# Message-Efficient In-Network Location Management in a Multi-sink Wireless Sensor Network

Chih-Yu Lin and Yu-Chee Tseng
Department of Computer Science
National Chiao-Tung University
Hsin-Chu, 300, Taiwan

Ten H. Lai
Department of Computer
Science and Engineering
The Ohio State University

## Abstract

*A wireless sensor network consists of many tiny sensor nodes. The distributed memory spaces of sensors can be considered as a large distributed database, in which one can conduct in-network data processing. This paper considers a sensor network used for object tracking where distributed location updates and queries are performed inside the network. Although this issue has been intensively studied for cellular networks, the same problem in sensor networks has very different characteristics. In this paper, we propose an efficient location management scheme for object tracking in a multi-sink sensor network where users can inquire the locations of objects via any sink in the network. A message-efficient algorithm that describes how to perform location updates and queries is proposed. Furthermore, two distributed virtual tree construction algorithms are also presented. The goal is to reduce the overall update and query cost. The efficiency of the proposed algorithms is evaluated and verified by simulations.*

**Keywords:** object tracking, in-network processing, sensor network, data aggregation, location management.

## 1 Introduction

The emerging wireless sensor network (WSN) technology may greatly facilitate human life. A WSN may consist of many inexpensive wireless nodes, each capable of collecting, processing, and storing environmental information, and communicating with other nodes. A lot of research efforts have been dedicated to WSNs, including deign of physical and medium access layers [14, 15] and routing and transport protocols [7, 8]. Applications of WSNs have been studied in [1, 5, 12].

Object tracking is an important application of WSNs (e.g., military intrusion detection and habitat monitoring). The key steps involved in tracking include event detection, target classification, and location estimation [2, 4, 10, 13]. In a WSN, when the locations of objects are successfully determined, a location management scheme for reporting objects' locations and disseminating users' queries is required [9, 11]. The main theme of this paper is location management. In particular, we explore the in-network data processing capability of WSNs by executing distributed location updates and queries inside the network. Updates are initiated when an object moves from one sensor to another. Queries are invoked to find out objects' locations. Location updates and queries are tradeoffs and may be done in various ways. A naive way for delivering a query is to flood the whole network. The sensor who knows the location of the queried object will reply to the query. This is clearly inefficient and not scalable. Alternatively, if all location information is stored at a designated sensor (e.g., the sink), no flooding is required. However, any movement has to be updated to that sensor. The cost is not justified when objects move frequently or when the query rate is low. The purpose of this work is to strike a balance between these two extreme approaches.

The in-network location management problem has been studied in [9, 11]. In [9], sensors are organized as a logical tree. When an object moves from one sensor to another, update messages are only forwarded to the lowest common ancestor of these two sensors in the tree. Further, queries are only forwarded along the path from the sink to the sensor containing the queried object. This work fails to consider the physical structure of the WSN. Reference [11] further takes the physical structure of the network into consideration while constructing the logical tree. This results in further reduction of the overall update and query cost.

In [11], it is assumed that there is only one sink in the network. In this work, we explore the possibility of having multiple sinks in the network. One advantage of having multiple sinks is for faster query response. In addition, using multiple sinks can also relieve the traffic congestion problem associated with a single-sink system (i.e., using multiple sinks can achieve load balance more easily). In or-

der to support location management in a multi-sink WSN, we can extend the tree structure used in the single-sink system [11] by constructing a logical tree for each sink. However, this implies that updating multiple trees is required when a movement event is detected. Assuming that there are $m$ sinks coexisting in the network, if each tree is updated independently, the update cost will become approximately $m$ times. It is desirable to further reduce the update cost when multiple trees coexist in the network. In this paper, we propose an algorithm for updating multiple trees using the concept of data aggregation. For example, with proper design, we show that the update cost only increases about 10 times when the number of trees (i.e., the number of sinks) increases from 1 to 1024. Based on the foregoing update algorithm, we formulate the update cost that gives us hints to develop efficient tree-construction algorithms. Two distributed multi-tree construction algorithms are presented in this paper. Finally, we show that the increased update cost with multiple trees can be compensated by lower query cost. It is found that the query cost (and thus the total update and query cost) depends on $m$, the number of sinks. This allows us to further investigate how to choose the value of $m$ under different scenarios.

The remainder of this paper is organized as follows. Sec. 2 formally defines the multi-sink object-tracking problem. The proposed in-network update and query mechanisms are discussed in Sec. 3. Sec. 4 presents two distributed multi-tree construction algorithms. Performance studies are given in Sec. 5. Sec. 6 draws our conclusions.

