Efficient In-Network Moving Object Tracking in Wireless Sensor Networks

Chih-Yu Lin, Wen-Chih Peng, and Yu-Chee Tseng
Department of Computer Science and Information Engineering
National Chiao Tung University
Hsin-Chu, 30050, Taiwan
{lincyu, wcpeng, yctseng}@csie.nctu.edu.tw
(corresponding author: Prof. Yu-Chee Tseng) [*]

Abstract

The rapid progress of wireless communication and embedded micro-sensing MEMS technologies has made *wireless sensor networks* possible. In light of storage in sensors, a sensor network can be considered as a distributed database, in which one can conduct *in-network* data processing. An important issue of wireless sensor networks is object tracking, which typically involves two basic operations: update and query. This issue has been intensively studied in other areas, such as cellular networks. However, the in-network processing characteristic of sensor networks has posed new challenges to this issue. In this paper, we develop several tree structures for in-network object tracking which take the physical topology of the sensor network into consideration. The optimization process has two stages. The first stage tries to reduce the location update cost based on a deviation-avoidance principle and a highest-weight-first principle. The second stage further adjusts the tree obtained in the first stage to reduce the query cost. The way we model this problem allows us to analytically formulate the cost of object tracking given the update and query rates of objects. Extensive simulations are conducted, which show a significant improvement over existing solutions.

Index Terms: object tracking, in-network processing, sensor network, data aggregation, mobile computing.

[*]Y. C. Tseng's research is co-sponsored by the NSC Program for Promoting Academic Excellence of Universities under grant number 93-2752-E-007-001-PAE, by Computer and Communications Research Labs., ITRI, Taiwan, by the Communications Software Technology Project of III, Taiwan, and by Intel Inc.

1 Introduction

The rapid progress of wireless communication and embedded micro-sensing MEMS technologies has made *wireless sensor networks* possible. Such environments may have many inexpensive wireless nodes, each capable of collecting, processing, and storing environmental information, and communicating with neighboring nodes. In the past, sensors are connected by wire lines. Today, this environment is combined with the novel *ad hoc* networking technology to facilitate inter-sensor communication [11, 12]. The flexibility of installing and configuring a sensor network is thus greatly improved. Recently, a lot of research activities have been dedicated to sensor networks [4, 5, 6, 7, 8, 9, 13, 14].

Object tracking is an important application of wireless sensor networks (e.g., military intrusion detection and habitat monitoring). Existing research efforts on object tracking can be categorized in two ways. In the first category, the problem of accurately estimating the location of an object is addressed [1, 10]. In the second category, in-network data processing and data aggregation for object tracking are discussed [8, 15]. The main theme of this paper is to propose a data aggregation model for object tracking. Object tracking typically involves two basic operations: *update* and *query*. In general, updates of an object's location are initiated when the object moves from one sensor to another. A query is invoked each time when there is a need to find the location of an interested object. Location updates and queries may be done in various ways. A naive way for delivering a query is to flood the whole network. The sensor whose sensing range contains the queried object will reply to the query. Clearly, this approach is inefficient because a considerable amount of energy will be consumed when the network scale is large or when the query rate is high. Alternatively, if all location information is stored at a specific sensor (e.g., the sink), no flooding is needed. But whenever a movement is detected, update messages have to be sent. One drawback is that when objects move frequently, abundant update messages will be generated. The cost is not justified when the query rate is low. Clearly, these are tradeoffs.

In [8], a *Drain-And-Balance* (DAB) tree structure is proposed to address this issue. As far as we know, this is the first in-network object tracking approach in sensor networks where query messages are not required to be flooded and update messages are not always transmitted to the sink. However, [8] has two drawbacks. First, a DAB tree is a logical tree not reflecting the physical structure of the sensor network; hence, an edge may consist of multiple communication hops and a high communication cost may be incurred. Second, the construction of the DAB tree does not take the query cost into consideration. Therefore, the result may not be efficient in some cases.

