Mobility Management Algorithms and Applications for Mobile Sensor Networks

You-Chiun Wang, Fang-Jing Wu, and Yu-Chee Tseng
Department of Computer Science, National Chiao-Tung University, Hsin-Chu, 30010, Taiwan
Email: \{wangyc, fangjing, yctseng\}@cs.nctu.edu.tw

Abstract

Wireless sensor networks (WSNs) offer a convenient way to monitor physical environments. In the past, WSNs are all considered static to continuously collect information from the environment. Today, by introducing intentional mobility to WSNs, we can further improve the network capability on many aspects, such as automatic node deployment, flexible topology adjustment, and rapid event reaction. In this article, we survey recent progress in mobile WSNs and compare works in this field in terms of their models and mobility management methodologies. The discussion includes three aspects. Firstly, we discuss mobility management of mobile sensors for the purposes of forming a better WSN, enhancing network coverage and connectivity, and relocating some sensors. Secondly, we introduce path-planning methods for data ferries to relay data between isolated sensors and to extend a WSN’s lifetime. Finally, we review some existing platforms and discuss several interesting applications of mobile WSNs.

KEY WORDS: mobility management, path planning, sensor applications, topology adjustment, wireless sensor networks.

1. Introduction

The development of wireless technologies and micro-sensing MEMS has triggered the success of wireless sensor networks (WSNs). A WSN is composed of one or multiple remote sinks and many tiny, low-power sensors, each equipped with actuators, sensing devices, and wireless transceivers [1]. These sensors are massively deployed in a region of interest (ROI) to continuously collect and report surrounding data. WSNs offer a convenient way to monitor physical environments. Many applications such as object tracking, health monitoring, security surveillance, and intelligent transportation [2, 3, 4, 5] have been proposed.

A WSN is usually deployed with static sensors to perform monitoring missions. However, due to the dynamics of events or environments, a purely static WSN could face these challenges: (1) Sensors are often scattered in a ROI by aircrafts or robots [6]. These randomly scattered sensors could not guarantee complete coverage of the ROI and may be partitioned into disconnected subnetworks. The existence of obstacles could even worsen the problem. (2) Sensors are usually powered by batteries. As some sensors exhaust their energy, holes could appear and the network could be broken. However, in many scenarios, it is quite difficult to recharge sensors or redeploy nodes. (3) A WSN may need to support multiple missions or have multiple types of sensors [7]. Sometimes, we may need to send a certain type of sensors to particular locations to support particular needs. Without mobility, this is difficult to achieve. (4) While most efforts assume that sensors are cheap, some types of sensors may be expensive. Dispatching of those expensive ones from locations to locations may be necessary.
By introducing mobility to a WSN, we can enhance its capability to handle the above problems. Nevertheless, mobile WSN and 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 the sense that we can control their movement to achieve our missions. In this article, we give a comprehensive survey of recent progress in mobile WSNs. Our discussion focus on two types of nodes: mobile sensors and data ferries. With the former, one may change the network topology by moving these mobile sensors. With the latter, one may maneuver these data ferries to collect or relay sensing data. We will cover some mobile platforms and applications that we can control their movement to achieve our missions. In this article, we give a comprehensive discussion focus on two types of nodes: mobile WSN and mobile ad hoc network (MANET). Nevertheless, mobile WSN and MANET are essentially different. Mobility in a MANET is often arbitrary, whereas mobility in a mobile WSN should be ‘intentional’, in the sense that we can control their movement to achieve our missions. In this article, we give a comprehensive survey of recent progress in mobile WSNs. Our discussion focus on two types of nodes: mobile sensors and data ferries. With the former, one may change the network topology by moving these mobile sensors. With the latter, one may maneuver these data ferries to collect or relay sensing data. We will cover some mobile platforms and applications that we can control their movement to achieve our missions. In this article, we give a comprehensive discussion focus on two types of nodes: mobile WSN and mobile ad hoc network (MANET).

2. Mobility Management of Mobile Sensors

2.1. Solutions to Deploying Mobile Sensors

Sensor deployment is a basic issue since it decides a WSN’s detection ability. A good deployment should satisfy both coverage and connectivity [8, 9]. Coverage requires that each location in the ROI be monitored by sensors, and connectivity requires that the network remain not partitioned. With mobile sensors, the deployment job becomes ‘automatic’. We introduce three deployment methods: The force-based deployment images that virtual forces will drive sensors to move. The graph-based deployment identifies uncovered holes and moves sensors to cover them. The assignment-based deployment computes the locations to be placed with sensors and then dispatches them in an energy-efficient way.

2.1.1. Force-Based Deployment

The work [10] considers moving sensors by virtual forces. Each sensor \( s_i \) is exerted by a compound force \( \vec{F}_i = \vec{F}_{iA} + \vec{F}_{iR} + \sum_{j=1,j\neq i}^n \vec{F}_{ij} \), where \( \vec{F}_{iA} \) is an attractive force by the ROI, \( \vec{F}_{iR} \) is the overall repulsive force by obstacles, \( \vec{F}_{ij} \) is the force produced by sensor \( s_j \), and \( n \) is the total number of sensors. Each force \( \vec{F}_{ij} \) is denoted by \( (r_{ij}, \theta_{ij}) \) in a polar coordinate, where \( r_{ij} \) is the magnitude and \( \theta_{ij} \) is the orientation. \( \vec{F}_{ij} \) is expressed as

\[
\vec{F}_{ij} = \begin{cases} 
  (w_A \cdot (d_{ij} - d_{th}), \theta_{ij}) & \text{if } d_{ij} > d_{th} \\
  (w_R \cdot \frac{1}{d_{ij}}, \pi + \theta_{ij}) & \text{if } d_{ij} < d_{th} \\
  0 & \text{otherwise,}
\end{cases}
\]

where \( w_A/w_R \) is the measure of an attractive/repulsive force, \( d_{ij} \) is the distance between \( s_i \) and \( s_j \), and \( d_{th} \) is a threshold distance to decide the force type. Fig. 1(a) gives an example, where \( d_{12} = d_{14} \). We see that \( s_2 \) exerts an attractive force \( \vec{F}_{12} \), \( s_3 \) exerts a repulsive force \( \vec{F}_{13} \), and \( s_4 \) exerts no force on \( s_1 \) because \( d_{12} > d_{th}, d_{13} < d_{th}, \text{ and } d_{14} = d_{th} \), respectively. Sensor \( s_1 \) is thus moved by the compound force \( \vec{F}_1 \).