# 2 Preliminaries

## 2.1 Network Model

We consider a WSN to be used for object tracking. We adopt a simple *nearest-sensor tracking* model, in which the sensor that receives the strongest signal from an object is responsible for tracking the object (this can be achieved by [6] and we omit the details). Therefore, the sensing field can be modelled by a *Voronoi graph* [3], as depicted in Fig. 1(a), where each sensor's responsible area is the polygon containing itself. Two sensors are called *neighbors* if their sensing ranges share a common boundary on the Voronoi graph. Multiple objects may be tracked concurrently by the network, and we assume that from mobility statistics, it is possible to collect the frequency that objects move between each pair of neighboring sensors, called the *event rate*. For example, in Fig. 1(a), the arrival and departure rates between sensors are shown on the edges of the Voronoi graph. In addition, the communication ranges of sensors are assumed to be large enough so that neighboring sensors can communicate with each other directly. Thus, the WSN is modelled by an undirected weighted graph $G = (V_G, E_G)$, where $V_G$ represents sensors and $E_G$ represents neighborhood relationship of sensors. The weight of each link $(a, b) \in E_G$, denoted by $w_G(a, b)$, is the sum of event rates from $a$ to $b$ and from $b$ to $a$. For example, Fig. 1(b) shows the corresponding weighted graph of the sensor network in Fig. 1(a).

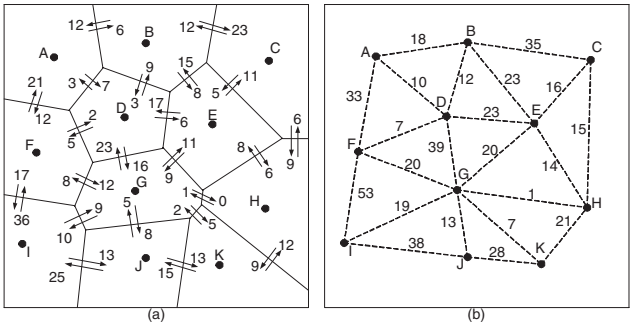


**Figure 1. (a) The Voronoi graph of a sensor network. The numbers around arrows are event rates. (b) The weighted graph $G$ corresponding to the sensor network in (a).**

## 2.2 From Single-sink to Multi-sink WSNs

In [11], an in-network location management scheme for a single-sink WSN is proposed. First, a tree $T$ rooted at the sink is constructed. If an object moves from one sensor to another, update messages will be forwarded to the lowest common ancestor of these two nodes in $T$. For example, in Fig. 2, a tree rooted at sensor $A$ is constructed. When $Car1$ moves from $H$ to $C$, an update message will be forwarded from $H$ to $B$ and another message from $C$ to $B$. This allows each node $x$ to always keep a fresh list of objects that are currently tracked by each of the subtrees rooted at $x$'s children. When a user in $F$ inquires $Car1$'s location, the query will be sent to the sink first and then forwarded along a path of the tree according to the lists maintained by sensors, as shown in Fig. 2.

In this work, we assume that multiple sinks coexist in $G$. Our goal is to reduce the number of messages transmitted for update and query. A naive way to extend a single-sink system to a multi-sink system is to construct a virtual tree $T_x = (V_G, E_{T_x})$ for each sink $x$, where $E_{Tx} \subseteq E_G$. For example, Fig. 3(a) extends the network in Fig. 2 such that both sensors $A$ and $B$ are sinks. Three issues need to be addressed when multiple trees coexist.

1. **Update and query mechanisms:** When an object moves, updating multiple trees is required in a multi-sink system. If we apply the same update mechanism used in a single-tree system to each tree independently,

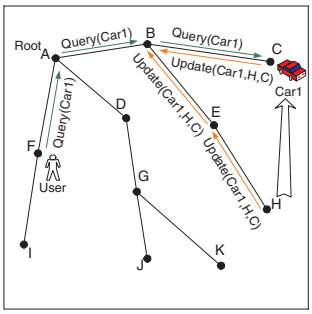


**Figure 2. An example of the single-sink system. The tree is rooted at $A$ and is constructed from the $G$ in Fig. 1(b).**

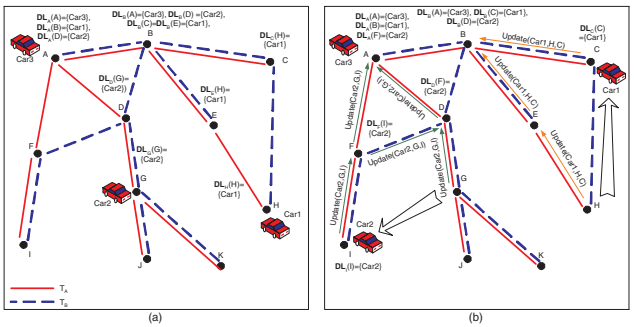


**Figure 3. (a) The $DL$s stored in sensors. Entries with empty set are not shown. (b) An example where $Car2$ moves from $G$ to $I$ and $Car1$ moves from $H$ to $C$.**

the update cost will increase approximately $m$ times, where $m$ is the number of trees. This is apparently inefficient. Update aggregation can be done to reduce cost. Further, the query mechanism should be designed carefully. We will show that the query paths from sinks to the target sensor may cause a cycle. The cycle problem should be avoided.

2. **Multi-tree construction:** The proposed update and query mechanisms can be applied to any multi-tree system. However, different multi-tree construction algorithms will cause different update costs. We will formulate the update cost and point out the factors that affect the update cost. Then, we propose two efficient distributed multi-tree construction algorithms.