To relieve the aforementioned problems, we propose a new tree structure for in-network object tracking in a sensor network. The location update part of our solution can be viewed as an extension of [8]. In particular, we take the physical topology of the sensor network into consideration. We take a two-stage approach. The first stage aims at reducing the update cost, while the second stage aims at further reducing the query cost. For the first stage, several principles, namely deviation-avoidance and highest-weight-first ones, are pointed out to construct an object tracking tree to reduce the communication cost of location update. Two solutions are proposed: *Deviation-Avoidance Tree* (DAT) and *Zone-based Deviation-Avoidance Tree* (Z-DAT). The latter approach tries to divide the sensing area into square-like zones, and recursively combine these zones into a tree. Our simulation results indicate that the Z-DAT approach is very suitable for regularly deployed sensor networks. For the second stage, we develop a *Query Cost Reduction* (QCR) algorithm to adjust the object tracking tree obtained in the first stage to further reduce the total cost.

The way we model this problem allows us to analytically formulate the update and query costs of the solution based on several parameters of the given problem, such as rates that objects cross the boundaries between sensors and rates that sensors are queried. We have also conducted extensive simulations to evaluate the proposed solutions. The results do validate our observations.

Several other tracking-related problems have also been studied, but they can be considered independent issues from our work. The authors in [14] 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 [1, 3, 10, 15]. In [3], a dynamic clustering architecture that exploits signal strength observed by sensors is proposed to identify the set of sensors to track an object. In [15], a *convoy tree* is proposed for object tracking using data aggregation to reduce energy consumption.

The rest of this paper is organized as follows. Preliminaries are given in Section 2. DAT, Z-DAT, and QCR algorithms are presented in Section 3. Performance studies are conducted in Section 4. This paper concludes with Section 5.

2 Preliminaries

We consider a wireless sensor network deployed in a field for the purpose of object tracking. Sensors' locations are already known at a special node, called *sink*, which serves as the gateway of the sensor network to the outside world. We adopt a simple *nearest-sensor* model, which only requires the sensor that receives the strongest signal from the object to report to the sink (this can be achieved by [3]). Therefore, the sensing field can be partitioned into a Voronoi graph [2], as depicted in Fig. 1(a), such that every point in a polygon is closer to its corresponding sensor in that polygon than to any other. In practice, a sensor under our model may represent the clusterhead of a cluster of reduced-function sensors. In this work, however, we are only interested in the reporting behavior of these clusterheads.

Our goal is to propose a data aggregation model for object tracking. We assume that whenever an object arrives at or departs from the sensing range (polygon) of a sensor, a *detection event* will

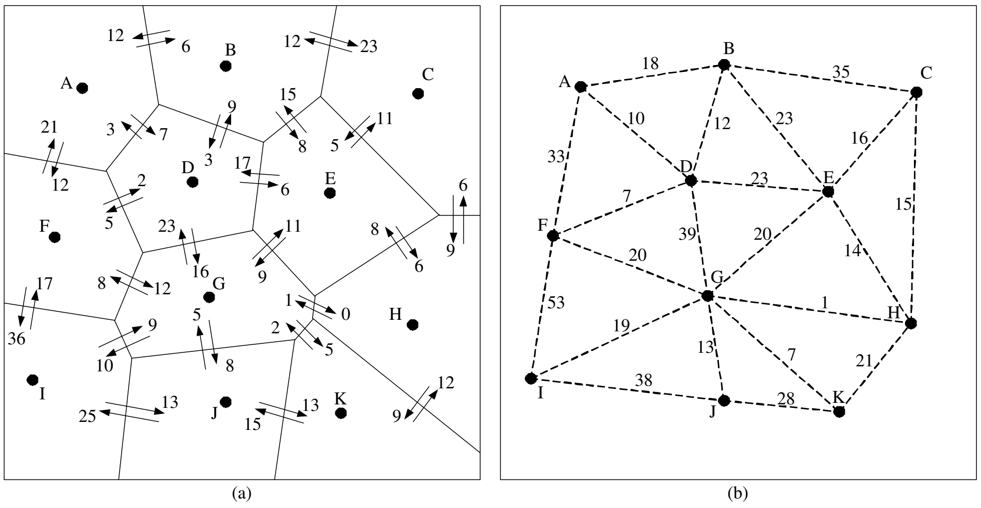


Figure 1: (a) The Voronoi graph of a sensor network. The arrival and departure rates between sensors are the numbers associated with arrows. (b) The graph G corresponding to the sensor network in (a). The number labelled on each edge represents its weight.

