Advance Access publication on May 10, 2009
doi:10.1093/comjnl/bxp043

Imprecision-Tolerant Location Management for Object-Tracking Wireless Sensor Network

Chih-Yu Lin[1],*, Yu-Chee Tseng[2] and Yung-Chih Liu[3]

[1]*Department of Information Science and Applications, Asia University, Taichung, Taiwan, R.O.C.*
[2]*Department of Computer Science, National Chiao Tung University, Hsin-Chu, Taiwan, R.O.C.*
[3]*Networks and Multimedia Institute, Institute for Information Industry, Taipei, Taiwan, R.O.C.*
**Corresponding author: lincyu@asia.edu.tw*

An important issue of wireless sensor networks is object tracking, where the key steps include event detection, target classification, location estimation and location management. The main theme of this paper is location management. Because imprecision is an inherent property in object-tracking sensor networks, this paper focuses on the scenarios where users can tolerate a certain degree of imprecision in their query results. We intend to develop a location management scheme that can achieve two goals. First, multiple precision levels are provided. Second, the query cost is proportional to the precision level. To achieve these two goals, we propose a tree-based imprecision-tolerant location management scheme that includes three major components: (1) update and query mechanisms that can support imprecision-tolerant queries, (2) the approach to taking the statistics of imprecision-tolerant queries and (3) a tree construction algorithm that can reduce the query cost and minimize the increment of update cost. Performance evaluations are conducted through simulations to verify the proposed scheme.

Keywords: imprecision-tolerance; in-network processing; location management; object tracking; wireless sensor networks; wireless communications

Received 5 November 2008; revised 3 March 2009
Handling editor: Ing-Ray Chen

1. INTRODUCTION

The rapid progress of wireless communication and embedded micro-sensing MEMS technologies have made *wireless sensor networks* (WSN) possible. Applications of WSN have been widely studied (e.g. in [1–3]). Object tracking is one of the important issues of WSN, which has applications in military intrusion detection, habitat monitoring and so on. The key steps of object tracking include event detection, target classification and location estimation [4–10]. In a WSN, when the locations of objects are successfully determined, a location management scheme for continuously reporting objects' locations and disseminating users' queries is required [11–14]. The main theme of this paper is location management. We explore the in-network data processing capability of WSN by executing distributed location updates and queries inside the network. In particular, we consider the scenarios where users can tolerate a certain degree of imprecision in their query results.

Inaccuracy of sensing data is inherent in WSN, and applications of WSN can usually tolerate some degree of imprecision. These properties have been exploited in the design of network protocols for WSN. For example, precision-constrained data aggregation is considered in [15], and a storage system that supports drill-down queries with different precision levels is proposed in [16]. Similarly, in an object-tracking sensor network, maintaining the exact locations of objects anytime is almost infeasible [10, 13, 17]. Not only the positioning results may contain errors, but also the data transfer delay and object mobility may make the locations of objects known by users not be up-to-date. Fortunately, imprecision is tolerable in many object-tracking applications. For example, when life scientists intend to pursue an animal, it may be sufficient to know its moving direction rather than its exact location. In addition, the location information recorded several hours ago, instead of at the current time, may still be helpful for the life scientists to understand the animal's daily life. Therefore, developing an in-network location management scheme to support imprecision-tolerant queries is desirable for object-tracking sensor networks.

In-network location management schemes for object-tracking sensor networks have been studied in [11–13, 18–20]. In [11], 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. Thus, the communication cost is reduced. Nevertheless, this work fails to consider the physical structure of the WSN. Reference [12] further takes the physical structure of the network into consideration while constructing the logical tree. In addition, when the query cost dominates the communication cost, a way to reduce the query cost is proposed. This results in further reduction of the overall cost. Statistics is required in [12]. Reference [20] proposes a Markov-chain model to generate the mobility profile so that it is unnecessary to take statistics. Besides, reference [19] shows that constructing an optimal tree is NP-complete. Both [11, 12] consider precise object tracking. A storage scheme called EASE, which can support imprecision-tolerant object tracking, is studied in [13]. The goal of the EASE scheme is to reduce network traffic for transmitting updates and queries. The location information of an object is stored in a centric storage node and a local storage node. When a user intends to know the location of an object, the query will be forwarded from the querying node to the centric storage node of that object. If the precision level is satisfactory, the centric storage node will reply to this query. Otherwise, the query will be forwarded to the local storage node, which has more precise location information of that object. Because the EASE scheme benefits from the structure-free property, it can tolerate faults such as sensor failure well. However, this scheme has two major drawbacks. First, when the querying node is very close to the local storage node of the queried object, the query will still be forwarded to the centric storage node, which may be far from the querying node. Second, only two precision levels are provided. Motivated by the EASE scheme, we argue that an imprecision-tolerant location management solution should achieve two desirable goals. First, multiple precision levels should be provided. Second, the query cost should be proportional to the precision level.

In this paper, we propose an in-network location management scheme to support imprecision-tolerant queries for object-tracking sensor networks. Two types of imprecision are considered. *Spatial imprecision* means that an object could be located *near*, rater than *at*, the location reported by the WSN. *Temporal imprecision* means that the location reported by the WSN may be recorded *near*, rather than *at*, the current time. For both of these two imprecision types, two desirable goals should be achieved. First, multiple precision levels should be provided. Second, the query cost should be proportional to the precision level. For example, for spatial imprecision, the report provided by node C should be more accurate than that provided by node A, because node C is farther from the sink (Fig. 1a). Similarly, for temporal imprecision, the location reported by node C should be newer than that reported by node A (Fig. 1b).

We observe that a tree-based location management scheme similar to those proposed in [11, 12] could achieve these two goals naturally. A detailed explanation will be given in Section 2.3. Based on this observation, this paper proposes a tree-based location management scheme to support imprecision-tolerant queries. To begin with, we describe the update and query mechanisms that can be used to support imprecision-tolerant queries. The proposed update and query mechanisms can be applied to any tree structure. We then make an observation regarding the relationship between the communication cost and the tree structure, and then propose a tree construction algorithm to facilitate the proposed imprecision-tolerant location management scheme by reducing the query cost while minimizing the increment of the update cost. Finally, performance studies are conducted via simulations.

The remainder of this paper is organized as follows. Section 2 describes the network model and reviews the operation of the tree-based location management schemes. The proposed imprecision-tolerant location management scheme is presented

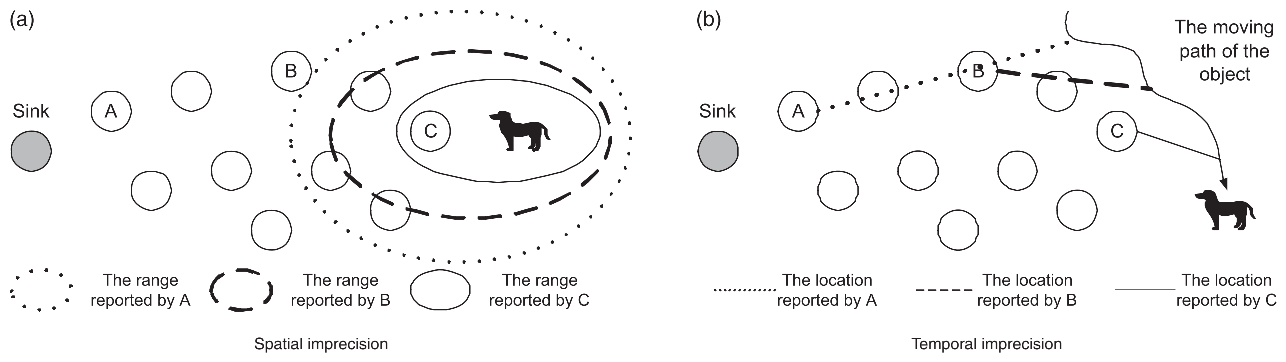


FIGURE 1. Examples of spatial imprecision and temporal imprecision.

in Section 3. Performance studies are given in Section 4. Section 5 concludes this paper; besides, possible future work is presented.

2. PRELIMINARIES

2.1. Network model

Our proposed network model includes a *sensing submodel* and a *communication submodel*. For the sensing submodel, we assume that a simple *nearest-sensor tracking* model is adopted, in which the sensor that receives the strongest signal from an object is responsible for tracking the object (this can be achieved from [21]). Therefore, the sensing field can be partitioned into a *Voronoi graph* [22], as shown in Fig. 2a, where each sensor is responsible for the its polygon area. In this paper, we assume that whenever an object crosses the boundary of the Voronoi graph, a movement event will be triggered. More specifically, these movement events can be regarded as events on the edges of the corresponding Delaunay triangulation, denoted by $G_S = (V_{G_S}, E_{G_S})$. Figure 2b shows the corresponding G_S of Fig. 2a. The definition of a movement event can be defined well using G_S.

For the communication submodel, given a set of sensors and the transmission range of sensors, we can derive a communication graph $G_C = (V_{G_C}, E_{G_C})$, where V_{G_C} represents sensors and an edge $(u, v) \in E_{G_C}$ if and only if u and v can communicate with each other directly. In this paper, we assume that the network G_C is connected. G_C will be used to calculate the communication cost.

