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

Chih-Yu Lin, Yu-Chee Tseng*, Wen-Chih Peng
Department of Computer Science,
National Chiao-Tung University, Hsin-Chu, 300, Taiwan
E-mail: lincyu@csie.nctu.edu.tw
E-mail: yctseng@csie.nctu.edu.tw
E-mail: wcpeng@csie.nctu.edu.tw
*Corresponding author

Ten H. Lai
Department of Computer Science and Engineering,
The Ohio State University, Columbus, Ohio 43210 USA
E-mail: lai@cse.ohio-state.edu

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 cel-
lar 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. The proposed location management scheme consists of two parts.
First, a message-efficient algorithm that describes how to perform location updates and
queries is proposed. Then, 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.

Reference to this paper should be made as follows: C.-Y. Lin, Y.-C. Tseng, T.-
in a Multi-sink Wireless Sensor Network’, Int. J. High Performance Computing and
Networking, Vol. 1, Nos. 1/2/3, pp.64–74.

Biographical notes: Chih-Yu Lin is a currently a Ph.D student. His research
interests include mobile computing and sensor networks. He is a student member of
IEEE.

Yu-Chee Tseng received a PhD in Computer and Information Science from the
Ohio State University in January 1994. He is currently a Professor and the Chairman
in the Department of Computer Science, National Chiao Tung University. His research
interests include mobile computing, wireless communication, network security and
parallel and distributed computing. He is a member of ACM and a Senior Member of
IEEE.

T.-H. Lai is currently a Professor in the Department of Computer Science and
Engineering, the Ohio State University. His research interests include distributed
computing and wireless networks.

Wen-Chih Peng is currently an assistant professor in the department of com-
puter science at the National Chiao Tung University. His research interests include
mobile data management, sensor data management, and data mining. He is a member
of IEEE.
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 design of physical and medium access layers (Shih et al., 2001; Ye et al., 2002) and routing and transport protocols (Ganesan et al., 2001; Intanagonwiwat et al., 2000). Applications of WSNs have been studied in Akyildiz et al. (2002), Burrell et al. (2004), and Mainwaring et al. (2002).

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 (Aslam et al., 2003; Blackman and Popoli, 1999; Li et al., 2002; Mechitov et al., 2003). 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 (Kung and Vlah, 2003; Lin and Tseng, 2004). 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 Kung and Vlah (2003) and Lin and Tseng (2004). In Kung and Vlah (2003), 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. Lin and Tseng (2004) 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 Lin and Tseng (2004), it is assumed that there is only one sink in the network. In this paper, we explore the possibility of having multiple sinks in the network. One advantage of having multiple sinks is to reduce the response time of queries. 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 order to support location management in a multi-sink WSN, we can extend the tree structure used in the single-sink system (Lin and Tseng, 2004) 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, by exploring the concept of data aggregation, we propose an algorithm to efficiently update multiple trees. With proper design, we show that the update cost increases only slightly when the number of trees (i.e., the number of sinks) increases from 1 to 64. 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. Experimental results show that the increased update cost with multiple trees can be compensated by lower query cost and the query cost also depends on m, the number of sinks. This allows us to further investigate how to choose the value of m under different scenarios.

A significant amount of research effort has been elaborated upon issues of object tracking problems. The authors in Xu and Lee (2003) explored a localized prediction approach for power efficient object tracking by putting unnecessary sensors in sleep mode. Techniques for cooperative tracking by multiple sensors have been addressed in Aslam et al. (2003), Chen et al. (2003), Mechitov et al. (2003), and Zhang and Cao (2004). In Chen et al. (2003), a dynamic clustering architecture that exploits signal strength observed by sensors is proposed to identify the set of sensors to track an object. In Zhang and Cao (2004), a convoy tree is proposed for object tracking using data aggregation to reduce energy consumption. To the best of our knowledge, prior works neither address the problem of tracking objects in a multi-sink environment, nor explore maintaining multiple trees to achieve location management. These features distinguish this paper from others.