In [11], each sensor is viewed as an electron and is repelled by other sensors. The force from a higher sensor density area is greater than that from a lower density area, and the force from a nearer sensor is greater than that from a farther sensor. Specifically, the force function \( F(\cdot) \) should satisfy three rules: (1) \( F(d_{ij}) \geq F(d_{ik}) \) if \( d_{ij} \leq d_{ik} \). (2) \( F(0^+) = F_{max} \). This gives an upper bound on forces. (3) \( F(d_{ij}) = 0 \) if \( d_{ij} > r_c \) (communication distance). This means that only neighboring sensors will generate forces.

Sensors are moved step by step. In each step, the repulsive force on sensor \( s_i \) exerted by a neighboring sensor \( s_j \) is \( \vec{F}_{ij} = \frac{D_l}{|A|} (r_{ij} |p_i - p_j|) \frac{p_i - p_j}{|p_i - p_j|} \), where \( D_l \) is the local sensor density seen by \( s_i, \mu \) is the expected sensor density after the final deployment, and \( p_i/p_j \) is the position of \( s_i/s_j \). The expected sensor density is computed by \( \mu = \frac{N}{|A|} \), where \(|A|\) is the ROI’s area.
Fig. 1(b) shows an example, where \( s_2, s_3, \) and \( s_4 \) all exert repulsive forces on \( s_1 \).

In the above two methods, oscillation check and stability check are performed to examine whether a sensor has reached its final destination. When a sensor \( s_i \) moves back and forth inside a small region many times, it has entered the oscillation state. On the other hand, when \( s_i \) moves less than a threshold distance in a fixed duration, it has entered the stable state. In both cases, \( s_i \) will stop moving.

Reference [12] considers that sensors work under a probability sensing model. The goal is to deploy a minimum number of sensors such that the detection probability of the ROI is above a predefined threshold. To achieve this, we can first deploy sufficient sensors to satisfy the detection probability. Then, sensors can exert repulsive forces on each other. In this way, the number of sensors may be reduced since some sensors may be pushed outside the ROI.

### 2.1.2. Graph-Based Deployment

The work [13] adopts a Voronoi diagram to search uncovered holes and moves sensors to cover these holes. Given a set of sensors on a 2D plane, the Voronoi diagram [14] consists of a number of Voronoi polygons such that each polygon contains one sensor and the points in the polygon are closer to the interior sensor than to other exterior sensors. When the sensing range of a sensor cannot completely cover its Voronoi polygon, there could be an uncovered hole in that polygon. In [13], it proposes the following methods to cover this hole:

**Voronoi-based (VOR) method:** A sensor should move toward the farthest vertex of its current polygon. Fig. 2(b) gives an example, where the dotted polygon is sensor \( s_i \)’s current polygon and \( u \) is the farthest vertex. Sensor \( s_i \) will move along the direction \( s_i u \) and stop at \( v_1 \), where \( |v_1u| = r_s \).

**Minimax method:** A sensor should move to the minimax point of its current polygon, where the minimax point of a polygon is the center of the circle with the minimum radius that can cover the whole polygon (refer to [13] for details about finding the circle). Fig. 2(c) gives an example, where \( v_2 \) is the minimax point of \( s_i \)’s current polygon.

### 2.1.3. Assignment-Based Deployment

Reference [15] focuses on deployment in ROIs with obstacles. It considers two related problems: *sensor placement* and *sensor dispatch*. The former asks how to use the minimum number of sensors in a ROI to guarantee coverage and connectivity. The latter asks how to dispatch mobile sensors to the designated locations computed by the placement result such that their moving energy is minimized.

To solve the placement problem, [15] partitions a ROI \( A \) into single-row and multi-row regions. A single-row region requires one row of sensors to cover it, and a multi-row region requires multiple rows of sensors to cover it. To partition \( A \), we first identify all single-row regions, which is achieved by expanding \( A \)’s boundaries inward and obstacles’ perimeters outward by a distance of \( \sqrt{3}r_{\text{min}} \), where \( r_{\text{min}} = \min\{r_c, r_s\} \). When the expanded line cuts off an obstacle with an area, we take a project from that area to identify a single-row region. Fig. 3(a) gives an example, where 5 single-row regions (with numbers) are identified. Other regions will be multi-row ones. Then, we place sensors in each region as follows:

**Single-row region:** We place a sequence of sensors along the region’s bisector, each separated by a distance of \( r_{\text{min}} \). Fig. 3(b) gives an example.

**Multi-row region:** Two cases are considered, as Fig. 3(c) shows. When \( r_c \geq \sqrt{3}r_s \), adjacent sensors are regularly separated by a distance of \( \sqrt{3}r_s \). When \( r_c < \sqrt{3}r_s \), sensors in each row are separated by a distance of \( r_c \). Adjacent rows are separated by a distance of \( r_s + \sqrt{r_s^2 - \frac{r_c^2}{4}} \) and shifted by a distance of \( \frac{r_c}{2} \). To connect adjacent rows, we add a column of sensors between them, each separated by a distance not larger than \( r_c \).
Given a set of mobile sensors $S$ and a set of locations $L$ computed by the placement result, [15] considers dispatching $S$ to $L$ such that the energy consumption of sensors is minimized. Assuming $|S| \geq |L|$, we construct a weighted complete bipartite graph $G = (S \cup L, S \times L)$, where the weight of each edge $(s_i, l_j), s_i \in S, l_j \in L$, is calculated by $-e_m \times d(s_i, l_j)$, where $e_m$ is the energy cost to move a sensor in one step and $d(s_i, l_j)$ is the shortest distance between $s_i$ and $l_j$. Then, we find a matching $M$ in $G$ with the maximum edge weights, which can be solved by the Hungarian method [16]. For each edge $(s_i, l_j) \in M$, we move sensor $s_i$ to location $l_j$ through the shortest path (refer to [15] for details about finding the shortest path).

Reference [17] focuses on deployment with multilevel coverage. It considers two related problems: $k$-coverage sensor placement and sensor dispatch. The former asks how to use the minimum number of sensors in a ROI to guarantee $k$-level coverage. The latter asks how to dispatch mobile sensors to the designated locations computed by the placement result such that their moving energy is minimized.

To solve the $k$-coverage placement problem, [17] proposes an interpolating placement method based on the placement in Fig. 3(c) ($r_c < \sqrt{3} r_s$ case). Specifically, we see that a large amount of regions in each row are more than 1-covered. So, we can reuse these regions and place the least number of sensors to cover those insufficiently covered regions. Three cases are considered:

**Case of $r_c \leq \frac{\sqrt{3}}{2} r_s$:** In Fig. 3(c), we see that the insufficiently covered regions (marked by gray) are located between adjacent rows. If we place a new $N_1$ row above each original $O_1$ row by a distance of $r_s$, as Fig. 4(a) shows, the ROI becomes 3-covered. Here, sensors in each $N_1$ row are separated by a distance of $r_c$. For $k > 3$, we can apply $\lceil \frac{k}{3} \rceil$ times of this 3-coverage placement and apply $(k \mod 3)$ times of the 1-coverage placement.

