{1}{3} \times (99 + 78 + 93) = 90$. 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 $\mathcal{M}$. Following the same operation, pair (s_3, l_2) is added in $\mathcal{M}$, 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 $|\mathcal{S}| < |\mathcal{L}|$, we can group event locations into n clusters $\hat{\mathcal{C}} = \{\hat{c}_1, \hat{c}_2, \cdots \hat{c}_n\}$ and then apply the aforementioned matching solution. In particular, we construct a weighted bipartite graph $\mathcal{G}' = (\mathcal{S} \cup \hat{\mathcal{C}}, \mathcal{S} \times \hat{\mathcal{C}})$, where the weight of each edge $(s_i, \hat{c}_j) \in \mathcal{S} \times \hat{\mathcal{C}}$ is formulated as $w(s_i, \hat{c}_j) = \Delta_{\text{move}} \times (d(s_i, l_k) + \phi(\hat{c}_j)), s_i \in \mathcal{S}, \hat{c}_j \in \hat{\mathcal{C}}$, where Δ_{move} is the energy cost for a mobile sensor to move in one-unit distance, $l_k \in \hat{c}_j$ 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

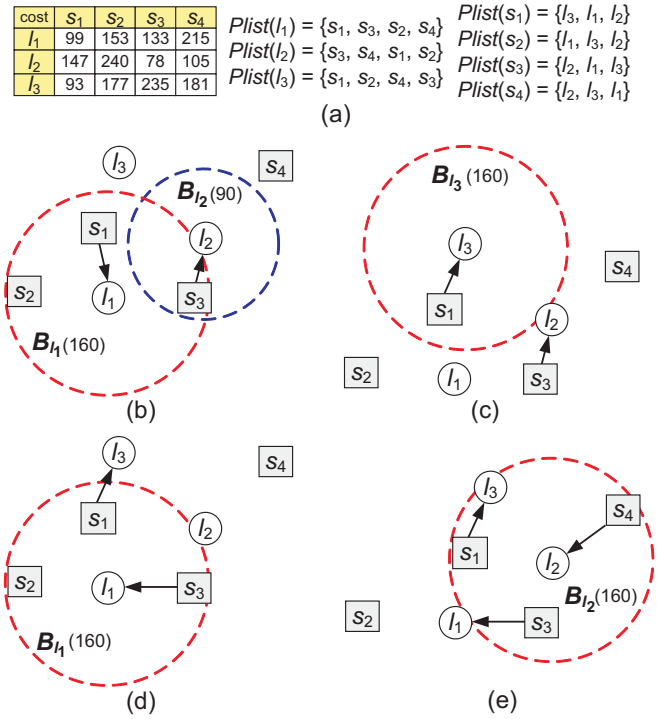


Fig. 10: An example to show how to find the maximum matching $\mathcal{M}$: (a) energy costs and preference lists, (b) $\mathcal{M} = \{(s_1, l_1), (s_3, l_2)\}$, (c) $\mathcal{M} = \{(s_3, l_2), (s_1, l_3)\}$, (d) $\mathcal{M} = \{(s_3, l_1), (s_1, l_3)\}$, and (e) $\mathcal{M} = \{(s_3, l_1), (s_4, l_2), (s_1, l_3)\}$.

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 $w_{\max}^{\text{intra}}$ among edges in all clusters and the minimum inter-cluster distance $w_{\min}^{\text{inter}}$, 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 $w_{\max}^{\text{intra}} > w_{\min}^{\text{inter}}$, we can split the cluster with the edge whose weight is $w_{\max}^{\text{intra}}$ (by removing that edge) and then merge two clusters whose distance is $w_{\min}^{\text{inter}}$ (by connecting them with the shortest edge). We can repeat the above procedure until $w_{\max}^{\text{intra}} \leq w_{\min}^{\text{inter}}$. Fig. 11 gives an example. In Fig. 11(b), $w_{\max}^{\text{intra}}$ is 60 (in cluster A) and $w_{\min}^{\text{inter}}$ 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

Several studies have implemented mobile platforms to support mobility for sensors. 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 the *iMouse system*, which combines video-based

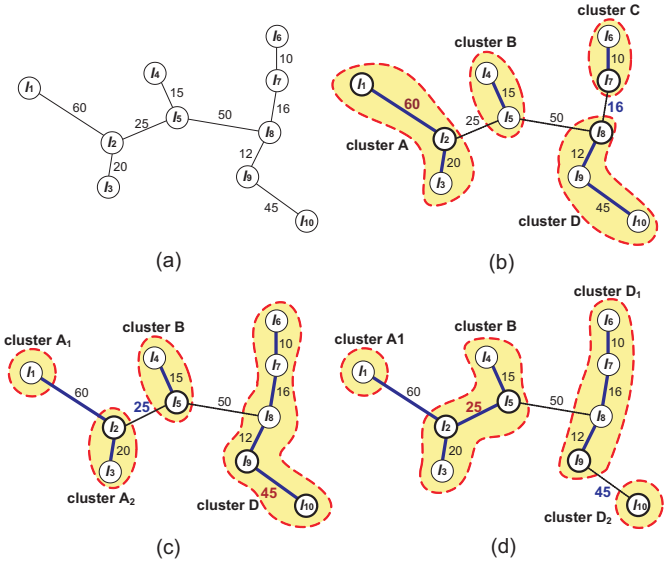


Fig. 11: An example to group event locations into four clusters: (a) initial topology, (b) total cost: 162, (c) total cost: 118, and (d) total cost: 98.