3. **The number of trees used:** Obviously, using multiple trees will increase update cost; however, the increase can be compensated by lower query cost (this will be verified further in simulation). Because both the update cost and the query cost are affected by the number of trees used, we will also investigate the proper value of $m$ under different scenarios.

## 3 Location Management in Multi-Sink WSNs: Update and Query Mechanisms

We consider a WSN with $n$ sensors, $m$ of which (denoted by $\sigma_i, i = 1, \cdots, m$) are designated as sinks. For each sink $\sigma_i$, we assume that a tree $T_{\sigma_i}$ rooted at $\sigma_i$ has been constructed from $G$. We will focus on the update and query mechanisms in this section and discuss the tree construction issue in Sec. 4. Table 1 summaries the notations used in this paper.

Each sensor $x$ keeps two tables in order to process updates and queries:

**Table 1. Summary of notations.**

| | |
|---|---|
| $dist_G(u, v)$ | The minimum hop count between $u$ and $v$ in $G$. |
| $nei(v)$ | The neighbors of $v$ in $G$. |
| $dist_{T_{\sigma_i}}(u, v)$ | The hop count of the path connecting $u$ and $v$ in $T_{\sigma_i}$ |
| $w_G(u, v)$ | The event rate between $u$ and $v$. |
| $lca_i(u, v)$ | The lowest common ancestor of $u$ and $v$ in $T_{\sigma_i}$. |
| $p_i(v)$ | The parent of $v$ in $T_{\sigma_i}$. |
| $\sigma_i$ | The root of $T_{\sigma_i}$. |
| $st\_root_i(v)$ | The root of the temporary subtree containing $v$ during the construction process of $T_{\sigma_i}$. |

- *Subtree_Member* $S_x$: It is an $m \times n$ table to indicate whether another sensor is a descendant of $x$ in a certain tree. Specifically, $S_x(T_{\sigma_i}, j) = 1$ if sensor $j$ is a descendant of $x$ in tree $T_{\sigma_i}$; otherwise, $S_x(T_{\sigma_i}, j) = 0$. For example, in Fig. 3(a), $S_D(T_B, F) = 1$ and $S_D(T_A, F) = 0$. All values in this table will not change after all trees are through with construction.

- *Detected_List* $DL_x$: It is a table with $k + 1$ entries, where $k$ is the number of neighbors of $x$. Each entry maintains a set of objects. For sensor $x$ itself, $DL_x(x)$ contains the objects currently being tracked by $x$. For each neighbor $y$ of $x$, $DL_x(y)$ contains all objects that are currently being tracked by the subtrees of some $T_{\sigma_i}$, $i = 1, \cdots, m$, rooted at $y$, i.e., $DL_x(y) = \{o | \exists z, i \text{ s.t. } o \in DL_z(z), S_y(T_{\sigma_i}, z) = 1, \text{and } x = p_i(y))\}$. This implies that if $o$ is tracked by sensor $z$ currently and $y$ is an ancestor of $z$ in a certain tree, then $x$ can know how to find $o$ by asking $y$. For example, in Fig. 3(a), $D$ is a neighbor of

$A$. Because $S_D(T_A, G) = 1$ and $Car2$ is tracked by $G$, $Car2 \in DL_A(D)$. *Detected_List* is a dynamic table. When an object moves from one sensor to another, *Detected_List*s maintained by some sensors have to be modified accordingly.

### 3.1 The Location Update Mechanism

The goal of location update is to ensure that the *Detected_List*s of sensors are fresh. The main idea here is that when an object $o$ moves from sensor $a$'s responsible polygon to sensor $b$'s responsible polygon, for each sink $\sigma_i$, the update messages should be sent from $a$ and $b$ along $T_{\sigma_i}$ to $lca_i(a, b)$, the lowest common ancestor of $a$ and $b$ in $T_{\sigma_i}$. The reason for doing so is that the *Detected_List*s of the ancestors of $lca_i(a, b)$ will not be affected by this movement. Furthermore, instead of allowing all trees to update independently, we will update trees simultaneously with some data aggregation techniques. We make the following observation. In a system with $m$ trees, a sensor $x$ needs to maintain $p_i(x)$ for each $T_{\sigma_i}$, $i = 1, \cdots, m$. Because the number of neighbors of $x$ may be smaller than $m$, some of the $p_i(x)$s may be duplicate and thus can be updated together. This also implies that when a node $y$ receives an update message, it should update its *Detected_List* by considering several trees rather than one tree. Thus, the update mechanism comprises two parts: (1) the forwarding rule of the update message, and (2) the updating rule of the *Detected_List*. The update message sent for the event that an object $o$ moves from sensor $a$ to sensor $b$ is denoted by $Update(o, a, b, eventid)$, where $eventid$ is to uniquely represent this event.