**Case of $\frac{\sqrt{3}}{2} r_s < r_c \leq \frac{2 + \sqrt{3}}{2} r_s$:** We can add one extra $N'_r$ row between each $N_i$ and $O_i$ rows to construct a 3-coverage placement, as Fig. 4(b) shows. These $N'_r$ rows are shifted by a distance of $\frac{\sqrt{3}}{2}$ and sensors are separated by a distance of $2 r_c$. For $k > 3$, we can apply $\lceil \frac{k}{3} \rceil$ times of this 3-coverage placement and apply $(k \mod 3)$ times of the 1-coverage placement.

**Case of $r_c > \frac{2 + \sqrt{3}}{2} r_s$:** We can duplicate $k$ sensors on each location in Fig. 3(c).

Given a set of mobile sensors $S$ and a set of locations $L = \{(l_1, n_1), (l_2, n_2), \ldots, (l_m, n_m)\}$ computed by the placement result, where each location $l_j$ will be placed with $n_j$ sensors, [17] proposes a distributed method to dispatch $S$ to $L$ as follows:

1. Each sensor $s_i$ maintains an OCC$_i[1..m]$ table, where each OCC$_i[j] = \{ (s_{j1}, d_{j1}), (s_{j2}, d_{j2}), \ldots, (s_{jn}, d_{jn}) \}, \alpha \leq n_j$, contains the set of sensors $s_{j\alpha}$
that select \( l_j \) as their destinations and their distances \( d_{i,j} \) to \( l_j \). Initially, \( OCC_i[j] = \emptyset, \forall j \). Then, \( s_i \) selects the nearest location \( l_j \) such that \( |OCC_i[j]| < n_j \) as its destination, adds \( (s_i, d(s_i, l_j)) \) in \( OCC_i[j] \), and moves to \( l_j \).

2. Sensor \( s_i \) periodically updates and exchanges its table with one-hop neighbors. When \( s_i \) hears the \( OCC_k \) table from a neighbor \( s_k \), \( s_i \) combines \( OCC_i \) with \( OCC_k \) as follows: For each \( j \), we calculate a union \( U_j = OCC_i[j] \cup OCC_k[j] \). If \( |U_j| > n_j \), we remove the records in \( U_j \) that have longer moving distances, until \( |U_j| = n_j \). Then, we replace \( OCC_i[j] \) by \( U_j \). If \( s_i \) was in the original \( OCC_i[j] \) entry, but is not in the new \( OCC_i[j] \) entry, it means that \( s_i \) is replaced by other sensors with a shorter distance to \( l_j \). Thus, \( s_i \) should reselect another destination.

3. After \( s_i \) reaches \( l_j \), it still exchanges its table with neighbors. Since the sink will eventually observe that all locations are covered, it can notify all sensors to exit from the dispatch method.

2.2. Solutions to Enhancing Coverage and Connectivity of a WSN

After deploying a WSN, some sensors may be broken or may exhaust their energy. These failed sensors may disconnect the network or cause uncovered holes. One can move some mobile sensors to relieve this problem. We introduce two such solutions for enhancing connectivity and coverage of a WSN.

2.2.1. Connectivity Enhancement

Reference [18] considers a static WSN with several isolated groups, called islands. To help these islands communicate with each other, we can add some mobile sensors between them. For two islands \( I_G \) and \( I_H \), the minimum number of mobile sensors required to connect them is \( M_{G,H} = \left\lfloor \frac{d_{G,H}}{r_c} - 1 \right\rfloor \), where \( d_{G,H} = \min_{s_i \in I_G, s_j \in I_H} \{d_{i,j}\} \) is the shortest distance between \( I_G \) and \( I_H \). Let \( N(I_G) \) be the number of sensors in island \( I_G \) and \( W(I_G, m) \) be the optimal set of islands that can be connected by \( m \) mobile sensors starting from island \( I_G \). It can be derived that \( W(I_G, m) = \max\{W(I_G \cup I_H, m - M_{G,H}) + N(I_G \cup I_H)\} \), where \( I_H \) is an island to be directly connected by \( I_G \) and \( N(I_G \cup I_H) = N(I_G) + N(I_H) + M_{G,H} \). However, for an island \( I_G \), if the remaining \( m \) mobile sensors cannot connect to any other island, we set \( W(I_G, m) = 0 \). Using dynamic programming, the minimum \( m \) to connect all islands can be found.

The work [19] considers strengthening the topology of a WSN to be biconnected. First, each cut-vertex is identified. For example, in Fig. 5(a), \( c_1 \) and \( c_2 \) are cut-vertices. By removing cut-vertices, the network is divided into several biconnected components (called blocks). Actually, we can ‘pull’ two neighboring blocks together to eliminate the cut-vertex between them. With this observation, a block movement method is proposed as follows: Given a network topology, we first identify all blocks along with their cut-vertices. A block can have zero, one, or multiple sensors. If two cut-vertices are directly connected, an empty block is established. Then, we can translate the network into a block tree, whose nodes contain blocks and cut-vertices. The block with the maximum number of sensors is the root. In Fig. 5(a), there are 5 blocks (including the empty block \( B_4 \)) and 2 cut-vertices \( c_1 \) and \( c_2 \). Block \( B_1 \) is the root and blocks \( B_2, B_3, \) and \( B_5 \) are leaves. The method executes in two iterations until the network becomes biconnected: (1) Move each leaf block toward the nearest sensor of its parent block, until a new edge appears. (2) If its parent block is empty, we further move it to the upstream cut-vertex of its parent block. Fig. 5(a) gives an example, where \( B_5 \) moves toward \( v \) of its parent block \( B_3 \), and \( B_2 \) and \( B_3 \) move toward the cut-vertex \( c_1 \) since their parent block \( B_4 \) is empty. The final topology is shown in Fig. 5(b).

2.2.2. Coverage Enhancement

The work [20] proposes a bidding protocol to enhance the coverage of a hybrid WSN composed of static and mobile sensors. Static sensors detect uncovered holes locally and bid for mobile sensors by the sizes of holes. It involves the following steps:

1. Each mobile sensor is assigned with a base price, which is an estimation of the hole size when it leaves its current position. Initially, the base price is zero for all mobile sensors. Then, mobile sensors broadcast their positions and base prices in their local areas.
2. Static sensors exchange their positions with their neighbors in two hops to construct a Voronoi diagram. If a static sensor $s_i$ detects an uncovered hole in its Voronoi polygon, it calculates a bid as $\pi \times (d - r_s)^2$, where $d$ is the distance between $s_i$ and its farthest polygon vertex and $r_s$ is the sensing distance. Here, the bid is an estimation of the uncovered hole size. Then, $s_i$ sends its bid to the nearest mobile sensor whose base price is lower than the bid.