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. This paper concludes with Sec. 6.
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 Chen et al., 2003 and we omit the details). Therefore, the sensing field can be modelled by a Voronoi graph (Aurenhammer, 1991), 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).

2.2 From Single-sink to Multi-sink WSNs

In Lin and Tseng (2004), 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 \( C \), update messages will be forwarded from \( H \) to \( B \) and from \( C \) to \( B \) respectively. 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 \( E_{T_x} \subseteq E_G \). For example, Fig. 3(a) extends the network in Fig. 2 such that both sensors \( A \) and \( B \) are sinks. Three issues should 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 inde-
update and query mechanisms in this paper. Moreover, each sensor sink \( \sigma_i \) by \( \sigma \)

\( DL \)

\( U \)

\( a \)

\( e \)

\( (C_a \), \( G \), \( I \))

\( Car_3 \)

\( 3.2. \)

\( Multi-tree \) construction: Our 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 through simulation). Because both the update cost and the query cost are affected by the number of trees used, we will investigate the proper value of \( m \) under various scenarios.

\( 3 \) Update and Query Mechanisms

\( 3.1 \) Notations and Data Structures

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 \( G \). Table 1 summaries the notations used in this paper. Then, we introduce the data structures used in this paper. Moreover, each sensor \( x \) will keep two tables in order to process updates and queries:

- **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., \( DL_x(y) = \{ o \mid \exists z, i \) s.t. \( o \in DL_z(z), S_y(T_{\sigma_i}, z) = 1, \) 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 \( A \).

\( Figure \) 3: (a) The \( DLs \) 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 \( C \).

| \( dist_G(u, v) \) | 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 \( v \). |
|\( lca_{T_{\sigma_i}}(u, v) \) | The lowest common ancestor of \( u \) and \( v \) in \( T_{\sigma_i} \). |
|\( p_{\tau_i}(v) \) | The parent of \( v \) in \( T_{\sigma_i} \). |
|\( \sigma_i \) | The root of \( T_{\sigma_i} \). |

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, some sensors' Detected_Lists have to be modified accordingly.
3.2 The Location Update Mechanism

The goal of location update is to ensure that the DetectedLists 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 DetectedLists 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, node y should update its DetectedList 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 DetectedList. Furthermore, 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. 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, node x 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):

$$\exists i((S_x(T_{\sigma_i}, a) = 0 \land S_x(T_{\sigma_i}, b) = 1) \lor (S_x(T_{\sigma_i}, a) = 1 \land S_x(T_{\sigma_i}, b) = 0))$$  \hspace{1cm} (1)

(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 y checking Eq. 1. If $x$ receives the update message for the first time and there is a tree $T_{\sigma_j}$, making Eq. 1 true, then an update message should be sent to $p_i(x)$. However, 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 update aggregation.

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

- For sensor a, it will remove o from DL$_a$(a) and check whether there exists a tree $T_{\sigma_j}$ and a neighbor y such that $S_a(T_{\sigma_j}, b) = 1$ and $a = p_i(y)$. If the answer is affirmative, this implies that a can find o by asking y. Thus, it adds o into DL$_a$(y).
- For any other sensor x that receives the update message from y, if $\exists i(S_x(T_{\sigma_i}, b) = 1 \land x = p_i(y))$ 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 DLs and the reported messages are also shown in Fig. 3(b). Our update scheme ensures that when an object o moves from one sensor to another, if no packet loss happens and the update procedure can be completed before o moves to another sensor, then the freshness of DetectedLists of sensors can be guaranteed.