*Forwarding Rule*: When an object $o$ moves from sensor $a$ to sensor $b$, for each tree $T_{\sigma_i}$, every node on the tree paths from $a$ to $lca_i(a, b)$ and from $b$ to $lca_i(a, b)$ should receive the update message at least once. We can note that if a node $x$ is on the path from $a$ to $lca_i(a, b)$ in $T_{\sigma_i}$ and $x \neq lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 1$ and $S_x(T_{\sigma_i}, b) = 0$. Similarly, if $x$ is on the path from $b$ to $lca_i(a, b)$ in $T_{\sigma_i}$ and $x \neq lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 0$ and $S_x(T_{\sigma_i}, b) = 1$. If $x$ is $lca_i(a, b)$, then $S_x(T_{\sigma_i}, a) = 1$ and $S_x(T_{\sigma_i}, b) = 1$. Thus, when any node $x$ receives a new $Update(o, a, b, eventid)$ message, it can use the following statement to determine whether it is on the tree paths from $a$ to $lca_i(a, b)$ or from $b$ to $lca_i(a, b)$:

$$\exists i((S_x(T_{\sigma_i}, a) = 0 \wedge S_x(T_{\sigma_i}, b) = 1) \vee (S_x(T_{\sigma_i}, a) = 1 \wedge S_x(T_{\sigma_i}, b) = 0)) \quad (1)$$

(Note that Eq. 1 includes the special cases of $x = a$ and $x = b$, in which the movement of $o$ rather than receiving an update message will make $x$ checking Eq. 1.) If $x$ receives the update message for the first time and there is a tree $T_{\sigma_i}$ making Eq. 1 true, then an update message should be sent to $p_i(x)$. However, note that if two trees $T_{\sigma_i}$ and $T_{\sigma_j}$ both satisfy Eq. 1 and $p_i(x) = p_j(x)$, then only one update message needs to be sent (the same applies if multiple trees satisfy Eq. 1). This is what we mean by data aggregation.

*Updating Rule*: When a node is notified that an object $o$ moves from sensor $a$ to sensor $b$, it will update its *Detected_List* as follows.

- For sensor $a$, it will remove $o$ from $DL_a(a)$ and check whether there exist a tree $T_{\sigma_i}$ and a neighbor $y$ such that $S_a(T_{\sigma_i}, b) = 1$ and $a = p_i(y)$. If the answer is affirmative, this implies that $a$ can find $o$ by asking $y$. Thus, it adds $o$ is into $DL_a(y)$.
- For sensor $b$, it will add $o$ into $DL_b(b)$ and remove $o$ from other entries of $DL_b$ if $o$ appears in other entries.
- For any other sensor $x$ that receives the update message from $y$, if $\exists i(S_x(T_{\sigma_i}, b) = 1 \wedge x = p_i(y))$ is true, this implies that $x$ can find $o$ by asking $y$; thus $o$ will be added to $DL_x(y)$. Otherwise, $o$ will be removed from $DL_x(y)$ if $o$ appears in $DL_x(y)$.

Fig. 3(b) shows an example where $Car2$ moves from $G$ to $I$ and $Car1$ moves from $H$ to $C$. The modified $DL$s and the reported messages are shown in the figure.

Next, we derive the number of messages required to be sent per unit time for location update:

$$U = \left(\sum_{i=1}^{m} U(T_{\sigma_i})\right) - \left(\sum_{v \in V_G} SC(v)\right), \quad (2)$$

where $U(T_{\sigma_i})$ is the update cost for tree $T_{\sigma_i}$ if $T_{\sigma_i}$ is the only tree in the network and $SC(v)$ is the saved cost for sensor $v$ due to the overlap of tree edges among $m$ trees. $U(T_{\sigma_i})$ can be formulated as

$$U(T_{\sigma_i}) = \sum_{\substack{(u,v) \in E_G \wedge \\ (u,v) \notin E_{T_{\sigma_i}}}} (w_G(u, v) \times (dist_{T_{\sigma_i}}(u, lca_i(u, v)) + dist_{T_{\sigma_i}}(v, lca_i(u, v)))), \quad (3)$$

where $dist_{T_{\sigma_i}}(x, y)$ is the hop count of the path connecting $x$ and $y$ in $T_{\sigma_i}$. To explain the meaning of Eq. 3, we assume that $T_{\sigma_i}$ is the only tree in the network. When an event occurs on $(u, v)$, the update messages will be forwarded to $lca_i(u, v)$ according to the forwarding rule. Eq. 3 is similar to the cost function for a single tree in [11], except that when $(u, v) \in E_{T_{\sigma_i}}$ there is no cost because either $u$ or $v$ is $lca_i(u, v)$ and thus no update message has to be sent. This leads to Eq. 3. $SC(v)$ can be formulated as

$$SC(v) = \sum_{u \in nei(v)} \left(\sum_{i=2}^{m} \left((i-1) \times \sum_{\substack{(s,t) \in E_G \wedge \\ f(s,t,v,u) = i}} w_G(s, t)\right)\right), \quad (4)$$

where $nei(v)$ denotes the neighbors of $v$ in $G$ and $f(s,t,v,u)$ represents the number of trees, each of which, say $T_{\sigma_j}$, makes the following statement true ($u = p_j(v)) \wedge ((s,t) \neq (v,u)) \wedge ((S_v(T_{\sigma_j},s) \wedge \neg S_v(T_{\sigma_j},t)) \vee (S_v(T_{\sigma_j},t) \wedge \neg S_v(T_{\sigma_j},s)))$. Intuitively, this means that when an object moves from $s$ to $t$ or from $t$ to $s$, $v$ will send an update message to $u$ for updating tree $T_{\sigma_j}$ and a single update message can update $i$ trees simultaneously; therefore, $(i-1)$ messages are saved. This leads to Eq. 4. The result will be used later to construct multiple low-cost trees.