2.2. Tree-based location management

Tree-based location management has been studied in [11, 12, 18]. Because the proposed imprecision-tolerant location management also follows this tree-based model, we briefly review the operation of the tree-based location management schemes.

First, a logical tree $T = (V_T, E_T)$ rooted at the sink will be constructed, where V_T represents sensors and E_T represents tree edges. Note that T is a logical tree and thus could be of any shape. Specifically, an edge $(u, v) \in E_T$ may neither belong to E_{G_S} nor belong to E_{G_C}. Therefore, for each $(u, v) \in E_T$, we assume that the communication cost of (u, v) is the hop count between u and v on G_C. By this design, the shape of the tree will be more flexible.

After T is constructed, when an object moves from one sensor u to another node v (i.e. across edge (u, v) on G_S), update packets will be forwarded to the lowest common ancestor of u and v in T. For example, in Fig. 2b, a tree rooted at sensor A is constructed (solid lines). When an object moves from K to E, update packets will be forwarded from K to D and from E to D, respectively. This allows any sensor to trace any object under its subtree. For example, when D receives the update packets, it knows that the object is located somewhere in the subtree rooted at E rather than that rooted at F. Afterward, when D receives a query, it will forward the query to E. Therefore, any object can be traced along T easily.

2.3. Observation

We observe that a tree-based location management could achieve the desired goals naturally; that is, multiple precision levels should be provided, and the query cost should be proportional to the precision level. We use an example to explain this observation. Figure 3a shows a tree used for location management. In the tree-based schemes, when an object moves from one sensor to another, the update messages will be forwarded to the lowest common ancestor of those two sensors in the tree. Thus, in Fig. 3a, when an object originally located

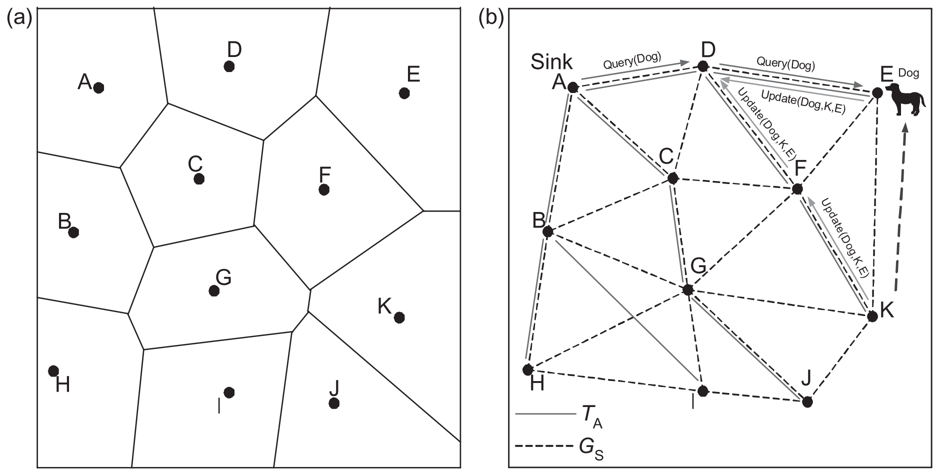
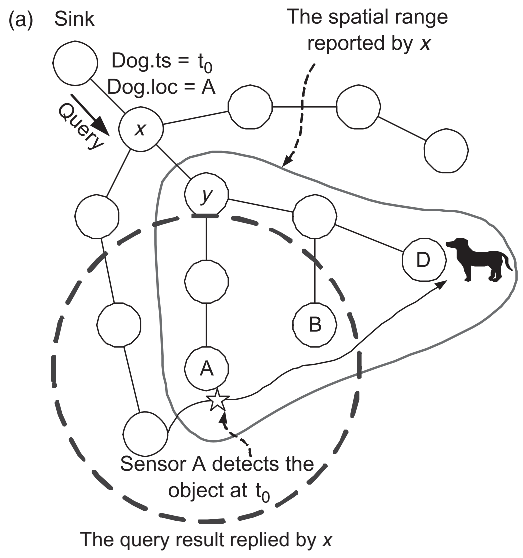


FIGURE 2. (a) The Voronoi graph of a sensor network that consists of eleven nodes $\{A, B, \ldots, K\}$. (b) The corresponding Delaunay triangulation G_S (dotted lines), where a logical tree rooted at sink A is constructed and represented by solid lines.


(a)
Sink
Dog.ts = t0
Dog.loc = A
Query
x
y
The spatial range reported by x
D
B
A
Sensor A detects the object at t0
The query result replied by x

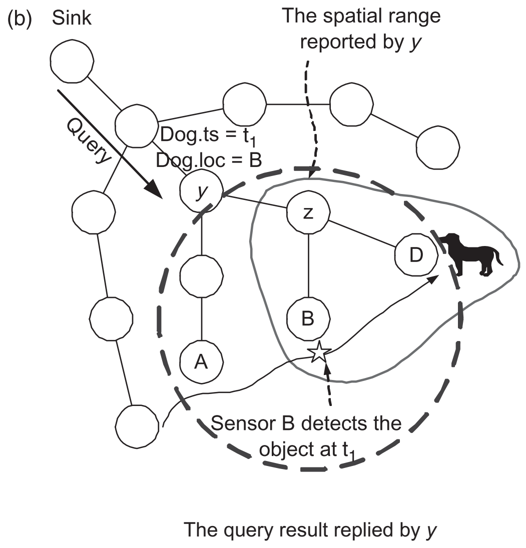

(b)
Sink
Query
Dog.ts = t1
Dog.loc = B
y
z
The spatial range reported by y
D
B
A
Sensor B detects the object at t1
The query result replied by y

FIGURE 3. An example of the tree-based location management scheme, where the dotted circles shown in (a) and (b) are $C(A, tolerant\_radius)$ and $C(B, tolerant\_radius)$, respectively, where $C(x, r)$ denotes the circle area centered at sensor x with a radius of r.

outside the spatial range of the subtree rooted at y moves into the range of A at time t_0, node x (i.e. the parent of y) will be updated. However, x will not be updated again unless the object leaves the range of y's subtree. Therefore, when the location of the object is provided by x, one can only derive that the object is located at some sensor in y's subtree after time t_0. On the contrary, in Fig. 3b, if the location of the object is provided by a deeper node, say y, then one can derive that the object is located at some sensor in z's subtree after time t_1. From the above example, we see that a user can get more precise location information when a query goes deeper down the tree. This also implies that a higher query cost is required. Thus, in a tree-based scheme, the query cost is proportional to the precision level. In addition, because of its hierarchical structure, a tree-based solution can provide multiple precision levels easily. Based on this observation, this paper proposes a tree-based location management scheme to support imprecision-tolerant queries. The detailed proposed scheme will be presented in the next section.

3. IMPRECISION-TOLERANT LOCATION MANAGEMENT

The proposed tree-based imprecision-tolerant location management scheme is presented in this section. First, assuming that a logical tree T has been constructed, we introduce our proposed update and query mechanisms. Second, we discuss how to collect query statistics. From these statistics, we show how to construct a cost-efficient tree T.

3.1. Imprecision-tolerant update and query mechanisms

We assume that a tree T rooted at the sink has been constructed. Each sensor x will maintain an object list OL_x to store the information of objects known by x. For each object o in OL_x, x maintains three fields:

- $o.next$: The sensor node from which more precise information about object o can be found. If o is currently tracked by x, then $o.next = x$. Otherwise, $o.next = y$, where y is a child of x and o is currently located somewhere in Subtree(y), where Subtree(y) denotes the subtree rooted at y. The major purpose of the update mechanism is to maintain the value of $o.next$ correctly.
- $o.loc$: The latest location of o known by x.
- $o.ts$: The timestamp when $o.loc$ was recorded by x.

Our update mechanism works as follows. When an object o moves from sensor a to sensor b at time t (recall that $(a, b) \in E_{G_S}$), two Update(o, a, b, t) packets will be initiated by a and b. These packets will be forwarded to the lowest common ancestor of a and b in T. When a sensor x receives such a packet, it will take the following actions. For simplicity, we say that a node itself is a descendant of itself.

- If b is not a descendant of x, then x will remove o from OL_x, because o is not located anywhere in Subtree(x) now. Then, x will further forward Update(o, a, b, t) to its parent.
- If b is a descendant of x but a is not a descendant of x, then x will add o into OL_x. If $x = b$, then $o.next = b$. Otherwise, $o.next$ is set to the child of x that sends Update(o, a, b, t) to x. In addition, x sets $o.loc = b$ and $o.ts = t$. Then, x will further forward Update(o, a, b, t) to its parent.

- If both a and b are descendants of x (i.e. x is the lowest common ancestor of a and b in T), then x will modify o's information in OL_x. If $x = b$, then $o.next = b$. Otherwise, $o.next$ is set to the child of x whose subtree contains b. In addition, x sets $o.loc = b$ and $o.ts = t$.

Next, we present our query format and query mechanism. Each query can be represented by Query($o, tole\_radius, tole\_time$), where $tole\_radius$ and $tole\_time$ denote the spatial imprecision and time imprecision that can be tolerated, respectively. More precisely, the distance between the *reported* location of o and the *real* location of o should be less than or equal to $tole\_radius$ and the reported location should be recorded after $cur\_time - tole\_time$, where cur_time denotes the current time.