3. On receiving bids, a mobile sensor selects the highest bid and moves to cover that hole. Then, it replaces its base price by the selected bid.

The bidding protocol repeats the above steps until no static sensor can give a bid higher than the base price of any mobile sensor.

Reference [21] considers moving sensors close to locations where events could appear. Given a set of event locations, sensors are moved such that their positions can eventually approximate the event distribution. Two moving methods are proposed. In the history-free method, each sensor $s_i$ at position $p_i^{k-1}$ reacts to the appearance of an event at location $l_k$ by moving to a new position $p_i^k = p_i^{k-1} + f_m(d(p_i^{k-1}, l_k))$, where function $f_m(\cdot)$ prohibits a sensor from passing another along the same vector in response to the same event. The history-based method requires sensors to maintain event history to approximate the event distribution. To maintain maximal coverage of the ROI, a sensor is not allowed to move if its movement will leave an uncovered hole.

2.3. Solutions to Assigning Mobile Sensors to Desired Locations

This may involve several issues: sensor relocation, sensor navigation, and sensor dispatch.

2.3.1. Sensor Relocation

Reference [22] divides a ROI into grids and moves sensors from high-density grids to low-density grids. It proposes grid-quorum to move sensors such that the number of exchanged messages are reduced. Specifically, a grid head is selected in each grid to maintain its information. A grid $G_j$ with more sensors sends an advertisement (ADV) to its row to announce that it has extra sensors. On the other hand, a grid $G_j$ with fewer sensors sends a request (REQ) message to its column to ask for extra sensors. These ADV and REQ will meet at a common grid. Fig. 6(a) shows an example, where grid $(1, 3)$ sends ADV to its row and grid $(3, 1)$ sends REQ to its column. They will meet at grid $(1, 1)$. This can reduce the message overhead significantly.

After identifying the targets, sensors are moved by cascaded movement rather than direct movement to prevent a single sensor from consuming too much energy. Fig. 6(b) is a direct movement, where $s_j$ needs to travel a long distance. Fig. 6(c) is a cascaded movement, where $s_j$ first moves to the target, then $s_i$ moves to $s_k$’s original position, and then $s_j$ moves to $s_i$’s original position.

2.3.2. Sensor Navigation

The work [23] considers navigating mobile sensors in a hybrid WSN. It assumes that all sensors do not know their own locations in the ROI. When a static sensor $s_i$ detects an event, it will broadcast a weight request (WREQ) packet to search mobile sensors. On receiving WREQ, a mobile sensor $s_j$ will bid for the event by replying its weight $w_j = \frac{A_j \times h(s_i, m_j)}{e_j}$, where $A_j$ is the area of Voronoi polygon of $m_j$, $h(s_i, m_j)$ is the hop count between $s_i$ and $m_j$, and $e_j$ is the energy of $m_j$. A mobile sensor with a smaller weight will win the bidding.

Static sensors will then guide $m_j$ to $s_i$’s location. An ADV packet is sent along the path from $s_i$ to $m_j$ to build up a navigation field, as Fig. 7 shows. In particular, $s_i$ sets the highest credit $C_1$ for itself. For each rebroadcast of ADV, a lower credit value will be set. Then, $m_j$ will try to move to $s_i$ by repeatedly searching higher credit values.

2.3.3. Sensor Dispatch

Given a set of mobile sensors $S$ and a set of event locations $L$, the work [24] considers dispatching $S$ to $L$ with a concept of load balance. Assuming $|S| \geq |L|$,
we first calculate the energy cost \( w(s_i, l_j) = e_m \times d(s_i, l_j) \) for each sensor \( s_i \in S \) to reach each location \( l_j \in L \), where \( e_m \) is the energy cost to move a sensor in one step. The scheme tries to find a matching \( \mathcal{M} \) between sensors and locations by allowing a bound \( B_j \) for each \( l_j \in L \) as follows:

1. For each \( l_j \in L \), we use a bound \( B_j \) to limit the candidate sensors that \( l_j \) can match with. A sensor \( s_i \) is said as \( l_j \)'s candidate if \( w(s_i, l_j) \leq B_j \). Since a larger bound may lead a sensor to select a farther location, \( B_j \) will be increased gradually. Initially, each \( B_j = \frac{1}{|L|} \sum_{l_i \in L} \min_{s_i, l_i \in S \times L} \{ w(s_i, l_i) \} \).

2. For each unmatched \( l_j \in L \), we find a candidate sensor \( s_i \) with the minimum \( w(s_i, l_j) \) to match with. If \( s_i \) is still unmatched, we add the pair \((s_i, l_j)\) in \( \mathcal{M} \). Otherwise, \( s_i \) must be matched with another location \( l_o \). In this case, \( l_j \) will compete with \( l_o \) for \( s_i \) by three rules: (1) If \( B_j > B_o \), we match \( s_i \) with \( l_j \) to avoid expanding \( B_j \). (2) If \( B_j = B_o \) and \( w(s_i, l_j) < w(s_i, l_o) \), we match \( s_i \) with \( l_j \) to reduce its energy consumption. (3) If \( B_j = B_o \) and \( s_i \) is the only candidate of \( l_j \) but is not that of \( l_o \), we match \( s_i \) with \( l_j \). When \( l_j \) wins the competition, the pair \((s_i, l_j)\) is replaced by the new pair \((s_i, l_j)\) in \( \mathcal{M} \), and \( l_o \) becomes unmatched. Otherwise, \( l_j \) checks other candidate sensors, until there is no candidate.

3. If \( l_j \) cannot find any match, we increase \( B_j \) by \( \Delta_B \) and go to step 2, until a match is found.

4. We repeat steps 2 and 3, until each \( l_j \in L \) can find a sensor to match with.

Fig. 8 gives an example, where \( \Delta_B = 70 \). The initial bound is \( \frac{70 + 97 + 94}{3} = 90 \). In Fig. 8(b), \( l_1 \) matches with \( s_2 \) with bound \( B_1 = 90 \) and \( l_2 \) matches with \( s_3 \) with bound \( B_2 = 90 + 70 = 160 \). Then, after expanding \( B_3 \), \( l_3 \) finds that its candidate \( s_4 \) has been matched with \( l_2 \), so it competes with \( l_2 \) for \( s_4 \). Since \( B_3 = B_2 \) and \( w(s_4, l_2) < w(s_4, l_2) \), \( s_4 \) is replaced by \((s_4, l_3)\) in Fig. 8(c). Similarly, \( l_2 \) obtains \( s_2 \) from \( l_1 \) in Fig. 8(d) and thus \( l_1 \) selects an unmatched sensor \( s_1 \). Fig. 8(e) shows the final result.