### 3.2 The Location Query Mechanism

Now, we describe our location query mechanism. We assume that a user can issue a query from any sensor. When a sensor $x$ receives a query for object $o$, there are two scenarios: (1) $o$ does not appear in any of the entries of $DL_x$, and (2) $o$ appears at least in one of the entries of $DL_x$.

In the first scenario, $x$ will forward the query to the closest sink, say $\sigma_j$, in order to inquire $o$'s location. The reason for doing so is that, for each sink $\sigma_i$, it can be easily shown that all objects tracked by the network will be contained in $DL_{\sigma_i}$. However, on the query's way to sink $\sigma_j$, if an intermediate node $y$ finds that $o$ appears in $DL_y$, then the second scenario will be initiated immediately.

In the second scenario, we will show how $x$ can forward the query to locate $o$. We can model the WSN responsible for tracking object $o$ as a directed *query graph* $G_o^{'} = (V_G, E_{G_o^{'}})$, where a directed edge $(u,v) \in E_{G_o^{'}}$ if and only if $o \in DL_u(v)$. Our location update mechanism guarantees that if $x$ forwards the query along the query graph $G_o^{'}$, then $o$ is always reachable. For example, Fig. 4(a) shows the query graph $G_{Car1}^{'}$ of Fig. 3(a) for $Car1$, where $A$ and $B$ are sinks. It means that $x$ can simply forward the query to any $y$ such that $o \in DL_x(y)$. This is repeated until a sensor $z$ such that $o \in DL_z(z)$ is reached. However, the fact that $o$ is reachable via $y$ from $x$ in $G_o^{'}$ does not necessarily imply that $G_o^{'}$ is cycle-free when multiple trees coexist in the network. For example, Fig. 4(b) shows two trees $T_A$ and $T_B$ and Fig. 4(c) shows the query graph for $Car1$, which have a cycle containing $D$, $F$, and $G$. A query forwarded as above may loop infinitely.

A simple way to solve the infinite loop problem is to force a query to always travel along a designated tree. In order to achieve this, we can add a field $tree\_index$ to the query request. Once the $tree\_index$ is set by a certain sensor, the following sensors can follow the tree designated by $tree\_index$. Here, we propose an alternative solution which imposes that all trees be shortest-path trees. If so, not only the query and update paths can be shortest, but also the corresponding $G_o^{'}$ for each object $o$ is always cycle-free. Thus, our query mechanism will work correctly.

**Theorem 1.** *If all trees are shortest-path trees, the query graph $G_o^{'}$ for each object $o$ tracked by the network must be cycle-free.*

*Proof.* Without loss of generality, we assume $o$ is tracked by sensor $x$ currently. For the purpose of contradiction, we assume that all trees are shortest-path trees but a cycle $< c_0, c_1, ..., c_k, c_0 >$ exists in $G_o^{'}$. Let $c_j$ be the vertex in the cycle with minimum $dist_G(x, c_j)$. The fact that $(c_j, c_{j+1})$ is an edge in the cycle implies that $o \in DL_{c_j}(c_{j+1})$. This means that there exists a tree, say $T_{\sigma_i}$ that contains the edge $(c_j, c_{j+1})$, which can lead to $x$. Because $dist_G(x, c_{j+1}) \geq dist_G(x, c_j)$, $T_{\sigma_i}$ must not be a shortest-path tree. This contradicts our assumption that all trees are shortest-path trees. Therefore, $G_o^{'}$ must not contain a cycle. □

After the query reaches the sensor currently tracking the queried object, the sensor can reply to the sensor initiating the query through a shortest path. In the case that the user is capable of mobility, the user should update with the initiating sensor its position until a reply is received. This would solve the mobility problem.

## 4 Multi-Tree Construction Algorithms

The above derivations have suggested that trees rooted at sinks should be shortest-path trees to avoid the cycle problem. In addition, following the derivation of Eq. 2, these trees should be constructed carefully to reduce communication costs. Below, we propose two distributed multi-tree construction algorithms, given $\sigma_1$, $\sigma_2$, ..., $\sigma_m$ as the sinks.

### 4.1 The MT-HW Algorithm

From Eq. 3, we observe that when an edge $(u,v)$ becomes an edge of $T_{\sigma_i}$, the events occurring on $(u,v)$ do not cause any message to be reported for updating $T_{\sigma_i}$. Therefore, in MT-HW (multi-tree construction with the high-weight-first property) algorithm, an edge $(u,v)$ with higher weight will be considered for being included into a tree earlier.