The imprecision-tolerant query mechanism operates as follows. (Note that we assume that all queries will be issued from the sink. When an object enters the network, an update packet will be forwarded to the sink; thus, the sink must have the information of all objects tracked by the WSN.) When a sensor x (including the sink) receives a query Query($o, tole\_radius, tole\_time$), x will check its OL_x and take the following actions. (Note that the update mechanism ensures that if x is the intended receiver of the query, then $o \in \mathrm{OL}_x$.) Here we denote by $C(x, r)$ the circle area centered at sensor x with a radius of r.

- If the queried object is tracked by x currently (i.e. $o.next = x$), then x will reply to the query immediately. The detected time will be set to the current time and the detected location will be set to x itself.
- If the queried object is not tracked by x currently, then x will check the following two conditions:

  (i) $o.ts \geq cur\_time - tole\_time$.
  (ii) For each sensor $z \in Subtree(o.next)$, z is inside $C(o.loc, tole\_radius)$.

  Based on the result of checking, x acts as follows.
- If both conditions are true, x will reply to the query by stating that o is located at $o.loc$ at time $o.ts$.
- Otherwise, the query will be further forwarded to $o.next$ and the same procedure will be repeated until the object is found or the above two conditions are satisfied.

Condition (i) is for temporal imprecision. Condition (ii) is for spatial imprecision. According to our update mechanism, o is currently being tracked by a sensor in Subtree($o.next$). (Note that x does not know exactly which sensor is currently tracking o.) If condition (ii) is true, then the distance from $o.loc$ to each sensor in Subtree($o.next$) will be less than or equal to the tolerable distance $tole\_radius$. This implies that the distance between $o.loc$ stored in OL_x and the real location of o is less than or equal to $tole\_radius$. This also means that $o.loc$ can be tolerated. Therefore, when both conditions are true, a reply can be sent.

An example is shown in Fig. 3, where a query Query($Dog, tole\_radius, tole\_time$) is issued. In the case of Fig. 3a, we can see that x cannot reply to this query even if $t_0 \geq cur\_time - tole\_time$, because D is one of y's descendants and D is not within $C(A, tole\_radius)$. On the contrary, in the case of Fig. 3b, y can reply to the query if $t_1 \geq cur\_time - tole\_time$, because z and all of its descendants are within $C(B, tole\_radius)$.

3.2. Query statistics

In this paper, we assume that users' queries have regular patterns. For example, in habitat monitoring, life scientists may query an animal everyday. Thus, we can collect statistics so as to optimize the communication cost. The purpose of the query statistics is to identify the correlation of sensors. Later, we will show that *uncorrelated sensors* should not be put together under a subtree to reduce the query cost and *correlated sensors* should be put together to reduce the update cost.

The statistics is done by the sink. The sink will maintain a counter $x.rep\_cnt$ for each sensor x. Whenever the sink receives a response to a query Query($o, tole\_radius, tole\_time$) indicating that o is located at sensor x, $x.rep\_cnt$ will be increased by 1. (Note that this query may be replied by a sensor z, where $z \neq x$.) In addition, a correlation counter c_cnt(x, y) is also increased by 1 for each y located inside $C(x, \min\{tole\_radius, tole\_time \times \mathrm{spd}(o)\})$, where spd($o$) is the observed average speed of o. Based on these two counters, we define a correlation function

$$\mathrm{Corr}(x, y) = \mathrm{c\_cnt}(x, y)/x.rep\_cnt. \qquad (1)$$

Note that the statistics will be taken continuously, and thus T may need to be recomputed periodically. However, we assume that the query patterns will not change frequently and so we do not need to reconstruct T frequently.

3.3. Tree optimization for imprecision-tolerant queries

To begin with, we make an observation. Then, based on the observation, we present our tree construction algorithm. We observe that uncorrelated sensors should not be put together under a subtree so as to reduce the query cost while correlated sensors should be put together so as to reduce the update cost.

We use an example to explain this observation. In Fig. 4, we consider a scenario where z receives a query Query($o, tole\_radius, tole\_time$) and OL_z indicates that $o.loc = x$. Assume that the values of Corr(x, d), Corr(x, e) and Corr(x, h) are high. This implies that the probability that d, e and h are located inside $C(x, \min\{tole\_radius, tole\_time \times \mathrm{spd}(o)\})$ is high and the probability that a, b, c, f, g and k are located inside $C(x, \min\{tole\_radius, tole\_time \times \mathrm{spd}(o)\})$ is low. Thus, d, e and h are correlated sensors of x and a, b, c, f, g and k are uncorrelated sensors of x. Now we consider the tree shown in Fig. 4a. Because Subtree(g) contains some uncorrelated sensors of x, it is very likely that z cannot reply to this query. This may lead to a high query cost because z needs to forward the query to its child. On the contrary, in Fig. 4b, because Subtree(x) only contains the correlated sensors of x, it is more likely that z can reply to this query. However, one drawback of the tree in Fig. 4b is its high update cost. For example, when an object moves from e to x, update messages have to be forwarded to z (i.e. the lowest common ancestor of e and x). To reduce the update cost, if we can put e and x (which are correlated sensors) together as shown in Fig. 4c, then z can still reply to this query with high probability and the update cost can also be kept low.

Now, we describe our tree construction algorithm. Based on the observation mentioned above, sensors will be divided into clusters based on their correlation. Each cluster is then organized into a subtree. However, when connecting these subtrees together, a subtree should not be connected to another subtree because this implies that uncorrelated sensors will be attached. For example in Fig. 5a, when Subtree(y) is connected to Subtree(x), this implies that the members of Subtree(x) will change and include some uncorrelated sensors. Therefore, in order to connect these clusters together, some nodes have to be sacrificed to serve as bridges of these clusters. We call the structure formed by these bridges the *backbone*. For example, in Fig. 5b, some nodes will be selected to be backbone nodes and subtrees will be connected to backbone nodes. By this design, the members of the subtrees will not be destroyed. Therefore, our algorithm consists of three steps: (1) *BackboneConstruction* (to select backbone nodes and to construct the backbone tree), (2) *SubtreeFormation* (to cluster non-backbone nodes and to organize them into subtrees) and (3) *ConnectingSubtrees* (to connect subtrees to the backbone). Details of these steps are described below.

(1) *BackboneConstruction*: In this steps, an important issue is how to select backbone nodes. We argue that the nodes that are close to the sink and with lower rep_cnt should be selected to be backbone nodes. The reason is explained as follows. With our design, we can see that if a reply indicates that the queried object is located at a sensor that is a non-backbone node, then

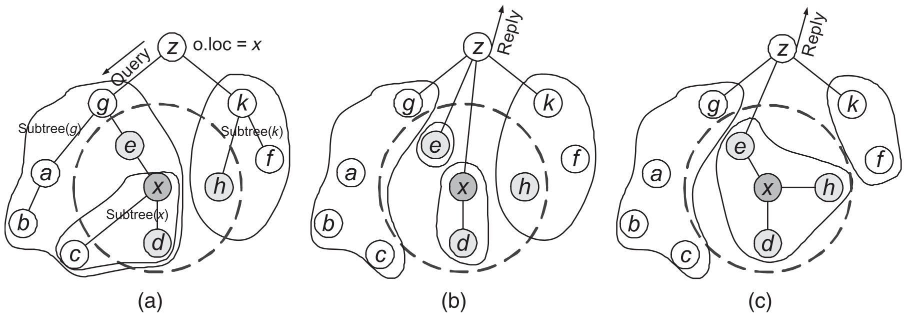


FIGURE 4. Some observations on arranging correlated and uncorrelated sensors, where each irregular area represents a subtree and each dotted circle denotes $C(x, \min\{tole\_radius, tole\_time \times \text{spd}(o)\})$.

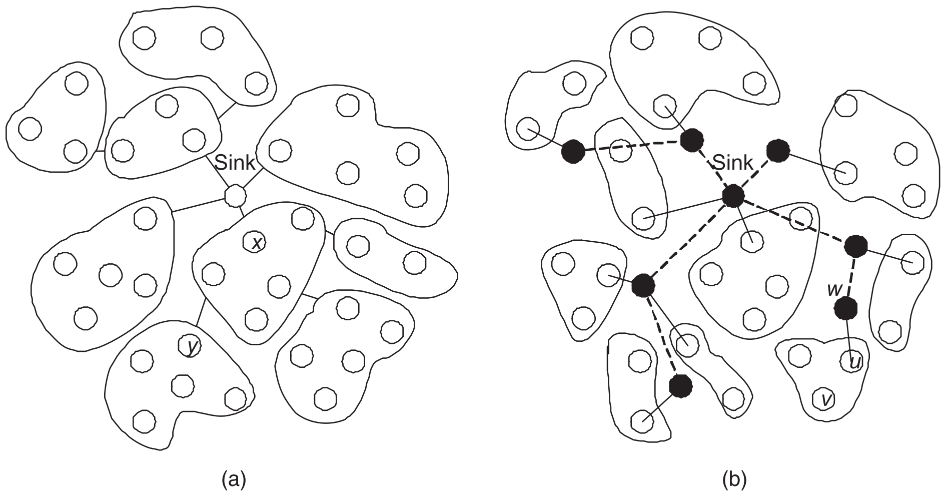


FIGURE 5. (a) An example of connecting a subtree to another subtree. (b) An example in which subtrees are connected to the backbone tree, where the black nodes are backbone nodes and the dotted lines are the edges of the backbone tree which may consist of multiple hops.

this reply will be sent by a backbone node with high probability. For example, in Fig. 5b, if a reply indicates that the queried object is located at sensor v, there is a high probability that this reply is sent by the backbone node w because the members of Subtree(u) are v's high correlated sensors. In this case, some query cost can be saved. On the contrary, if a reply indicated that the queried object is located at a sensor that is a backbone node, then this reply will be sent by that backbone node itself with high probability, because many uncorrelated sensors will be attached to that backbone node. In this case, there can be no saving in the query cost. Therefore, it is preferred that the backbone nodes are selected from those nodes with lower *rep_cnt*, because for a backbone node x, $x.rep\_cnt$ will be increased by 1 when a reply indicates that the object is located at x and there is high probability that this reply is sent by x itself. (There can be no saving in the query cost) In addition, we can see that most queries will be replied by backbone nodes. Thus, when backbone nodes are close to the sink, more query cost can be saved.