When \( |S| < |L| \), a clustering approach is proposed and then the similar matching steps are executed (we omit the details).

Reference [25] considers a mobile WSN as a multi-robot system and addresses the cooperation among robots. Each robot is regarded as a resource and may be required by multiple concurrent missions. It points out that deadlock may happen when some missions never finish executing and resources are tied up, preventing other missions from starting. Then, a deadlock avoidance policy based on the Petri nets is proposed.

2.4. Summary of Mobility Management

Table I summarizes the mobility management methods for mobile sensors. While most methods consider a purely mobile WSN, [18, 20, 23, 24, 25] consider a hybrid WSN. References [18, 20] use mobile sensors to improve the topology of a static WSN, and [23, 24] use static sensors to detect events and send mobile sensors to event locations. For sensors’ detection, [10, 12, 15, 17] consider the probabilistic sensing model. For coverage and connectivity, most deployment methods [10, 11, 13, 15, 17] address both issues, but the work [12] addresses only the coverage issue. References [20, 21, 22] move sensors to improve a WSN’s coverage, while references [18, 19] move sensors to improve the network connectivity. For
energy concern, the dispatch solutions in [15, 17, 23] try to minimize the energy consumption of mobile sensors. Balance of energy consumption is addressed in [22, 24].

3. Path Planning of Data Ferries

3.1. Solutions to Relaying Messages by Data Ferries

Data ferries are a type of mobile sensors that are mainly designed for carrying data. For example, they can travel between isolated sensors to relay information. So, path planning is a critical issue for data ferries to minimize message delay and meet bandwidth requirements. We will discuss two types of path planning: The adaptive planning considers that all sensors are isolated and the final path is adjusted from the initial TSP solution. The probabilistic planning considers that sensors may arbitrarily roam around a ROI and ferries may meet them by a probability model.

3.1.1. Adaptive Planning

Given a data ferry \( F \) and a set of isolated sensors \( S = \{s_1, s_2, \ldots, s_n\} \), the work [26] considers how to schedule a path for \( F \) to visit \( S \). The goal is to exchange data between sensors such that the average message delay is minimized and the bandwidth requirement of each sensor is satisfied. To solve this problem, a 4-step algorithm is proposed:

1. We first calculate an initial path \( p \) by any TSP solution. The message delay between two sensors \( s_i \) and \( s_j \) on \( p \) is defined by \( T_{ij}^p = \frac{|p|}{v} + \frac{d_{ij}}{v} \), where \(|p|\) is \( p \)'s total length, \( v \) is \( F \)'s speed, and \( d_{ij} \) is the distance between \( s_i \) and \( s_j \) on \( p \). It is assumed that \( F \) will repeatedly travel along \( p \). Here, \( \frac{|p|}{v} \) is the average waiting time for \( s_i \) to be visited by \( F \), and \( \frac{d_{ij}}{v} \) is the time for \( F \) to deliver \( s_i \)'s data to \( s_j \). So, the average delay incurred by \( p \) is \( T^p = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{ij} T_{ij}^p \), where \( b_{ij} \) is the average amount of data to be sent from \( s_i \) to \( s_j \).

2. Next, we try to improve \( p \) by two operations:

   - **Edge Replacement:** Let \( s_i s_j \) and \( s_i s_m \) be two edges in \( p \). Let \( p' \) be the path modified from \( p \) by replacing \( s_i s_j \) with \( s_i s_m \) with \( s_i s_l \) and \( s_j s_m \). If \( T^{p'} < T^p \), then we replace \( p \) by \( p' \).
   - **Sequence Reordering:** We construct a new path \( p' \) by moving any \( s_i \) in \( p \) from its original position to another position. If \( T^{p'} < T^p \), then we replace \( p \) by \( p' \).

The above operations are repeated until no better path can be found.

3. Since the communication ranges of sensors may overlap, a time allocation policy \( \Phi_p \) is needed to assign \( F \)'s communication time with sensors. Specifically, we cut \( p \) into \( m \) segments \( \{\xi_1, \xi_2, \ldots, \xi_m\} \) as \( F \) enters or leaves a sensor's communication range, and define \( \Phi_p(s_i, \xi_j) \) as the portion of \( F \)'s communication time with \( s_i \) when \( F \) moves along segment \( \xi_j \). Fig. 9(a) gives an example, where the subpath from \( v \) to \( v \) is cut
into 4 segments $\xi_1$, $\xi_2$, $\xi_3$, and $\xi_4$. $\Phi_p(\cdot, \xi_1) = 0$ since $\mathcal{F}$ cannot communicate with any sensor. $\Phi_p(s_2, \xi_2) = \Phi_p(s_1, \xi_4) = 1$ since $\mathcal{F}$ can only communicate with one sensor. $\Phi_p(s_2, \xi_3) = \Phi_p(s_1, \xi_3) = \frac{1}{2}$ since $\mathcal{F}$ should share its time to $s_1$ and $s_2$.

4. To meet the bandwidth requirement of each $s_i$, $\mathcal{F}$ should spend sufficient time to communicate with $s_i$. If there is no sufficient time, the segments for $s_i$'s traveling path.

A linear programming is formulated to minimize the total extra communication time of $\mathcal{F}$:

$$\min \sum_{j=1}^{m} x_j,$$

$$\text{subject to } \frac{(\xi_d + \sum_{j=1}^{m} \Phi_p(s_i, \xi_j) x_j) \cdot R}{|p| + \sum_{j=1}^{m} x_j} \geq b_i, \quad (1)$$

where $R$ is $\mathcal{F}$'s data rate and $b_i$ is the bandwidth requirement of $s_i$. Here, the numerator and denominator are the expected amount of data that can be sent and received by $s_i$ and the total moving time of $\mathcal{F}$ after extension, respectively. The path $p$ after extension is $\mathcal{F}$'s traveling path.

The work [27] further considers multiple data ferries. Given $n$ sensors and $m$ data ferries, the goal is to find a set of paths for data ferries to visit all sensors such that the average message delay is minimized and the bandwidth requirement of each sensor is met. Four types of solutions are proposed.