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

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

$$U(T_{\sigma_i}) = \sum_{(u,v) \in G} (w_G(u,v) \times (dist_{T_{\sigma_i}}(u, lca_i(u,v)) + dist_{T_{\sigma_i}}(v, lca_i(u,v)))),$$ \hspace{1cm} (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 Lin and Tseng (2004), except that when (u, v) $\in$ $\mathcal{E}_{T_{\sigma_i}}$, 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. The formulation of $SC(v)$ depends on the forwarding schemes. Two forwarding schemes are considered: the broadcast scheme and the unicast scheme. Due to the broadcast nature of wireless radio, when a sensor sends an update message, we assume all its neighbors will receive the update message in the broadcast scheme. In this case, $SC(v)$ can be formulated as

$$SC(v) = \sum_{i=2}^{m} \left( i - 1 \right) \times \sum_{(s,t) \in G^i \land f(s,t) \in \mathcal{F}_{T_{\sigma_i}}} w_G(s,t),$$ \hspace{1cm} (4)

where $f(s,t,v)$ represents the number of trees, each of which, say $T_{\sigma_j}$, makes the following statement true $((s,t) \neq (v, p_j(v))) \land (\neg S_v(T_{\sigma_j}, s) \land S_v(T_{\sigma_j}, t) \land \neg S_v(T_{\sigma_j}, s))$. Intuitively, this means that when an object moves from s to t or from t to s, v will
broadcast an update message to its neighbors for updating tree $T_x$. And this broadcast message can update these $i = f(s, t, v)$ trees simultaneously; therefore, $(i - 1)$ messages are saved. This leads to Eq. 4.

However, the packet transmission is unreliable in the broadcast scheme. Once the update messages are lost during the transmission, Detected Lists may not contain up-to-date information, resulting in the failures of queries. Thus, one also can adopt the unicast scheme to forward update messages in which each update message has a designated destination. In this case, $SC(v)$ can be formulated as

$$SC(v) = \sum_{u \in \text{nei}(v)} \left( \sum_{i=2}^{m} (i - 1) \times \sum_{(x, t) \in E_G} w_G(s, t) \right),$$

where $\text{nei}(v)$ denotes the neighbors of $v$ in $G$ and $g(s, t, v, w)$ represents the number of trees, each of which, say $T_x$, makes the following statement true $(u = p_j(v)) \land \exists t_0(s, t, v, u) \land \text{nei}(v)$. Eq. 3, Eq. 4 and Eq. 5 will give us hints for constructing message-efficient multiple virtual trees.

### 3.3 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 $o \in DL_{\sigma_i}(v)$. 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 $G'_{\sigma_1}$ of Fig. 3(a) for $\sigma_1$, 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 Car 1, 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 $\text{tree}_i$ to the query request. Once the $\text{tree}_i$ is set by a certain sensor, the following sensors can follow the tree designated by $\text{tree}_i$. 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 $< c_0, c_1, \ldots, c_k, c_0 >$ exists in $G'_o$. Let $c_j$ be the vertex in the cycle with minimum $\text{dist}_G(x, c_j)$. The fact that $(c_j, c_{j+1})$ is an edge in the cycle implies that $o \in DL_{\sigma_i}(c_{j+1})$. 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 $\text{dist}_G(x, c_{j+1}) > \text{dist}_G(x, c_j)$, $T_{\sigma_i}$ 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 $\sigma_1, \sigma_2, \ldots, \sigma_m$ 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 \( \text{dist}_G(\sigma_i, x) = \text{dist}_G(\sigma_i, y) + 1 \). We assume that when the network is initialized, each sink \( \sigma_i \) will flood a message in the network, which helps each sensor \( x \) to derive \( \text{dist}_G(\sigma_i, x) \) 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.