Based on the aforementioned discussions, we adopt two parameters to select backbone nodes. The first parameter α ($0 \le \alpha \le 1$) is to set a constraint on the *rep_cnt* values of backbone nodes. Nodes with lower *rep_cnt* will be selected. (Note that we can also see that at most $\lfloor \alpha \times |V_{G_C}| \rfloor$ nodes will be selected as backbone nodes.) The second parameter β ($0 \le \beta \le 1$) is to set a constraint on the distances between backbone nodes and the sink.

Then, the backbone nodes will be organized into a backbone tree. We enforce that a backbone node x can only choose a backbone node y as its parent such that $\text{hc}(\text{sink}, x) = \text{hc}(\text{sink}, y) + \text{hc}(x, y)$, where $\text{hc}(u, v)$ denotes the hop count between u and v on G_C. (Recall that a tree edge may consist of multiple hops.) If there are multiple candidate parents, then the sensor y with the minimum $\text{hc}(x, y)$ will be selected. The reason behind this is to reduce the update cost.

The corresponding pseudo-code is shown in Procedure 1, where V_{BG} denotes the set of backbone nodes, cp denotes the set of candidate parents, and $\text{MAX\_HC} = \max\{\text{hc}(x, \text{sink}) | \forall x \in V_{G_C}\}$. The backbone selection is from line 1 to line 7, and the backbone construction is from line 8 to line 17. It is not hard to see that with this greedy strategy, a virtual backbone tree can be formed at the end.

(2) *SubtreeFormation*: This step will divide non-backbone nodes into clusters and form a subtree for each cluster. It works as follows. To begin with, we sort non-backbone nodes into a list L by their *rep_cnt* values in an ascending order. The reason for using an ascending order is explained below. An ideal tree for reducing query cost is one where non-backbone nodes are alone or in smaller clusters, because less uncorrelated sensors will be attached. However, small clusters will incur higher update cost. Figure 4b has shown an example. To minimize the increment of the update cost, we expect that correlated sensors with smaller *rep_cnt* are grouped into a larger cluster. When we examine non-backbone nodes in an ascending order, sensors

Procedure 1 *BackboneConstruction*(G_C, QueryStatistics).

```
1:  $V_{BG} \leftarrow \{\text{sink}\}$
2:  $y \leftarrow$ the sensor with the $\lfloor \alpha \times |V_G| \rfloor$th least $rep\_cnt$ among all sensors
3:  for each node $x \in V_{G_C}$ except for the sink do
4:    if $(\text{hc}(x, \text{sink}) < \beta \times \text{MAX\_HC}) \wedge (x.rep\_cnt < y.rep\_cnt)$ then
5:      $V_{BG} \leftarrow V_{BG} \cup \{x\}$
6:    end if
7:  end for
8:  for each node $x \in V_{BG}$ except for the sink do
9:    cp $\leftarrow \phi$
10:   for each node $y \in V_{BG}$ do
11:     if $\text{hc}(\text{sink}, x) = \text{hc}(\text{sink}, y) + \text{hc}(x, y)$ then
12:       cp $\leftarrow$ cp $\cup \{y\}$
13:     end if
14:   end for
15:   choose a node $p$ such that $\text{hc}(p, x) = \min\{\text{hc}(y, x) | \forall y \in \text{cp}\}$
16:   $x$'s parent $\leftarrow p$
17: end for
```

with lower *rep_cnt* will form subtrees first and sensors with higher *rep_cnt* will have a higher chance to be alone or in smaller clusters.

With the sorted list L, we visit each node in L sequentially. If a node $x \in L$ is not clustered yet, then a cluster C containing only x will be formed and x will assume itself as the leader of C denoted by C_ldr. Besides, a candidate list cl will be constructed and nodes that are candidates for joining C will be added into cl. To begin with, all of x's unclustered non-backbone neighbors will be added into cl. Then, each node $y \in$ cl will be checked whether it can be added into C. Two scenarios are considered.

- If $y.rep\_cnt > C\_ldr.rep\_cnt$, then y will check whether all current members of C are its correlated nodes. If so, y will be added into C. Since y has a higher *rep_cnt* value than the C_ldr, y will become the new leader of C. In addition, cl also will be updated by including all of y's unclustered non-backbone neighbors. (By this design, two members in the same cluster could be more than one hop away.)
- Otherwise, the leader will check whether y is its correlated nodes. If so, y will be added into C. Again, cl also will be updated by including all y's unclustered non-backbone neighbors.

The same procedure will be performed on all nodes in cl until cl becomes empty. Then, C is determined.

After the cluster members are determined, nodes in C will be organized into a subtree. Similar to the backbone construction, each backbone node $x \in C$ will choose a node $y \in C$ as its parent such that $\text{hc}(\text{sink}, x) = \text{hc}(\text{sink}, y) + \text{hc}(x, y)$. If there are more than one such candidate, then the sensor y with the minimum $\mathrm{hc}(x, y)$ will be selected. Then, we shall construct the next subtree until all non-backbone nodes are clustered. The pseudo-code of the *SubtreesFormation* is shown in Procedure 2, where $\mathrm{Nei}(x)$ denotes the neighbors of sensor x and min_corr is an adjustable system parameter. It is possible that some nodes in a cluster cannot determine their parents after this step. In the next step, those nodes will choose backbone nodes as their parents.

Procedure 2 *SubtreesFormation*(G_C, QueryStatistics).

```
1:  Sort non-backbone nodes into a list $L$ by their $rep\_cnt$ in ascending order
2:  for each node $x$ in $L$ do
3:    $x.examined \leftarrow 0$
4:  end for
5:  for each node $x$ in $L$ do
6:    if $x.clustered = 0$ then
7:      $C \leftarrow \{x\}$
8:      $C\_ldr \leftarrow x$
9:      $\mathrm{cl} \leftarrow \{y | y \notin V_{BG} \wedge y \in \mathrm{Nei}(x) \wedge y.clustered = 0\}$
10:     while $\mathrm{cl} \neq \phi$ do
11:       Extract a sensor $y$ from cl
12:       if $y.rep\_cnt > C\_ldr.rep\_cnt$ then
13:         if $\forall z \in C, \mathrm{Corr}(y, z) > min\_corr$ then
14:           $C\_ldr \leftarrow y$
15:           $C \leftarrow C \cup \{y\}$
16:           $\mathrm{cl} \leftarrow \mathrm{cl} \cup \{z | z \notin V_{BG} \wedge z \in \mathrm{Nei}(y) \wedge z.clustered = 0\}$
17:           $y.clustered \leftarrow 1$
18:         end if
19:       else if $y.rep\_cnt \leq C\_ldr.rep\_cnt$ then
20:         if $\mathrm{Corr}(C\_ldr, y) > min\_corr$ then
21:           $C \leftarrow C \cup \{y\}$
22:           $\mathrm{cl} \leftarrow \mathrm{cl} \cup \{z | z \notin V_{BG} \wedge z \in \mathrm{Nei}(y) \wedge z.clustered = 0\}$
23:           $y.clustered \leftarrow 1$
24:         end if
25:       end if
26:     end while
27:     for each node $x \in C$ do
28:       $\mathrm{cp} \leftarrow \phi$
29:       for each node $y \in C$ do
30:         if $\mathrm{hc}(\mathrm{sink}, x) = \mathrm{hc}(\mathrm{sink}, y) + \mathrm{hc}(x, y)$ then
31:           $\mathrm{cp} \leftarrow \mathrm{cp} \cup \{y\}$
32:         end if
33:       end for
34:       choose a node $p$ such that $\mathrm{hc}(p, x) = \min\{\mathrm{hc}(y, x) | \forall y \in \mathrm{cp}\}$
35:       $x$'s parent $\leftarrow p$
36:     end for
37:   end if
38: end for
```

(3) *ConnectingSubtrees*: Now we have a number of subtrees, each of which will be connected to the backbone tree in this step. As mentioned earlier, a subtree's parent (i.e. the parent of the root of the subtree) is usually responsible for replying to queries on behalf of the subtree. If a subtree's parent is closer to the sink, more query cost may be saved. Thus, we adopt a parameter γ ($0 \leq \gamma \leq 1$) to limit the distance between the root of the subtree and its parent. The pseudo-code of *ConnectingSubtrees* is shown in Procedure 3. Note that a tree edge may consist of multiple hops.