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 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 quickly tell the pursuer where the evader locates. To achieve this, static sensors that detect the evader will form a group and elect a leader to report the current location of evader. 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 a 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 approximate 0.02 meters. The maximum speed of the robot is approximate 0.5 meters per second. In addition, a static WSN is deployed 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 refer to [24].

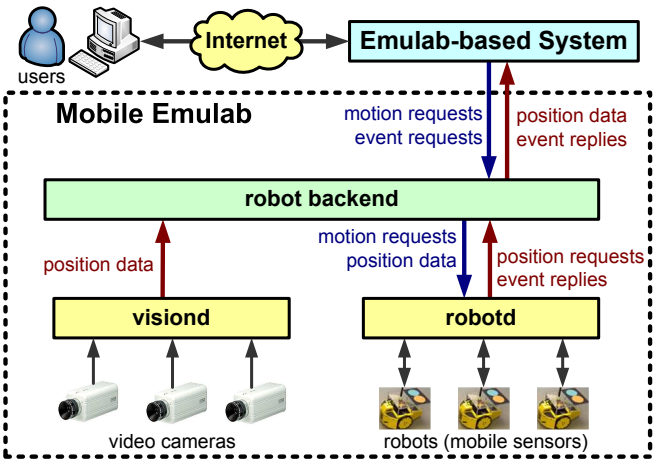


Fig. 12: The software architecture of Mobile Emulab.

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, the static sensor 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 the static sensor, 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.

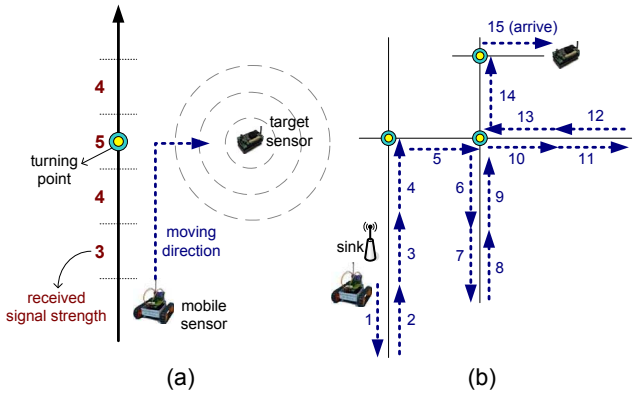


Fig. 13: Navigate a mobile sensor by the received signal strength: (a) find a turning point and (b) the moving steps of the mobile sensor.

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

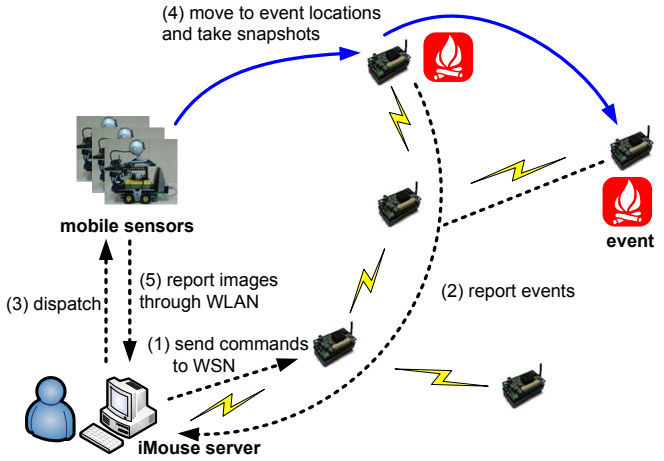
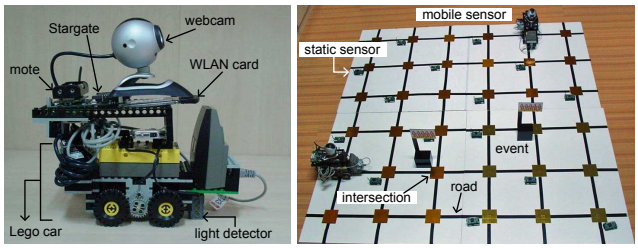


Fig. 14: System Architecture of iMouse.

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.