**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 and Eq. 5, 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 \( 256 \times 256 \), where 1024 sensors are deployed in the sensing field. Two deployment models are considered. In the regular deployment model, sensors are regularly deployed as a \( 32 \times 32 \) grid-like network. In the random deployment 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 given and choosing the sensor that is the 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 (Lin and Tseng, 2004). Queries could be issued 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 other two schemes called QF and MC respectively. In the QF scheme, no update message will be sent. When a user intends to query an object’s location, the query message will be flooded in the network. In the MC scheme, when an object moves to a new sensor, a multicast spanning tree will be formed from the new location of the object to all sinks and the update message containing the up-to-date...
location information of the object is sent to all sinks. In this scheme, any query only needs to be sent to its nearest sink. Based on the tree construction algorithms and the forwarding schemes, four schemes proposed by us are compared with the QF and the MC schemes. Specifically, in the HW-B scheme, the MT-HW algorithm and the broadcast forwarding scheme are used. In the HW-U scheme, the MT-HW algorithm and the unicast forwarding scheme are used. In the EO-B scheme, the MT-EO algorithm and the broadcast forwarding scheme are used. Finally, in the EO-U scheme, the MT-EO algorithm and the unicast forwarding scheme are used.

As mentioned above, when an object moves from one sensor to another, if no packet loss arises and the update procedure can be completed within a period during which the object does not move again, our proposed update mechanism can ensure that the Detected Lists of sensors are fresh. However, packet loss is a common phenomenon in a wireless network and transmission delay should also be taken into consideration. In order to investigate the impact of packet loss, we develop an event-oriented simulator using C language in which the unslotted CSMA defined in IEEE 802.15.4 (IEEE Std 802.15.4, 2003) is implemented. Because we observe that the collision phenomenon is very severe, we assume that a node has to wait 10 ~ 60 milliseconds to start a new transmission after it successfully transmits a packet in order to avoid multiple sensors transmit packets at the same time. Finally, we assume each sensor’s sending buffer is limited such that for a sensor, if there are too many packets to be sent simultaneously, some of packets will be discarded. The related parameters are shown in Table 2.

### Table 2: Parameters used in our simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Size</td>
<td>10</td>
</tr>
<tr>
<td>The length of DATA</td>
<td>30 Bytes</td>
</tr>
<tr>
<td>The length of ACK</td>
<td>17 Bytes</td>
</tr>
<tr>
<td>Bit rate</td>
<td>250 kbps</td>
</tr>
<tr>
<td>Symbol rate</td>
<td>62.5 ksymbol/s</td>
</tr>
<tr>
<td>aUnitBackoffPeriod</td>
<td>20 symbols</td>
</tr>
<tr>
<td>aTurnaroundTime</td>
<td>12 symbols</td>
</tr>
<tr>
<td>macMinBE</td>
<td>3</td>
</tr>
<tr>
<td>aMaxBE</td>
<td>5</td>
</tr>
<tr>
<td>macMaxCSMABackoffs</td>
<td>4</td>
</tr>
<tr>
<td>The maximum number of retransmission</td>
<td>5</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>1 hour</td>
</tr>
<tr>
<td>Number of Objects</td>
<td>128</td>
</tr>
</tbody>
</table>

5.1 Impact of Objects’ Speeds

First, we consider the scenario in which the update cost dominates the overall communication cost. To achieve this, we compare all schemes under various objects’ speeds. Higher the speed is, more events are generated; thus, the update cost will dominate the performance. In Fig. 5, sensors are deployed regularly and four sinks are deployed. The query rate is set to be 1 query/second in this experiment. Fig. 5(a) shows the communication cost (i.e., the number of packets transmitted in the network) of these schemes with the value of object speed varied. As can be seen in Fig. 5(a), the update cost is constant in the QF scheme because no update packet has to be sent. The update costs of all other schemes will grow when the speed becomes higher since more update packets have to be sent. The update cost of the MC scheme grows enormously, because no in-network processing technique is applied. Our proposed schemes outperform the QF scheme and the MC scheme when the speed is lower than 10 units/second. Since the sensing radius of a sensor is 4 units, 10 units/second is relatively high. We further give an insight into our proposed scheme. Obviously, the broadcast forwarding scheme has lower update cost than the unicast scheme has. However, as can been seen later, the unicast scheme has higher query success rate than the broadcast scheme has. Besides, we can see that the MT-EO scheme outperforms the MT-HW scheme slightly, because more packets are saved due to the overlap of tree edges.