Because we assume that T will not be reconstructed frequently, the cost for tree construction can be ignored. Finally, we prove some properties of the virtual tree T constructed above. We say that a tree T is *deviation-free* if for all $x \in V_{G_C}$, the hop count of the tree path from x to the sink is equal to $\mathrm{hc}(x, \mathrm{sink})$.

Procedure 3 *ConnectingSubtrees*(G_C).

```
1:  for each subtree root $x$ do
2:    $\mathrm{cp} \leftarrow \phi$
3:    for each node $y \in V_{BG}$ do
4:      if $(\mathrm{hc}(\mathrm{sink}, x) = \mathrm{hc}(\mathrm{sink}, y) + \mathrm{hc}(x, y)) \wedge (\mathrm{hc}(\mathrm{sink}, y) \leq \gamma \times hc(sink, x))$ then
5:        $\mathrm{cp} \leftarrow \mathrm{cp} \cup \{y\}$
6:      end if
7:    end for
8:    choose a node $p$ such that $\mathrm{hc}(p, x) = \min\{\mathrm{hc}(y, x) | \forall y \in \mathrm{cp}\}$
9:    $x$'s parent $\leftarrow p$
10: end for
```

Theorem 1. *If G_C is connected, the tree T constructed by the proposed algorithm is a connected deviation-free tree rooted at the sink.*

Proof. First, we show that T is connected. For backbone nodes, we argue that nodes except for the sink can determine their parents successfully, because the sink is a backbone node and $\mathrm{hc}(\mathrm{sink}, x) = \mathrm{hc}(\mathrm{sink}, \mathrm{sink}) + \mathrm{hc}(x, \mathrm{sink})$ is always true (i.e. cp will not be empty). Thus, the backbone tree rooted at the sink is connected. For non-backbone nodes, some nodes will determine their parents in the step *SubtreeFormation*. On the contrary, nodes that do not determine their parents yet will determine their parents in the step *ConnectingSubtrees*. Because the sink is a backbone node, and both $\mathrm{hc}(\mathrm{sink}, x) = \mathrm{hc}(\mathrm{sink}, \mathrm{sink}) + \mathrm{hc}(x, \mathrm{sink})$ and $\mathrm{hc}(\mathrm{sink}, \mathrm{sink}) = 0 \leq \gamma \times \mathrm{hc}(\mathrm{sink}, x)$ are true, cp must not be empty. Therefore, all nodes except for the sink will have their own parents and this implies that T is connected.

Then, we show that T is a deviation-avoidance tree. When a node chooses its parent, the constraint $\mathrm{hc}(\mathrm{sink}, x) = \mathrm{hc}(\mathrm{sink}, y) + \mathrm{hc}(x, y)$ ensures that the tree path from x to x's parent is deviation-free. Therefore, the tree path from every node except for the sink to its parent will be deviation-free. This implies that T is deviation-free. Hence, the theorem follows. □

3.4. Practicality issues

In this section, we discuss two practicality issues related to our scheme. The first one is regarding the nearest-sensor model, in which it is assumed that the sensor that receives the strongest signal from the object is responsible for tracking the object. In practice, signal strength is unstable and is highly sensitive to environment change [23–25]. We believe that many existing schemes can solve this issue. In addition, our scheme can be extended to the imprecise nearest-sensor model easily. Note that the nearest-sensor model is used to distinguish which sensor should be responsible for tracking the detected object. That is, when an object moves from sensor a to sensor b, sensors a and b should be identified in our scheme. There are many ways to achieve this goal. For simplicity, we adopt the simple nearest-sensor model to identify sensors that should be responsible for tracking the object.

The second one is regarding structure maintenance. In this paper, we assume that the tree structure is known by all sensors. That is, each sensor knows the members of its subtrees so as to determine whether it is the lowest common ancestor of any two sensors. However, maintaining the tree structure is costly for storage-constrained sensors when the network scale becomes large. One alternative is to apply the clustering techniques [26, 27] to relieve the scalability problem. First, sensors will be clustered. Then, each cluster can be viewed as a tree node so that the number of tree nodes can be reduced.

4. SIMULATION RESULTS

We developed an event-driven simulator using C language to demonstrate the efficiency of our proposed imprecision-tolerant location management scheme. A sensing field with a size of 256 × 256 units is simulated, in which 1024 sensors are deployed randomly with uniform distribution. The sensor located at one of the corners of the sensing field is selected to be the sink. The detailed setting of the simulator is described as follows. The nearest-sensor tracking model is adopted to simulate the detection of objects; in addition, we simply assume that $G_C = G_S$ in the simulation. We assume that a medium access control (MAC) protocol supporting ideal wireless communication is adopted; thus, collisions and packet drops are not modeled in our simulator. The modified city mobility model proposed in [12] is adopted to simulate the movement of objects. Two query scenarios are simulated. In the first scenario, each object is queried evenly. In the second scenario, some objects will be queried frequently such that there are some query hotspots in the sensing field. Besides, for each query, the value of $tole\_radius$ is selected randomly from 0 to MAX_TOLE_RADIUS with uniform distribution, and the value of $tole\_time$ is selected randomly from 0 to MAX_TOLE_TIME with uniform distribution. The default setting of the simulation is shown in Table 1.

The tree constructed by our proposed tree construction algorithm is called (imprecision-tolerant query tree (QT). We compare the performance of the IQT trees with that of the DAT trees proposed in [12]. In the original DAT scheme, the query mechanism proposed in [12] is used. That query mechanism does not support imprecision-tolerant queries. In the DAT-I scheme, a DAT tree will be constructed and our proposed imprecision-tolerant query mechanism mentioned in Section 3.1 will be applied. The metric used for evaluating the performance is the communication cost, which is the sum of the update cost and the query cost. The update cost is defined as the number of hops used for transmitting update packets. For example, when an object moves from sensor a to sensor b, two update packets will be forwarded to the lowest common ancestor of a and b, denoted by c. Then, the update cost of this event is $\text{hops}(a, c) + \text{hops}(b, c)$, where $\text{hops}(u, v)$ denotes the number of hops between u and v. Similarly, the query cost is defined as the number of hops used for transmitting query packets. Lower communication cost implies better performance.

In Section 4.1, we compare the IQT scheme with the DAT scheme. In Section 4.2, we further investigate the impact of important parameters used in the IQT scheme, that is, α, β, γ and min_corr. In Section 4.3, two query scenarios are compared.

4.1. Performance comparison between IQT and DAT

To begin with, we consider the scenario in which each object is queried evenly. In Fig. 6, we observe the impact of objects' speeds. A higher speed implies a higher update cost. First, we compare the DAT scheme with the DAT-I scheme. We can see that the saved query cost is limited, because the DAT tree is optimized by minimizing the update cost rather than the query cost. On the contrary, the proposed IQT tree optimized by reducing the query cost incurred by imprecision-tolerant queries can be used to save more on query cost. Especially, when the query cost dominates the communication cost (i.e. when objects' speed is low), the IQT trees can reduce the communication cost significantly. We can further find that when the query rate

TABLE 1. The default setting of the simulation.

Mobility model	Modified city mobility model [12]
Sensing model	Nearest-sensor tracking model
Communication model	An ideal MAC protocol
Simulation time	2 592 000 s
MAX_TOLE_RADIUS	30 units
MAX_TOLE_TIME	3600 s
Number of objects	128

increases from 0.2 to 0.4, the total cost of IQT trees almost does not increase, because the IQT tree can make the query cost as low as possible. On the other hand, when the query rate increases from 0.2 to 0.4, the total cost of DAT trees is doubled. However, when the speed is high enough, the DAT trees still outperform the IQT trees, because the DAT tree is optimized by minimizing the update cost.

We also investigate the impact of query rates in Fig. 7. We can see that the total costs of the IQT trees do not increase when the query rates become higher. The major reason is also that the IQT trees are optimized by minimizing the query cost. On the other hand, the total cost of DAT trees increases rapidly when the query rate becomes higher. We can make a brief conclusion for the results shown in Figs. 6 and 7 as follows. When the update cost dominates the query cost, DAT that is designed for minimizing the update cost outperforms IQT. On the contrary, when the query cost dominates the update cost, IQT that is designed for minimizing the query cost outperforms DAT.

We further argue that the IQT tree can benefit from low query response time. In Fig. 8, we consider two cases in which the IQT trees and the DAT trees have similar performances in terms of total cost. However, we can see that the query cost of IQT trees can be minimized significantly. The major reason is that queries can be replied earlier. This implies low query response time to which users are sensitive.