(a) (b)

Fig. 15: Implementation of the iMouse system: (a) mobile sensor and (b) a grid-like sensing field.

6 Conclusions

Traditional static WSNs have limitations on supporting multiple missions and handling different situations when the network condition changes. 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

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Comm. Magazine*, vol. 40, no. 8, pp. 102–114, 2002.
[2] S.S. Dhillon and K. Chakrabarty, "Sensor placement for effective coverage and surveillance in distributed sensor networks," *proc. IEEE Wireless Comm. and Networking Conf.*, 2003, pp. 1609–1614.
[3] G. Cao, G. Kesidis, T.L. Porta, B. Yao, and S. Phoha, "Purposeful mobility in tactical sensor networks," *Sensor Network Operations*, 2006.
[4] Y. Zou and K. Chakrabarty, "Sensor deployment and target localization based on virtual forces," *proc. IEEE INFOCOM*, 2003, pp. 1293–1303.
[5] N. Heo and P. K. Varshney, "Energy-efficient deployment of intelligent mobile sensor networks," *IEEE Tran. Systems, Man and Cybernetics–Part A: Systems and Humans*, vol. 35, no. 1, pp. 78–92, 2005.
[6] G. Wang, G. Cao, and T.L. Porta, "Movement-assisted sensor deployment," *proc. IEEE INFOCOM*, 2004, pp. 2469–2479.
[7] F. Aurenhammer, "Voronoi diagrams – a survey of a fundamental geometric data structure," *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, 1991.
[8] Y.C. Wang, C.C. Hu, and Y.C. Tseng, "Efficient placement and dispatch of sensors in a wireless sensor network," *IEEE Trans. Mobile Computing*, vol. 7, no. 2, pp. 262–274, 2008.
[9] H.W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, 2001.
[11] G. Wang, G. Cao, T.L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," *proc. IEEE INFOCOM*, 2005, pp. 2302–2312.
[12] P. Basu and J. Redi, "Movement control algorithms for realization of fault-tolerant ad hoc robot networks," *IEEE Network*, vol. 18, no. 4, pp. 36–44, 2004.
[13] R. Diestel, "Graph theory," *Graduate Texts in Mathematics*, 1997.
[14] Z. Butler and D. Rus, "Event-based motion control for mobile-sensor networks," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 34–42, 2003.
[15] S. Zhou, M.Y. Wu, and W. Shu, "Finding optimal placements for mobile sensors: wireless sensor network topology adjustment," *proc. IEEE Circuits and Systems Symp. Emerging Technologies: Frontiers of Mobile and Wireless Comm.*, 2004, pp. 529–532.
[16] A. Verma, H. Sawant, and J. Tan, "Selection and navigation of mobile sensor nodes using a sensor network," *proc. IEEE Int'l Conf. Pervasive Computing and Comm.*, 2005, pp. 41–50.
[17] Y.C. Wang, W.C. Peng, M.H. Chang, and Y.C. Tseng, "Exploring load-balance to dispatch mobile sensors in wireless sensor networks," *proc. IEEE Int'l Conf. Computer Comm. and Networks*, 2007, pp. 669–674.
[18] M. Blaser, "A new approximation algorithm for the asymmetric TSP with triangle inequality," *proc. ACM-SIAM Symp. Discrete Algorithms*, 2003, pp. 638–645.
[19] J. Han and M. Kamber, *Data mining: concepts and techniques*, Academic Press, 2001.
[20] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler, "Design and implementation of a sensor network system for vehicle tracking and autonomous interception," *proc. European Workshop on Sensor Networks*, 2005, pp. 93–107.
[21] M. Demirbas, A. Arora, and M. Gouda, "A pursuer-evader game for sensor networks," *proc. Sixth Symp. Self-Stabilizing Systems*, 2003, pp. 1–16.
[22] Crossbow, "MICA Series," http://www.xbow.com/.
[23] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5, pp. 93–104, 2000.
[24] D. Johnson, T. Stack, R. Fish, D.M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: a robotic wireless and sensor network testbed," *proc. IEEE INFOCOM*, 2006.
[25] Acroname, "Garcia robot," http://www.acroname.com/garcia/garcia.html.
[26] Crossbow, "Stargate Gateway," http://www.xbow.com/.
[27] J.P. Sheu, P.W. Cheng, and K.Y. Hsieh, "Design and implementation of a smart mobile robot," *proc. IEEE Int'l Conf. Wireless And Mobile Computing, Networking And Comm.*, 2005, pp. 422–429.
[28] Y.C. Tseng, Y.C. Wang, K.Y. Cheng, and Y.Y. Hsieh, "iMouse: an integrated mobile surveillance and wireless sensor system," *IEEE Computer*, vol. 40, no. 6, pp. 60–66, 2007.
[29] MINDSTORM, "Robotics Invention System," http://mindstorms.lego.com.