Fig. 5(b) shows the query response time of these schemes, where the query response time is defined as the time elapsed between the time at which the query issued and the time at which the query result returned. The MC scheme is the best because any query only has to be forwarded to the sink. Our proposed schemes are slightly worse than the QF scheme because two phases are required in our schemes. Although the MC scheme has the best performance in terms of query response time, the query result may not be the most up-to-date one. This problem becomes further severe when packet loss happens. A measurement, query error, is defined as the number of hops between the real location of the object and the location carried by the query reply at the time at which the query is returned to the user. In Fig. 5(c), it can be seen that the MC scheme suffers from higher query errors. Finally, Fig. 5(d) shows the query success rates under different schemes. Note that a query may fail due to packet collision, packet loss, buffer overflow and contaminated Detected Lists. More packets transmitted in the network usually means more collision. Thus, our proposed scheme and the MC scheme perform worse than the QF scheme does eventually, but all schemes have similar performance under reasonable speed. Note that the broadcast forwarding scheme has the worst performance due to the contaminated Detected List problem; however, the unicast forwarding scheme can be used to solve this problem.

Since the number of sinks is an important issue in this paper, the scenario used in Fig. 5 is applied again in Fig. 6 except that 256 sinks are deployed now. It is observed that if the number of sinks is large, a considerable amount of update messages will be generated. Thus, when the update cost dominates the communication cost, using less sinks is better. Finally, experiments with the random deployment model is investigated in Fig. 7, where the number of sinks is 4. We can see that the success rates under
Figure 5: Performance study with objects’ speeds varied, where sensors are deployed regularly and four sinks are deployed.

Figure 6: Performance study with objects’ speeds varied, where sensors are deployed regularly and 256 sinks are deployed.
the random deployment model are lower than that under the regular deployment model, because the collision phenomenon is very severe in the random deployment model. When a node has many neighbors, this node usually suffers severe collision due to the contention and the hidden terminal problem. Therefore, we further compute the average number of neighbors of a sensor. The average numbers of neighbors of a sensor under the regular deployment model and the random deployment model are 3.875 and 5.666 respectively. Thus, we conjecture that the severe collision phenomenon in the random deployment model is caused by the hidden terminal problem and the higher contention between sensors. We further give an insight into our proposed scheme. We can find that the performance of the unicast forwarding scheme is very bad due to the buffer overflow problem. The reason can be explained as follows: when an event occurs, there are averagely 5.666 update packets will be injected into the sending buffer and the length of sending buffer is 10 only. Thus, the length of the sending buffer should be designed carefully. Other most observations made under the regular deployment model could be applied to the random deployment model. In the following experiments, we only show the results under the regular deployment model.

5.2 Impact of Query Rates

Now we consider the scenario in which the query cost dominates the overall communication cost. To achieve this, we compare all schemes by adjusting query rates. When the query rate is high, the query cost will dominate the performance. The object’s speed is set to be 1 unit/second in this experiment. 4 and 256 sinks are deployed in Fig. 8 and Fig. 9 respectively. First, we compare the communication costs under different schemes. As shown in Fig. 8(a) and Fig. 9(a), the QF scheme is the worst one, because queries are disseminated by flooding. On the contrary, in our proposed schemes, queries are disseminated by unicasting. Thus, our proposed schemes have the best performance. We can further observe that when the number of sinks increases from 4 to 256, the communication cost of the MC scheme also grows due to higher update costs. However, our proposed schemes can achieve almost the same cost when the number of sinks increases. This is because using multiple sinks can reduce the query cost by a shorter query path and the saved query cost can be used to compensate the increased update cost. Thus, the advantage of using multiple sinks can be achieved when the query cost dominates the performance. In addition, when the number of sinks increases (i.e., from 4 in Fig. 8 to 256 in Fig. 9), it can be seen that the query response time of our proposed schemes in Fig. 9(b) is slightly smaller than that in Fig. 8(b) due to shorter query paths. As shown in Fig. 8(c) and Fig. 9(c), although the MC scheme is the best one in terms of query response time, it is the worst one in terms of query error. Finally, in Fig. 8(d) and Fig. 9(d), we can see that the QF scheme is the worst one in terms of success rate, because of the collision incurred by the flooding.