4.2. Performance evaluation under different parameter setting

To get further insight into the performance of IQT, we investigate the impact of important parameters used in the IQT scheme, that is, α, β, γ and min_corr. Two scenarios are conducted. In the first scenario, the query cost dominates the total cost. The results are shown in Fig. 9. In Fig. 6a, we can see that the value of α should be low when the query cost dominates the total cost. As we mentioned in Section 3.3, most queries will be replied by backbone nodes. When the number of backbone nodes is high (i.e. the value of α is high), most queries cannot be replied to early. The values of β and γ do not affect the performance significantly. This is because only a few nodes are selected as backbone nodes. (In the

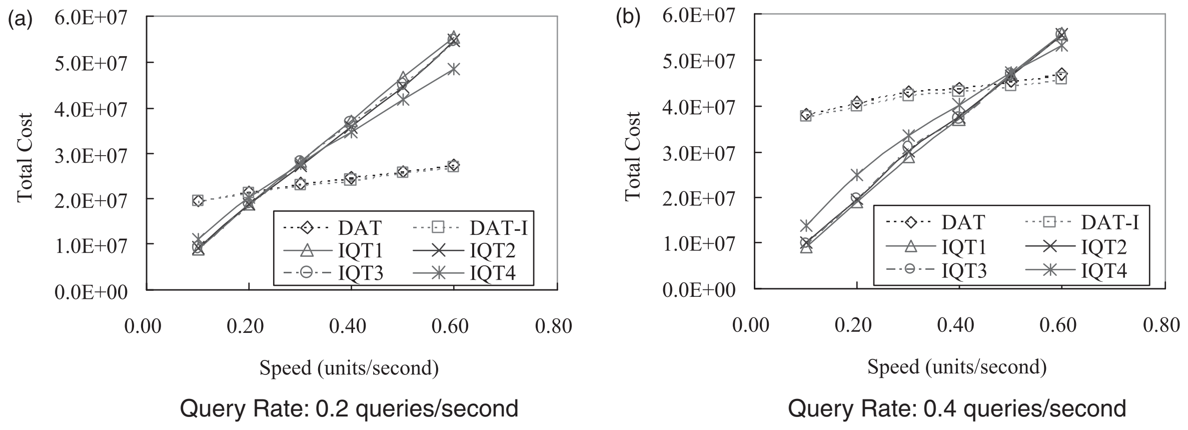


FIGURE 6. The impact of objects' speeds, where the settings of parameters $\langle\alpha, \beta, \gamma, min\_corr\rangle$ used in IQT1, IQT2, IQT3, IQT4 are $\langle 0.1, 0.3, 0.3, 0.9\rangle$, $\langle 0.3, 0.3, 0.3, 0.9\rangle$, $\langle 0.1, 0.5, 0.5, 0.9\rangle$, and $\langle 0.1, 0.3, 0.3, 0.6\rangle$.

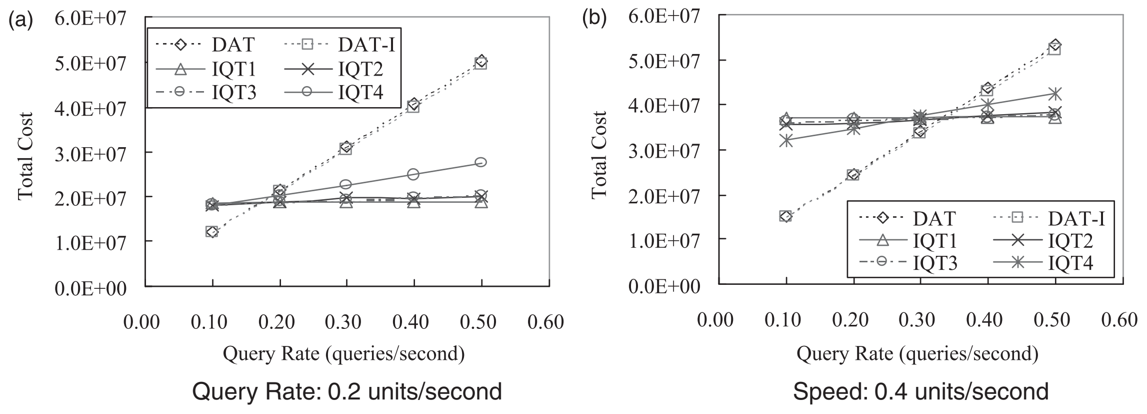


FIGURE 7. The impact of query rates.

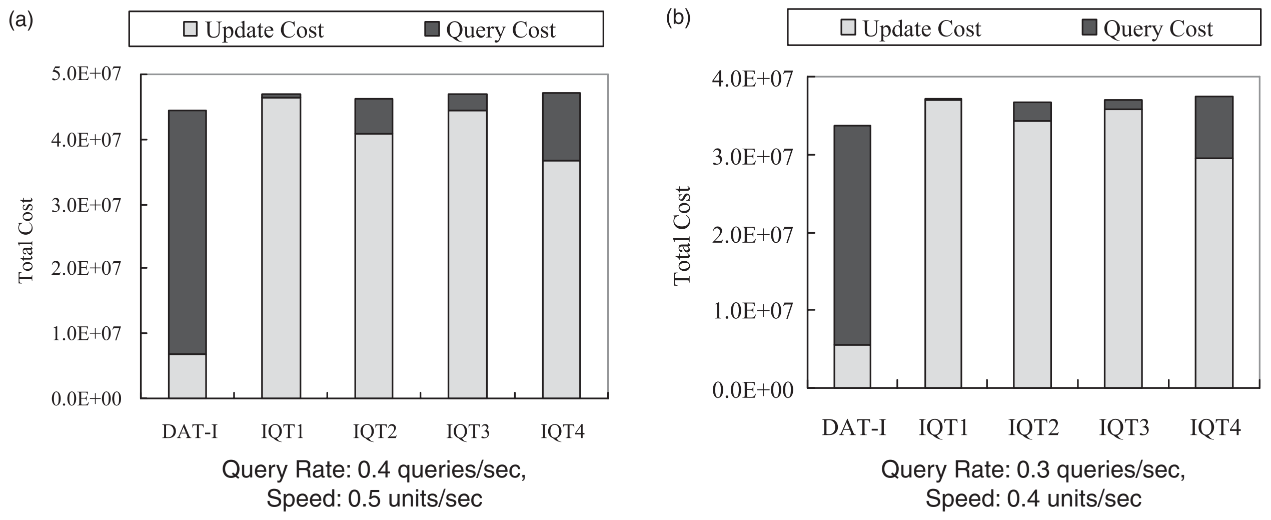


FIGURE 8. Comparison of ratios of update cost to query cost.

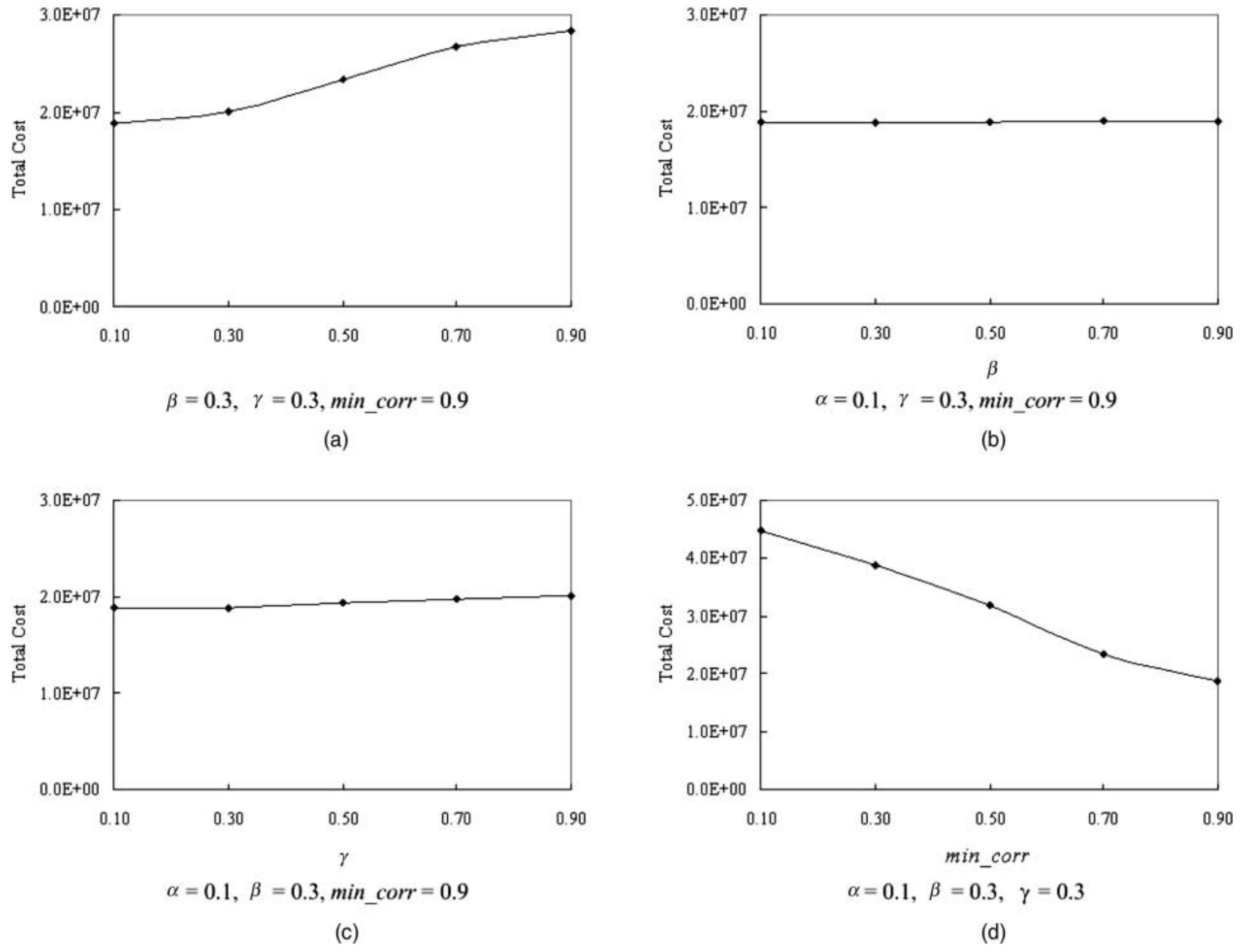


FIGURE 9. The impact of parameters in the IQT scheme, where the query rate is 0.5 queries/second and the objects' speed is 0.2 units/second.

experiments shown in Fig. 9b and 9c, the value of α is 0.1.) Thus, the impact is limited even though some of backbone nodes may be far from the sink (when the values of β and γ are high). Finally, we can see that the value of min_corr impacts the performance of the IQT scheme significantly. When the value of min_corr is low, uncorrelated sensors may be put together under a subtree. Thus, the result shown in Fig. 9 demonstrates our observation that uncorrelated sensors should not be put together under a subtree so as to reduce the query cost.