**Single-route algorithm (SIRA):** All data ferries will move along the same path and there is no communication between them. Fig. 9(a) gives an example with two ferries. This algorithm directly extends that of [26]. For any path $p$, the delay to deliver data from $s_i$ to $s_j$ on path $T_{ij}^p = \frac{|p|}{2nm} + \frac{d_{ij}}{R}$. So, the average delay of $p$ is $T_p = \sum_{i\in P, j\in P} w_{ij} T_{ij}^p$, where $w_{ij}$ is the weight assigned to each $T_{ij}^p$. Still, edge replacement and sequence reordering are applied to improve $p$. Finally, the linear programming in Eq. (1) can be rewritten as:

$$\min \sum_{i=1}^{n} y_i,$$

$$\text{subject to } R \cdot \frac{(2n + y_i)}{|p| + \sum_{j=1}^{n} y_j} \geq \frac{b_i}{m}, \quad (2)$$

where $y_i$ is the extra moving length of data ferries in the communication range of $s_i$. In Eq. (2), the left-hand term is the product of data ferries' data rate and the ratio of data ferries' communication time allocated to $s_i$, and the right-hand term means that $s_i$'s bandwidth requirement $b_i$ is shared by $m$ data ferries.

**Multi-route algorithm (MURA):** Each data ferry will move along a different path and there is no communication between them. Fig. 9(b) gives an example. In this algorithm, given a set of paths $\mathcal{P}$, we use a 2-tuple $(E_1(\mathcal{P}), E_2(\mathcal{P}))$ as the cost function to evaluate the quality of $\mathcal{P}$, where $E_1(\mathcal{P})$ is the estimated total overload of data ferries in $\mathcal{P}$ and $E_2(\mathcal{P})$ is the estimated total message delay incurred by $\mathcal{P}$ (refer to [27] for details). Intuitively, overload is the amount of data that newly appear and cannot be delivered over a time interval. Initially, we assume that each sensor has a ferry. Let $\mathcal{P}$ be the current path set and $n_i$ be the number of ferries in $p_i \in \mathcal{P}$. We adopt four operations to reduce the number of ferries and to refine the path set $\mathcal{P}$: (1) overlap($p_i, p_j$): We extend path $p_j \in \mathcal{P}$ by including one sensor in path $p_i \in \mathcal{P}$, $p_i \neq p_j$ such that the cost is minimized. (2) merge($p_i, p_j$): We combine $p_i$ and $p_j$ into one new path, and put all $n_i + n_j$ ferries on the new path. (3) merge$^-$($p_i, p_j$): This is the same as merge($p_i, p_j$), except that we decrease the number of ferries by one. (4) reduce($p_i$): We decrease $n_i$ by one for $p_i$ if $n_i > 1$. We iteratively select one operation in a greedy manner to minimize the cost, until there are only $m$ paths. After obtaining $m$ paths, we can apply SIRA to optimize each path.
Node replying algorithm (NRA): In this scheme, each data ferry will move along a different path and static sensors will serve as relay nodes to propagate data from paths to paths. First, the ROI is divided into \( c_1 \times c_2 \) grids, where \( c_1 \times c_2 \leq m \), and each grid will be served by a ferry that travels on a path constructed by SIRA. Among all possible combinations of \( c_1 \) and \( c_2 \), we select the one with the minimum cost (as defined in MURA). Suppose that grids \( G_s \) and \( G_d \) want to exchange data. To relay data between them, we will try to connect \( G_s \) and \( G_d \) directly or indirectly. Two grids can be connected using the overlap\((p_i, p_j)\) operation in MURA to find a relaying node. Fig. 9(c) gives an example, where there are 4 grids and 4 paths. Then, these paths will be connected by extending one to another.

Ferry relaying algorithm (FRA): Like NRA, the ROI is divided into grids, each to be served by one data ferry. Data ferries may exchange their data when they meet with each other. Contact points are designated along grid boundaries for this purpose, as shown in Fig. 9(d). These contact points are separated by a distance of one half of the grid boundary, so each ferry have up to eight contact points to communicate with other ferries. Data ferries of any two adjacent grids will move in reverse directions of each other. To guarantee that data ferries can meet at contact points, \([27]\) suggests extending the path in each grid by connecting to the contact points and also extending paths such that they have the same lengths.

The work \([28]\) considers that sensors may have different communication ranges. A data ferry only needs to touch any point within the communication range of each sensor to collect its data. Thus, the moving path of the ferry can be further reduced. Fig. 10 shows an example, where the ferry is initially placed at \( l_0 \). We can observe that the path \( l_0 \rightarrow l_1 \rightarrow l_2 \rightarrow l_0 \) is shorter than the path \( l_0 \rightarrow s_1 \rightarrow s_2 \rightarrow l_0 \). Here, \( l_1 \) and \( l_2 \) are called touching points of sensors \( s_1 \) and \( s_2 \), respectively. Based on this observation, \([28]\) adopts evolutionary algorithms to calculate the touching points of sensors.

3.1.2. Probabilistic Planning

Given a set of mobile sensors and a data ferry \( F \), \([29]\) considers planning \( F \)'s path such that the overall probability that \( F \) can meet each mobile sensor is larger than a predefined threshold \( \tau \) and the path length is minimized. It is assumed that these mobile sensors may move following a predefined mobility model. It is also assumed that \( F \) will stop at a few points for some periods of time when moving along the path. A 3-step solution is proposed:

1. We divide a ROI into grids. Let \( L \) be the set of the central point of each grid, called way-point, as shown in Fig. 11. For each \( l_i \in L \), let \( c_i \) be the circle centered at \( l_i \) and with a radius \( r_c \). We define \( g(c_i, s_j) \) as the instantaneous contact probability that \( F \) can meet sensor \( s_j \) inside \( c_i \) at a time instance, and \( h(c_i, l_i, s_j) \) as the time-cumulative contact probability that \( F \) can meet sensor \( s_j \) inside \( c_i \) when \( F \) stays at \( l_i \) for a time period \( t_i \). These probabilities depend on the mobility model of mobile sensors. In \([29]\), these probability are developed for both a periodic mobility model and a random way-point mobility model.