be reported (note that this update message are not always forwarded to the sink, as will be elaborated later). Two sensors are called *neighbors* if their sensing ranges share a common boundary on the Voronoi graph; otherwise, they are *non-neighbors*. Multiple objects may be tracked concurrently in the network, and we assume that from mobility statistics, it is possible to collect the event rate between each pair of neighboring sensors to represent the frequency of objects travelling from one sensor to another. For example, in Fig. 1(a), the arrival and departure rates between sensors are shown on the edges of the Vonoroi graph. In addition, the communication range of sensors is assumed to be large enough so that neighboring sensors (in terms of their sensing ranges) can communicate with each other directly. Thus, the network topology can be regarded as an undirected weighted graph $G = (V_G, E_G)$ with V_G representing sensors and E_G representing links between neighboring 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 b to a. This is because both arrival and departure events will be reported in our scheme. In fact, G is a Delaunay triangulation of the network [2]. Fig. 1(b) shows the

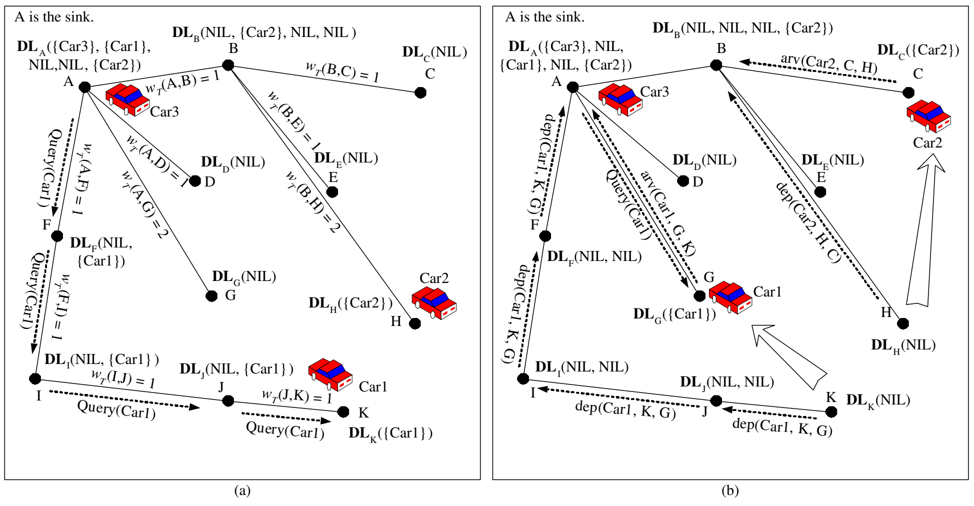


Figure 2: (a) An object tracking tree T, where the dotted lines are the forwarding path of a query for Car1. (b) The events generated as Car1 moves from sensor K to G and Car2 moves from H to C.

corresponding Delaunay triangulation of the sensor network in Fig. 1(a).

In light of the storage in sensors, the sensor network is able to be viewed as a distributed database. We will exploit the possibility of conducting in-network data aggregation for object tracking in a sensor network. Similar to the approach in [8], a logical weighted tree T will be constructed from G. For example, Fig. 2(a) shows an object tracking tree T constructed from the network G in Fig. 1(b). Movement events of objects are reported based on the following rules. Each node a in T will maintain a *detected list* $\boldsymbol{DL}_a = (L_0, L_1, \ldots, L_k)$ such that L_0 is the set of objects currently inside the coverage of sensor a itself, and $L_i, i = 1, \cdots, k$, is the set of objects currently inside the coverage of any sensor who is in the subtree rooted at the i-th child of sensor a, where k is the number of children of a. When an object o moves from the sensing range of a to that of b ($(a, b) \in E_G$), a departure event $dep(o, a, b)$ and an arrival event $arv(o, b, a)$ will be reported by a and b, respectively, alone the tree T. On receiving such an event, a sensor x takes the following actions:

- If the event is $dep(o, a, b)$, x will remove o from the proper L_i in $\boldsymbol{DL}_x$ such that sensor a belongs to the i-th subtree of x in T. If $x = a$, o will be removed from L_0 in $\boldsymbol{DL}_x$. Then x checks whether sensor b belongs to the subtree rooted at x in T or not. If not, the event $dep(o, a, b)$ is forwarded to the parent node of x in T.