5.3 Impact of the Number of Sinks

From the previous experimental results, it can be seen that when the query cost dominates the communication cost, using multiple sinks can achieve better performance. Thus, we further investigate the impact of the number of sinks on the performance. The query rate is set to be 10 queries/second and the objects’ speed is set to be 0.333 and 0.111 respectively. In Fig. 10(a) and Fig. 11(a), it can be seen that the communication costs almost do not increase when the number of sinks increases, because the increased update cost can be compensated by lower query cost. As can be seen in Fig. 10(b) and Fig. 11(b), using multiple sinks can reduce the query response time slightly due to shorter query paths. Fig. 10(c) and Fig. 11(c) show the values of the standard deviation of the number of packets transmitted by each sensor. It is observed that when the number of sinks increases, the values of the standard deviation are reduced. This is because queries are dispersed to multiple sinks rather than a single sink. Thus, load balance can be achieved by using multiple sinks. Finally, in Fig. 10(d) and Fig. 11(d), it can be seen that using multiple sinks is able to increase the success rate, because shorter query paths could result in less collision.

5.4 Multi-Sink Systems with Partial Storage

As mentioned above, using multiple trees will increase the update cost. A simple way to reduce the update cost while achieving the advantage of load balance at the same time is to explore the partial storage technique. The partial storage technique is motivated by GHT (Ratnasamy et al., 2002). The basic idea is that each object’s location will be stored in only some of the sinks. In our simulation, the partial storage technique is implemented as follows.

First, we evenly divide the sensing field into m zones, each of which has an unique ID. For each zone, the sensor closest to the center of the zone is designated as the sink. Then, each object is hashed into l zones, where l (< m) is a predefined number, and an object only needs to update its location with the sinks of these l zones.

Now, we demonstrate the benefit of the partial storage technique by simulation. The query rate is set to be 2 queries/second and the objects’ speed is set to be 1 unit/second. We compare the EO-B and the EO-U schemes with α sinks against the EO-B-PS and the EO-U-PS schemes (which mean the EO-B and the EO-U schemes extended with the partial storage technique) with 1024 zones and α hashed zones per object. Fig. 12 shows the results with the value of α varied. It can be observed that, although the communication costs of the EO-B-PS scheme and the EO-U-PS scheme are higher, the values of the standard deviation of the numbers of packets transmitted by each sensor are lower. Thus, using the partial storage technique can achieve better load balance.
Figure 7: Performance study with objects’ speeds varied, where sensors are deployed randomly and four sinks are deployed.

Figure 8: Performance study with query rates varied, where sensors are deployed regularly and four sinks are deployed.
Figure 9: Performance study with query rates varied, where sensors are deployed regularly and 256 sinks are deployed.

Figure 10: Performance study with the number of sinks varied, where the objects’ speed is set to be 0.333 unit/second.
Figure 11: Performance study with the number of sinks varied, where the objects’ speed is set to be 0.111 unit/second.

Figure 12: The performance of the partial storage technique.
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 verified the benefits of using multiple sinks in a WSN from different aspects, including the total (update plus query) cost, the number of sinks, query response time, query success rate, and load balance factor. Our simulation results show that when the query cost dominates the communication cost, using multiple sinks can achieve better performance. In addition, through the usage of multiple sinks, loads of sensors are easily balanced.

Acknowledgment


REFERENCES


IEEE Std 802.15.4 (2003). IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs).