First, we define the term *candidate parents*. A sensor $y$ is called a candidate parent of $x$ for sink $\sigma_i$, if $y$ is $x$'s neighbor and $dist_G(\sigma_i, x) = dist_G(\sigma_i, y) + 1$. We assume that when the network is initiated, each sink $\sigma_i$ will flood a message in the network, which helps each sensor $x$ to derive $dist_G(\sigma_i, x)$ and thus $x$'s candidate parents. The MT-HW algorithm works as follows. Each sensor $x$ will sort its neighbors in a decreasing order according to the event rates between it and its neighbors. Then, for each sink $\sigma_i$, $x$ will pick one neighbor $y$ as its parent that has the highest event rate among $x$'s candidate parents for $\sigma_i$ and set $y = p_i(x)$.

**Theorem 2.** *If $G$ is connected, the trees constructed by the MT-HW algorithm must be connected shortest-path trees.*

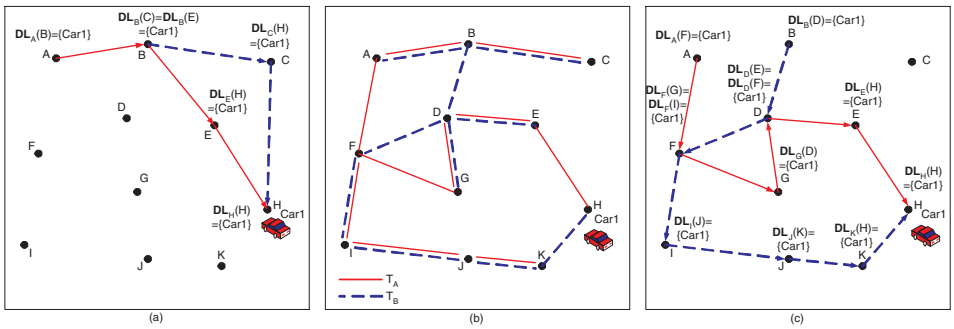


**Figure 4. (a) The query graph $G'_{Car1}$ of Fig. 3(a) for $Car1$. (b) Another example of a two-tree system. (c) The query graph of (b) for $Car1$, which contains a cycle.**

*Proof.* Since $G$ is connected, for each $T_{\sigma_i}$, a sensor $x$ ($x \neq \sigma_i$) can always find one candidate parent as its parent in $T_{\sigma_i}$. Thus, $T_{\sigma_i}$ will be a connected tree. Now, we further show that $T_{\sigma_i}$ will be a shortest-path tree. By the definition of the candidate parent, the parent must be closer to $\sigma_i$ than the node itself. Therefore, all $T_{\sigma_i}$ are shortest-path trees. □

### 4.2 The MT-EO Algorithm

From Eq. 4, we observe that if we can increase the number of the tree edges that overlap with each other, the value of $SC(v)$ may increase and $U$ can be reduced. The MT-EO (multi-tree construction with the edge-overlap-first property) algorithm is designed to increase the level of the overlap among tree edges.

As the MT-HW algorithm, each sensor $x$ will determine all candidate parents for each sink $\sigma_i$. Each of $x$'s neighbors is associated with an *overlap counter* for $x$. The counter is increased by one whenever a neighbor of $x$ is considered as a candidate parent for a sink. Then, $x$ selects the neighbor, say $y$, whose overlap counter is the largest. For each sink $\sigma_i$ where $y$ is a candidate parent of $x$, we set $y = p_i(x)$ for $T_{\sigma_i}$. Then, the overlap counters of all $x$'s neighbors are recomputed for those sinks for which $x$ has not yet determined its parents. Again, the neighbor $y$ whose overlap counter is the largest is selected as $x$'s parent for the corresponding sinks. This procedure is repeated until $x$ has determined its parents for all sinks.

**Theorem 3.** *If $G$ is connected, the trees constructed by the MT-EO algorithm must be connected shortest-path trees.*

*Proof.* The proof is similar to that of Theorem 2. The theorem holds because a non-sink node can always find a parent that is closer to the sink. □

In fact, we can easily combine the MT-HW algorithm with the MT-EO algorithm and vice versa. Whenever there is a tie (either the same event rate or the same overlap counter value), the other algorithm can be used.

## 5 Simulation Results

We have simulated a sensing field of size $256 \times 256$. 1024 sensors are deployed in the sensing field. Two deployment models are considered. In the first one, sensors are regularly deployed as a $32 \times 32$ grid-like network. In the second model, sensors are randomly deployed. In both models, sinks are determined by uniformly partitioning the sensing field into equal-size grids according to the number of sinks required and choosing the sensor that is nearest to the center of the grid as the sink. Further, the event rates of links are generated based on the *modified city mobility model* [11]. Queries may be generated from any sensor. The query rate is defined as the number of queries generated in the network per unit time.