2. We then select a subset \( L' \subseteq L \) and determine the time \( t_i \) for \( F \) to stay in each way-point \( l_i \in L' \) with the following objective:

\[
\min \{ \sum_{l_i \in L'} t_i + 3d(l_i, O) \}, \quad (3)
\]
Given a set of sensors $S$ and a data ferry $F$, [30] considers planning $F$’s path to visit some sensors in $S$ such that the $F$’s moving distance (or time) can be bounded by a pre-defined threshold, and the network lifetime is maximized. Suppose that $F$ will move from a location $l_a = (x_a, y_a)$ to another location $l_b = (x_b, y_b)$. The idea is to recursively pick a turning point between $l_a$ and $l_b$, until we can find a path $l_a \rightarrow l_1 \rightarrow \cdots \rightarrow l_m \rightarrow l_b$ such that the distance (or time) bounded can be meet, and the network lifetime is maximized when $F$ moves along the path, where $l_1, l_2, \cdots, l_m$ are the turning points. A divide-and-conquer scheme is proposed as follows:

1. Given two locations $l_a$ and $l_b$, we select a set of possible turning points such that each point locates at $(x_c + \Delta x, k\Delta y)$, where $k$ is an integer and $\Delta y$ is a constant such that every turning point will be inside the ROI. Among these turning points, we select the point $l_v$ and construct a path $l_a \rightarrow l_v \rightarrow l_b$ such that the network lifetime can be maximized when $F$ moves along that path (refer to [30] for the details about calculating the network lifetime). Fig. 12(a) gives an example, where there are 4 turning point and a path $l_a \rightarrow l_2 \rightarrow l_b$ is constructed.

2. We divide sensors into two groups according to their distances to the line segments $l_a \rightarrow l_v$ and $l_v \rightarrow l_b$ (a sensor will favor the closer line segment). For example, in Fig. 12(b), sensors $s_1, s_2$, and $s_3$ are in one group, while $s_4$ and $s_5$ are in another group.

3. For each cluster of sensors, we recursively execute the above two steps, until the distance (or time) bounded is reached. Fig. 12(c) shows the final result, where there are two iterations.
scheduling a cyclic path for $F$ to visit a subset of nodes $V' \subseteq V$, such that $B \in V'$, the path length is not longer than $l_{\text{max}}$, and the overall hop count along $T$ from each sensor to a node in $V'$ is minimized. We denote by $\delta_{TSP}(V')$ the length of a path calculated by any TSP solution to traverse all nodes in $V'$. This algorithm involves five following steps:

1. Initially, $V' = \{B\}$.

2. Then, we construct a candidate set $W$ as follows: For each $v \in V - V'$, we add $v$ to $W$ if $\delta_{TSP}(V' \cup \{v\}) \leq l_{\text{max}}$. If $W = \emptyset$, the algorithm is terminated.

3. For each $v \in W$, we calculate its utility by

$$U(v) = \frac{\sum_{s_i \in S} d_T(s_i, v') - \sum_{s_i \in S} d_T(s_i, V' \cup \{v\})}{\delta_{TSP}(V' \cup \{v\}) - \delta_{TSP}(V')}$$

where $d_T(s_i, v')$ is the hop count along $T$ from $s_i$ to a node in $V'$. Here, the utility of $v$ is the ratio of the reduction of total hop count that data has to be relayed along $T$ to the increase of $F$’s length after adding $v$. We then add the node with the maximum utility to $V'$.

4. After adding a new node, we recalculate the utility of each $s_i \in V'$. If any $s_i \in V'$ has $U(s_i) = 0$, we remove it from $V'$.

5. If all sensors are included in $V'$, the algorithm is terminated. Otherwise, we go to step 2.

Fig. 13 gives an example. We will include $s_1$ and $s_2$ into $V'$ in the first two iterations. In the third iteration, $s_3$ is added. Since $U(s_1)$ becomes zero, we remove $s_1$ from $V'$. The final path is $B \rightarrow s_2 \rightarrow s_3 \rightarrow B$.

3.2.3. Single-Hop Collection

The work [34] considers planning a ferry’s path to travel in a WSN such that each sensor can directly communicate with the ferry. Let $L = \{l_1, l_2, \ldots, l_k\}$ be a set of candidate polling points which contains all sensor locations and some predefined locations. Let $N_i$ be the set of sensors that the ferry can directly communicate with when it arrives at point $l_i \in L$. The goal is to find a subset of polling points $L' \subseteq L$ such that each sensor belongs to at least one $N_i$, $l_i \in L'$, and the total length is minimized. Initially, we set $L' = \{B\}$, where $B$ is the base station. Then, a greedy solution is proposed to iteratively add a polling point $l_i$ in $L$ with the minimum covering cost $\tau(i)$, until all sensors belong to at least one $N_i$, $l_i \in L'$. Here, we define $\tau(i) = \min_{\{l_j \in L'\}} \frac{d(l_i, l_j)}{|N_i \cap \{l_j\}|}$, where $U$ is the set of sensors that are not in any $N_i$, $l_i \in L'$ and $d(l_i, l_j)$ is the distance between two polling points $l_i$ and $l_j$. Then, a TSP scheme can be applied for the data ferry to visit all points in $L'$.

Reference [35] extends the above work by assuming that each sensor is equipped with one antenna and the data ferry is equipped with two antennas such that the data relay may be performed with two sensors simultaneously by a space division multiple access (SDMA) technology. It redefines the coverage region when the ferry stays at a location and then a similar path planning scheme in [34] is adopted.

3.2.4. Distributed Navigation

Unlike the previous centralized approaches, some efforts focus on designing a fully distributed protocol to navigate a data ferry for data collection. Given a set of sensors without location information and a data ferry with an antenna system which can accurately compute the direction of arrival (DOA) for received signals, reference [36] proposes a distributed navigation protocol to visit some representative sensors. These representative sensors are called navigation agents (NAs). Then, the data ferry is navigated by the intermediate sensors between these NAs, called intermediate navigators (INs). Fig. 14 shows an example. A 3-phase protocol is proposed:

1. Identification of NAs and INs: The set of NAs should be a dominating set of this network. The heuristic in [37] is adopted. First, a spanning tree rooted at any sensor is formed. Second, nodes mark themselves as NAs as follows:

   - The root declares itself as a NA by broadcasting a Declare-NA message.
When a sensor receives a Declare-NA, it will give up becoming a NA by broadcasting an Accept-NA message.

When a sensor receives Accept-NA from all lower-depth neighbors, it will declare itself as a NA by broadcasting a Declare-NA message.

This process is repeated until each sensor is either a NA or a one-hop neighbor of a NA. Then, for each pair of NAs, the nodes passed by the shortest path (in terms of hop count) between these two NAs are marked as INs.

2. Path computation: A path $\mathcal{P}$ is formed to visit each NA. The work proposes adopting the ant colony optimization-TSP solution [38].

3. Navigation: Finally, the data ferry travels along $\mathcal{P}$ with the assistance of INs based on a DOA model. When visiting a NA, both NA and those sensors dominated by NA will send their data to the ferry.

Reference [39] extends the above protocol to the k-hop data collection scheme where sensors that are within $k$ hops from a NA can send their data to the NA (and thus the ferry). To reduce the latency to deliver data to a NA, a sensor can pre-transmit its data to a sensor that is 1-hop away from a NA.