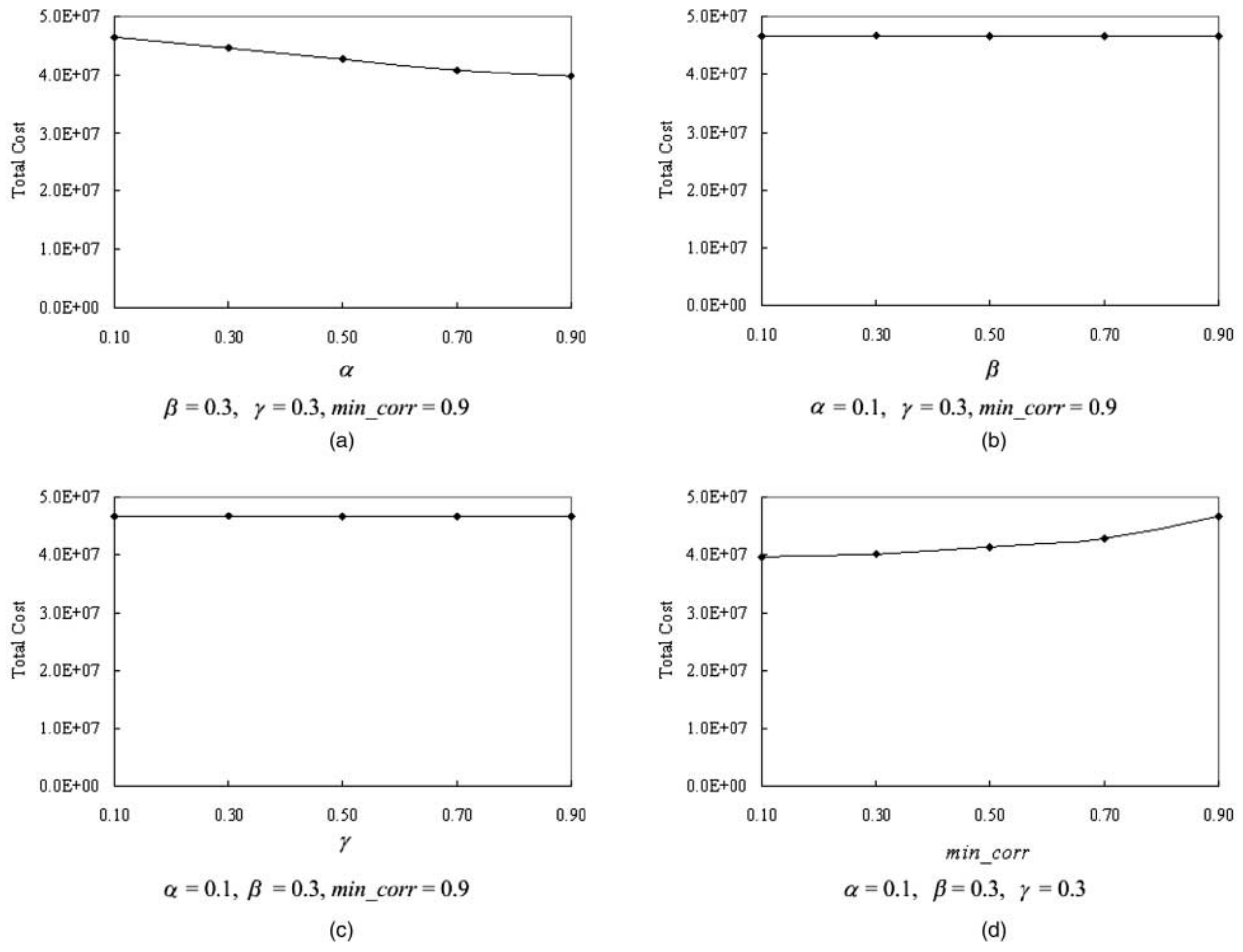


FIGURE 10. The impact of parameters in the IQT scheme, where the query rate is 0.2 queries/second and the objects' speed is 0.5 units/second.

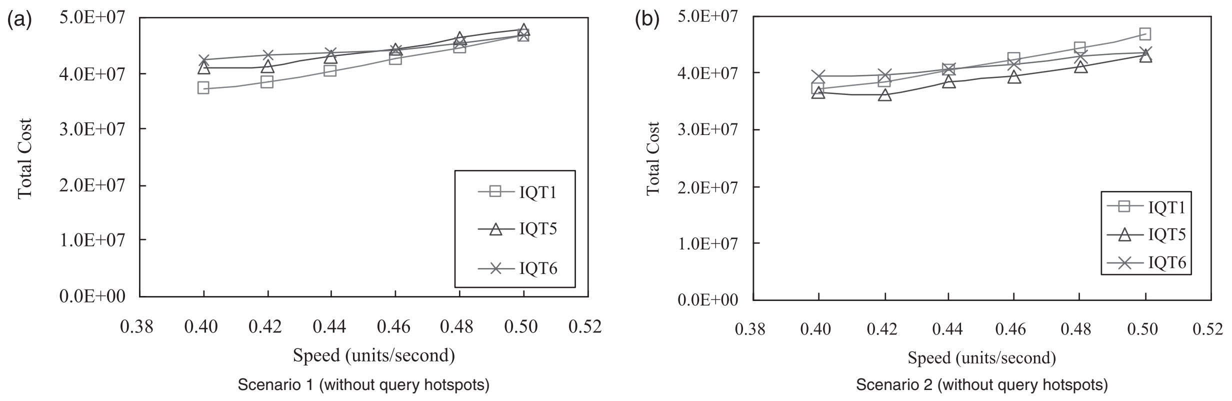


FIGURE 11. The impact of objects' speeds under two different query scenarios, where the query rate is 0.4 queries/second and the settings of parameters $\langle \alpha, \beta, \gamma, min\_corr \rangle$ used in IQT5 and IQT6 are $\langle 0.5, 1.0, 0.3, 0.9 \rangle$, and $\langle 0.8, 1.0, 0.3, 0.9 \rangle$, respectively.

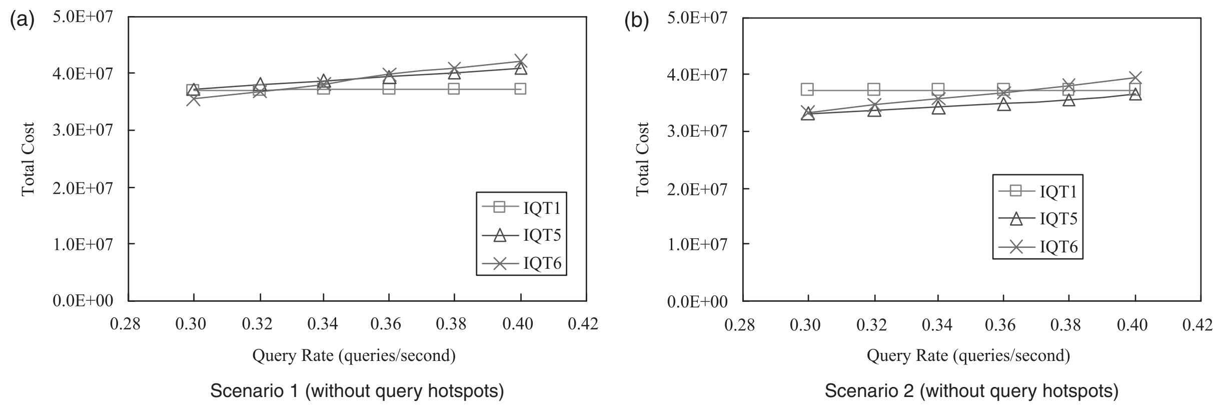


FIGURE 12. The impact of query rates under two different query scenarios, where the objects' speed is 0.4 units/second.