We compare our schemes with a naive scheme and the $m$-DAT scheme. In the naive scheme, any update is sent to all sinks immediately (i.e., there is no in-network processing capability). Specifically, when an object moves to a new sensor, a multicast spanning tree formed from the new location of the object to all sink and the update message containing the up-to-date location information of the object is sent to all sinks like a multicast. Thus, any query only needs to be sent to its nearest sink. In the $m$-DAT scheme, we run the DAT algorithm [11] $m$ times for $m$ sinks to construct $m$ trees. Then the update and query mechanisms proposed in Sec. 3 are used on these $m$ trees. Note that if each tree performs the update independently, then the update cost will become approximately $m$ times when the number of trees increases from 1 to $m$. Thus, in the $m$-DAT scheme, the update and query mechanisms proposed in this paper are adopted to reduce the update cost. We comment that the DAT algorithm is a centralized algorithm and is similar to the MT-HW algorithm in that it also takes the event rates of

links into consideration. However, these $m$ trees are constructed independently.

First, we observe the advantage of using data aggregation to reduce update cost. Fig. 5 shows the results. In the naive scheme with regular sensor deployment, when the number of trees increases by a factor of $N$, the update cost will increase by a factor of $\sqrt{N}$. On the contrary, in MT-HW and MT-EO algorithms, the update cost only increases by factors of 17.47 and 10.06 respectively, when the number of trees increases from 1 to 1024. This demonstrates the effectiveness of data aggregation in a multi-sink sensor network. In addition, we will show later that the increased update cost can be compensated by lower query cost when multiple sinks coexist in the network. We can also observe from Fig. 5 that MT-EO has better performance than MT-HW and *m*-DAT have. This implies that considering the overlap of tree edges is more important. Since *m*-DAT performs similarly to MT-HW, we will ignore it in the subsequent discussion. Moreover, the trend of the performance in random deployment model is similar to that in regular deployment model; thus, we also ignore the performance in random deployment model in the following discussion.

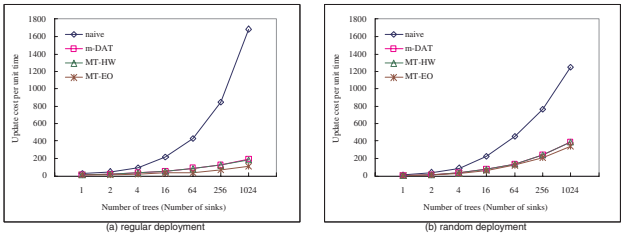


**Figure 5. Comparison of update costs.**

Next we investigate the total (update plus query) cost under different query rates. Fig. 6 shows the results. The proposed algorithm for in-network update and query keeps its advantage until the query rate is larger than 9 and 40 when the numbers of sinks are 16 and 1024 respectively. One should note that in these experiments only about 1.75 events are generated in the network per unit time. This means that the naive scheme is better only when the query rate is relatively much higher than the object movement rate. We believe our proposed method can be applied to most applications.

Last, we focus on comparing a multi-sink system with a single-sink system. A multi-sink system can benefit from lower query cost. Fig. 7 shows the message costs under different query rates and numbers of sinks. Note that when the number of sinks is one, it is a single-sink system. Only when there is no query (rate=0), the single-sink system has the advantage of lower update cost. When the query rate is median (rate=5) or high (rate=10 or 15), we do find some numbers of sinks such that the multi-sink system performs

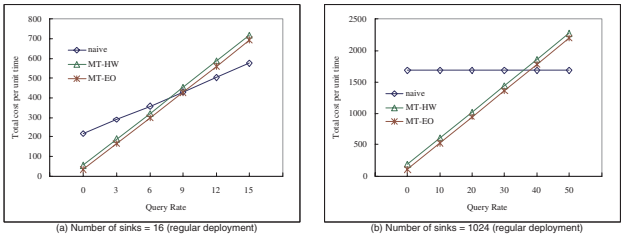


**Figure 6. Comparison of total costs under different query rates.**

better. In fact, as the query rate is higher, the advantage is more significant. The figure can also be used as a reference to choose the best number of sinks under different scenarios. For example, in MT-EO, $m = 2$ has the best performance when the query rate is 5.0 and $m = 64$ has the best performance when the query rate is 15.0. Two implicit results should also be addressed. First, a multi-sink system has a faster query response time. To verify this, Fig. 8 shows the average hops per query under different numbers of sinks. As we can see, when the number of sinks increases, the average hops per query is reduced. Second, a multi-sink system can achieve a better load balance factor. To verify this, Fig. 9 shows the standard deviation of the number of packets transmitted by each sensor. As we can see in (b), (c) and (d), when the number of sinks increases, the stand deviation is reduced. This is because queries dispersed to multiple sinks rather than a single sink.