3.3. Summary of Path Planning

Table II summarizes the path-planning methods for data ferries. While most methods consider centralized approaches, [36, 39] uses sensors to navigate a data ferry in a distributed manner. References [30, 31, 32, 34, 35, 36, 39] consider that data sent from sensors to a data ferry can be multi-hop transmission; other work [26, 27, 28, 29, 34, 35, 36] consider that ferries should directly communicate with each sensor. For the issue of communication time, [26, 27] extend the communication time of sensors to meet their bandwidth requirements, [29] minimizes the total waiting time of a ferry at each point along the path, and [35] adopts an physical layer technology to help a ferry quickly collect data from sensors. For energy concern, [30] balances the traffic loads among sensors, while [31, 32, 34, 35, 36, 39] reduce the total energy consumption of sensors. For the length concern, [30, 31, 32] give constraints on path lengths, while [26, 27, 28, 29, 34, 35, 36, 39] try to minimize path lengths.

4. Platforms and Applications of Mobile WSNs

Below, we review some interesting platforms and applications. Mobile Emulab [40] is a robotic testbed developed for mobile WSNs. Mobile sensors are robots that carry single-board computers and sensing devices. Remote users can control these mobile sensors in a real-time and interactive way, or through a script. Fig. 15 shows its system architecture. The video cameras will overlook the ROI and track mobile sensors. Snapshots are periodically reported to the vision system. Through image processing, the positions of mobile sensors are determined. The robot system can send motion commands to mobile sensors, which can report their sensing data to the robot system. On the other hand, the robot system can query the current positions of mobile sensors via the robot-backend system. Remote users can send motion requests to control mobile sensors, or send event requests to obtain the ROI’s status.

![Fig. 15. The system architecture of Mobile Emulab.](image-url)
Visual surveillance systems typically collect a large amount of images from video cameras, which require a huge computation cost to analyze. Introducing the intelligence of mobile WSNs can help reduce such overheads while supporting more advanced, context-rich services. The iMouse system [41] is proposed to integrate static sensors and mobile sensors, as Fig. 16(a) shows. Static sensors continuously monitor the ROI and notify the server when detecting abnormal events. Once receiving such notifications, the server will dispatch mobile sensors to take snapshots at event locations. Thus, iMouse can avoid recording unnecessary images when nothing happens. Fig. 16(b) shows the components of a mobile sensor, which consists of a LEGO car carrying a MICAz mote, a webcam, an 802.11 WLAN card, and a Stargate. Fig. 16(c) shows an experimental grid-link deployment.

Robomote [42] is a mobile platform with MICA2 motes and some infrared sensors to detect obstacles. Two case studies have been tested on this platform. Based on the sensor-based path-planning scheme [43], it uses one Robomote to move along a desired contour constructed by querying neighboring sensor readings. The second case is to implement a tracking algorithm proposed in [44] to locate the light source by a Robomote.

The work in [45] uses a WSN to implement the pursuer-evader game. There is a moving object (called evader) and a data ferry (called pursuer). The evader roams around arbitrarily and the pursuer tries to intercept the evader based on the data reported by static sensors. One challenge for static sensors is how to quickly tell the pursuer where the evader is. To address this issue, static sensors detecting the evader will elect a leader to report to the pursuer. Such reports are sent to the moving pursuer through a landmark routing [46], which operates over a tree-building mechanism. Finally, the pursuer will determine the interception path to chase the evader. The platform is developed based on MICA2 motes, an 802.11 WLAN card, and high-precision differential GPS devices.

The work in [47] proposes an implementation of data ferries. Two critical issues are addressed: (1) how to reduce the speed of a data ferry when its MAC layer encounters interference and collision, and (2) how to construct the relaying path from each sensor to a data ferry’s moving path. To address the first issue,
this work designs an adaptive speed control algorithm to determine whether a data ferry should slow down depending on its current data deliver rate. Specifically, a data ferry has three speeds: SLOW, STOP, and FAST. A sensor can indicate how much data that it wishes to transfer in a packet header. Then, the data ferry can select a speed accordingly. To address the second issue, a data ferry can broadcast an interest message to help sensors learn their distances to the data ferry’s moving path.

5. Conclusions

Static WSNs have limitations on supporting multiple missions and handling different situations when network conditions change. Introducing mobility to WSNs can improve the network capability and thus relieve the above limitations. This article provides a comprehensive survey of current works on mobile WSNs. Various mobility management and path-planning schemes have been discussed. Also, several mobile platforms and applications have introduced.

Acknowledgment


References


Authors’ Biographies

You-Chiun Wang received his BEng and MEng degrees in Computer Science and Information Engineering from the National Chung-Cheng University and the National Chiao-Tung University, Taiwan, in 2001 and 2003, respectively. He obtained his Ph.D. degree in Computer Science from the National Chiao-Tung University, Taiwan, in October of 2006. Currently, he is a postdoctoral research fellow at the Department of Computer Science, National Chiao-Tung University, Taiwan. His research interests include wireless network and mobile computing, communication protocols, and wireless sensor networks. Dr. Wang served as a Local Arrangement Vice Chair in the IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS), 2007, as a Guest Editor for The Computer Journal of the special issue on “Algorithms, Protocols, and Future Applications of Wireless Sensor Networks”, 2009, as a Member in the Editor Board of the IARAI International Journal on Advances in Networks and Services, 2009-present, and as TPC members of several international conferences. He is a member of the IEEE and the IEEE Communication Society.

Fang-Jing Wu received the B.S. degree in Mathematics form the Fu Jen Catholic University and the M.S. degree in Computer Science and Information Engineering from the National Chiao-Tung University, Taiwan, in 2001 and 2004, respectively. She was a research assistant in the Department of Communication Engineering, National Chiao-Tung University, Taiwan, in 2004. She is currently pursuing Ph.D. in the Department of Computer Science, National Chiao-Tung University, Taiwan. Her current research interests are primarily in pervasive computing and wireless sensor networks.

Yu-Chee Tseng obtained his Ph.D. in Computer and Information Science from the Ohio State University in January of 1994. He is Professor (2000-present), Chairman (2005-present), and Associate Dean
(2007-present) at the Department of Computer Science, National Chiao-Tung University, Taiwan. He is also Adjunct Chair Professor at the Chung Yuan Christian University (2006-present).

Dr. Tseng received Outstanding Research Award, by National Science Council, ROC, twice in periods 2001-2002 and 2003-2005. Best Paper Award (Int’l Conf. on Parallel Processing, 2003), the Elite I. T. Award in 2004, and the Distinguished Alumnus Award, by the Ohio State University, in 2005. His research interests include mobile computing, wireless communication, and parallel and distributed computing.