In the second scenario, the update cost dominates the total cost. The results are shown in Fig. 10. In Fig. 10a, we can see that the value of α should not be too low in this scenario. Recall that the subtrees formed by non-backbone nodes will connect to the backbone nodes. When only few nodes are selected as backbone nodes, the average number of hops between the roots of subtrees and the backbone nodes will be large. This means that when an object moves from one subtree to another subtree, the update cost will be high. In Fig. 10d, we can further see that it is meaningless to put correlated sensors together. The more important requirement is to shorten the average height of the lowest common ancestors when the update cost dominates the total cost.

Therefore, we can draw a similar conclusion made in Section 4.1. When the update cost dominates the query cost, the IQT scheme may not be a good choice. On the contrary, when the query cost dominates the update cost, the IQT scheme that is designed for minimizing the query cost should be considered.

4.3. Evaluation under different query scenarios

Finally, we consider a scenario in which some objects will be queried frequently such that there are some query hotspots in the sensing field. Figures 11a and 12a show the results of the scenario in which each object is queried evenly. On the other hand, Figs. 11b and 12b show the result of the scenario in which some objects are queried frequently. Intuitively, one may think that γ should be larger than or equal to β, because some backbone nodes will be useless (i.e. no non-backbone nodes will connect to them) when γ is less than β. This can be verified by the results shown in Figs. 11a and 12a. In the setting of IQT1, γ is equal to β. On the contrary, γ is less than β in the setting of IQT5 and IQT6. It can be seen that IQT1 outperforms IQT5 and IQT6. However, in the second query scenario, we argue that it is useful to make γ less than β. The reason is explained as follows. Because sensors in the query hotspots are queried frequently, it is better to select them to be non-backbone nodes. When the sensors in the query hotspots are close to the sink and γ is larger than or equal to β, the probability of selecting them as backbone nodes is high. This problem can be solved by setting γ less than β so that sensors in the query hotspots will not be selected as backbone nodes. This can be verified by the results shown in Figs. 11b and 12b, in which IQT5 outperforms IQT1.

5. CONCLUSIONS

Inaccuracy of location information is an inherent property in object-tracking sensor networks. In this paper, we propose a tree-based location management scheme for object-tracking sensor networks that can tolerate a certain degree of spatial and temporal imprecision. The proposed scheme consists of update and query mechanisms that can be used to support imprecision-tolerant queries. In addition, a tree construction algorithm is proposed to facilitate the proposed location management scheme, which can reduce the query cost while minimizing the increment of the update cost. By exploiting the features of the tree-based location management scheme, the proposed scheme can provide multiple precision levels and ensure that the query cost will be proportional to the precision level. We have also demonstrated the efficiency of the proposed scheme by simulation. With regard to future direction of research it will be interesting to develop a fault-tolerant tree construction algorithm and design a cross-layer protocol that take the contention and collision problems into consideration.

FUNDING

This research is supported by National Science Council, Taiwan, under the grant number NSC-97-2218-E-468-008. Y.-C. Tseng's research is co-sponsored by MoE ATU Plan, by NSC grants 95-2221-E-009-058-MY3, 96-2218-E-009-004, 97-3114-E-009-001, 97-2221-E-009-142-MY3, and 97-2218-E-009-026, by MOEA under grant 94-EC-17-A-04-S1-044, by ITRI, Taiwan, and by III, Taiwan.

REFERENCES

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

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

[3] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. and Anderson, J. (2002) Wireless Sensor Networks for Habitat Monitoring. *Proc. WSNA'02*, Atlanta, GA, USA, September 28, pp. 88–97. ACM, New York, NY, USA.

[4] Aslam, J., Butler, Z., Constantin, F., Crespi, V., Cybenko, G., and Rus, D. (2003) Tracking a Moving Object with Binary Sensors. *Proc. SenSys'03*, Los Angeles, CA, USA, November 5–7. ACM, New York, NY, USA.

[5] Blackman, S. and Popoli, R. (1999) *Design and Analysis of Modern Tracking Systems*. Artech House, Norwood, MA, USA.

[6] Kim, W., Mechitov, K., Choi, J.-Y. and Ham, S. (2005) On target tracking with binary proximity sensors. *Proc. IPSN'05*, Los Angeles, CA, USA, April 25–27, p. 40. IEEE, Piscataway, NJ, USA.

[7] Li, D., Wong, K., Hu, Y. and Sayeed, A. (2002) Detection, classification, and tracking of targets. *IEEE Signal Process. Mag.*, **19**, 17–29.

[8] Shrivastava, N., Madhow, R.M.U. and Suri, S. (2006) Target Tracking with Binary Proximity Sensors: Fundamental Limits, Minimal Descriptions, and Algorithms. *Proc. SenSys'06*, Boulder, CO, USA, October 31 - November 3, pp. 251–264. ACM, New York, NY, USA.

[9] Wang, Z., Bulut, E. and Szymanski, B.K. (2008) A Distributed Cooperative Target Tracking with Binary Sensor Networks. *Proc. ICC'08*, Beijing, China, May 19-23, pp. 306–310. IEEE, Washington, DC, USA.

[10] Wang, Z., Bulut, E. and Szymanski, B.K. (2008) Distributed Target Tracking with Imperfect Binary Sensor Networks. *Proc. GLOBECOM'08*, New Orleans, LA, USA, November 30 – December 4, pp. 1–5. IEEE, Washington, DC, USA.

[11] Kung, H.T. and Vlah, D. (2003) Efficient Location Tracking using Sensor Networks. *Proc. WCNC'03*, New Orleans, LA, USA, March 16–20, pp. 1954–1961. IEEE, Washington, DC, USA.

[12] Lin, C.-Y., Peng, W.-C. and Tseng, Y.-C. (2006) Efficient in-network moving object tracking in wireless sensor networks. *IEEE Trans. Mob. Comput.*, **5**, 1044–1056.

[13] Xu, J., Tang, X. and Lee, W.-C. (2008) A new storage scheme for approximate location queries in object-tracking sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, **19**, 262–275.

[14] Tseng, Y.-C., Kuo, S.-P., Lee, H.-W. and Huang, C.-F. (2004) Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. *Comput. J.*, **47**, 448–460.

[15] Tang, X. and Xu, J. (2006) Extending Network Lifetime for Precision-Constrained Data Aggregation in Wireless Sensor Networks. *Proc. INFOCOM'06*, Barcelona, Catalunya, Spain, April 23–29. IEEE, Washington, DC, USA.

[16] Ganesan, D., Greenstein, B., Perelyubskiy, D., Estrin, D. and Heidemann, J. (2003) An Evaluation of Multi-resolution Storage for Sensor Networks. *Proc. SenSys'03*, Los Angeles, CA, USA, November 5–7, pp. 89–102. ACM, New York, NY, USA.

[17] Cheng, R., Kalashnikov, D.V. and Prabhakar, S. (2004) Querying imprecise data in moving object environments. *IEEE Trans. Knowl. Data Eng.*, **16**, 1112–1127.

[18] Lin, C.-Y., Tseng, Y.-C., Lai, T.-H. and Peng, W.-C. (2008) Message-efficient in-network location management in a multi-sink wireless sensor network. *Int. J. Sensor Netw.*, **3**, 3–15.

[19] Liu, B.-H., Ke, W.-C., Tsai, C.-H., and Tsai, M.-J. (2008) Constructing a message-pruning tree with minimum cost for tracking moving objects in wireless sensor networks is np-complete and an enhanced data aggregation structure. *IEEE Trans. Comput.*, **57**, 849–863.

[20] Yen, L.-H. and Yang, C.-C. (2006) Mobility Profiling Using Markov Chains for Tree-Based Object Tracking in Wireless Sensor Networks. *Proc. SUTC'06*, Taichung, Taiwan, June 5–7, pp. 220–225.

[21] Chen, W.-P., Hou, J.C. and Sha, L. (2004) Dynamic clustering for acoustic target tracking in wireless sensor networks. *IEEE Trans. Mob. Comput.*, **3**, 258–271.

[22] Aurenhammer, F. (1991) Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, **23**, 345–405.

[23] Bahl, P. and Padmanabhan, V.N. (2000) RADAR: An In-building RF-Based user Location and Tracking System. *Proc. INFOCOM'00*, Tel-Aviv, Israel, March 26–30, pp. 775–784. IEEE, Washington, DC, USA.

[24] de Moraes, L.F.M. and Nunes, B.A.A. (2006) Calibration-Free WLAN Location System Based on Dynamic Mapping of Signal Strength. *Proc. MobiWac'06*, Terromolinos, Malaga, Spain, October 2, pp. 92–99. ACM, New York, NY, USA.

[25] Elnahrawy, E., Li, X. and Martin, R.P. (2004) The Limits of Localization Using Signal Strength: A Comparative Study. *Proc. SECON'04*, Santa Clara, CA, USA, October 4–7, pp. 406–414. IEEE, Washington, DC, USA.

[26] Xin, G., YongXin, W. and Fang, L. (2008) An Energy-Efficient Clustering Technique For Wireless Sensor Networks. *Proc. NAS'08*, Chongqing, China, June 12–14, pp. 248–252. IEEE, Washington, DC, USA.

[27] Younis, O. and Fahmy, S. (2004) Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach. *Proc. INFOCOM'04*, Hong Kong, China, March 7–11, pp. 629–640. IEEE, Washington, DC, USA.