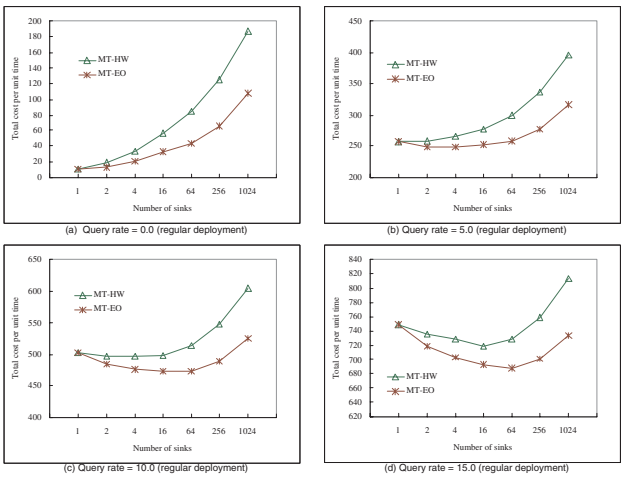


**Figure 7. Comparison of total costs.**

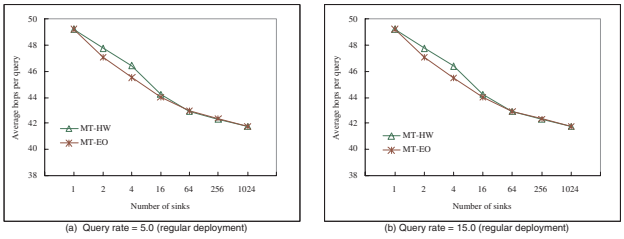


**Figure 8. Comparison of average hops per query under different numbers of sinks.**

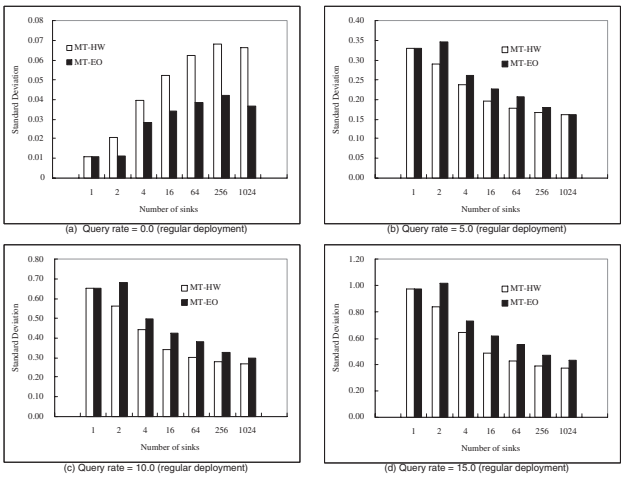


**Figure 9. The standard deviation of the number of packets transmitted by each sensor.**

## 6 Conclusions

In this paper, we have proposed an in-network update and query algorithm for a multi-sink WSN. This algorithm strikes the tradeoff between the update and query costs. Having multiple sinks is important when the network scale is large or when the query rate is high. The corresponding update cost is formulated formally. Based on the formulation, we have presented two distributed algorithms to construct multiple trees. We have verifies the benefits of a multi-sink WSN from different aspects, including the total (update plus query) cost, the number of sinks, query response time, and load balance factor.

## Acknowledgment

The authors would like to thank Prof. Wen-Chih Peng for helpful conversations about this work. Y. C. Tseng's research is co-sponsored by NSC under grant number 93-2752-E-007-001-PAE, by MOEA under grant number 94-EC-17-A-04-S1-044, by ITRI, Taiwan, and by Intel Inc.

## References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[2] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with binary sensors. In *Proc. of ACM SenSys*. ACM Press, Nov. 2003.

[3] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, Sept. 1991.

[4] S. Blackman and R. Popoli. *Design and Analysis of Modern Tracking Systems*. Artech House, 1999.

[5] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, 2004.

[6] W.-P. Chen, J. C. Hou, and L. Sha. Dynamic clustering for acoustic target tracking in wireless sensor networks. In *Proc. of IEEE Int'l Conf. on Network Protocols (ICNP)*, Nov. 2003.

[7] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, Oct. 2001.

[8] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *ACM Int'l Conf. on Mobile Computing and Networking (MobiCom)*, Aug. 2000.

[9] H. T. Kung and D. Vlah. Efficient location tracking using sensor networks. In *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, Mar. 2003.

[10] D. Li, K. Wong, Y. Hu, and A. Sayeed. Detection, classification, and tracking of targets. *IEEE Signal Processing Magazine*, 19(2):17–29, Mar. 2002.

[11] C.-Y. Lin and Y.-C. Tseng. Structures for in-network moving object tracking in wireless sensor networks. In *IEEE Int'l Conf. on Broadband Networks (Broadnets)*, pages 718–727, Oct. 2004.

[12] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, Sept. 2002.

[13] K. Mechitov, S. Sundresh, and Y. Kwon. Cooperative tracking with binary-detection sensor networks. *University of Illinois at Urbana-Champaign, Technical Report UIUCDCS-R-2003-2379*, 2003.

[14] E. Shih, S.-H. Cho, N. Ickes, R. Min, A. Shnha, A. Wang, and A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *ACM Int'l Conf. on Mobile Computing and Networking (MobiCom)*, pages 272–287, July 2001.

[15] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proc. of IEEE Infocom*, pages 1567–1576, June 2002.