- If the event is $arv(o, b, a)$, x will add o to the proper L_i in $\boldsymbol{DL}_x$ such that sensor b belongs to the i-th subtree of x in T. If $x = b$, o will be added to L_0 in $\boldsymbol{DL}_x$. Then x checks whether sensor a belongs to the subtree rooted at x in T or not. If not, the event $arv(o, b, a)$ is forwarded to the parent node of x in T.

The above data aggregation model guarantees that, disregarding transmission delays, the data structure $\boldsymbol{DL}_i$ always maintains the objects under the coverage of any descendant of sensor i in T. Therefore, searching the location of an object can be done efficiently in T; a query is only required to be forwarded to a proper subtree and no flooding is needed. For example, Fig. 2(a) shows the forwarding path of a query for Car1 in T. Fig. 2(b) shows the reporting events as Car1 and Car2 move and the forwarding path of a query for the new location of Car1.

Our goal in this paper is to construct an object tracking tree $T = (V_T, E_T)$ that incurs the lowest communication cost given a sensor network $G = (V_G, E_G)$ and the corresponding event rates and query rates, where $V_T = V_G$ and E_T consists of $|V_T| - 1$ edges with the sink as the root. Intuitively, T is a logical tree constructed from G, in which each edge $(u, v) \in T$ is one of the shortest paths connecting sensors u and v in G. Therefore, the weight of each edge (u, v) in T, denoted by $w_T(u, v)$, is modelled by the minimum hop count between u and v in G. The cost function can be formulated as $C(T) = U(T) + Q(T)$, where $U(T)$ denotes the update cost and $Q(T)$ is the query cost.

Table 1 summaries the notations used in this paper.

Table 1: Summary of notations.

$dist_G(u, v)$	The minimum hop count between u and v in G.
$dist_T(u, v)$	The sum of w_Ts of edges on the path connecting u and v in T.
$w_G(u, v)$	The event rate between u and v.
$w_T(u, v)$	The weight of edge (u, v) in T. ($= dist_G(u, v)$).
$lca(u, v)$	The lowest common ancestor of u and v.
$p(v)$	The parent of v in T.
$Subtree(v)$	Members of the subtree rooted at v.
$root(v)$	The root of the temporary subtree containing v during the construction of T.
$q(v)$	The query rate of v.
$neighbors(v)$	Neighbors of v.
$children(v)$	Children of v.

3 Tree Construction Algorithms

This section presents our algorithms to construct efficient object tracking trees. In Section 3.1, we develop algorithm DAT targeted at reducing the update cost. Then, in Section 3.2, based on the concept of divide-and-conquer, we devise algorithm Z-DAT to further reduce the update cost. In Section 3.3, algorithm QCR is developed to adjust the tree obtained by algorithm DAT/Z-DAT to further reduce the total cost.

3.1 Algorithm DAT (Deviation-Avoidance Tree)

Object tracking typically involves two basic operations: update and query. Based on the aggregation model in Section 2, updates will be initiated when an object o moves from sensor a to sensor b. It can be seen that both the departure event $dep(o, a, b)$ and the arrival event $arv(o, b, a)$ will be forwarded to the root of the minimum subtree containing both a and b. Therefore, the update cost $U(T)$ of a tree T can be formulated by counting the average number of messages transmitted in the network per unit time:

$$U(T) = \sum_{(u,v)\in E_G} w_G(u,v) \times (dist_T(u, lca(u,v)) + dist_T(v, lca(u,v))), \tag{1}$$

where $lca(u,v)$ denotes the root of the minimum subtree in T that includes both u and v (from now on, we will call $lca(u,v)$ the lowest common ancestor of u and v), and $dist_T(x,y)$ is the sum of weights of the edges on the path connecting x and y in T. For example in Fig. 2(a), $dist_T(F,K) = w_T(F,I) + w_T(I,J) + w_T(J,K) = 3$. In order to identify which factors affecting the value of $U(T)$, we show that $U(T)$ also can be formulated in a different way as follows.

Theorem 1. *Given any logical tree T of the sensor network G, we have*

$$U(T) = \sum_{(p(v),v)\in E_T} \left(w_T(p(v),v) \times \sum_{\substack{(x,y)\in E_G \wedge x\in Subtree(v) \\ \wedge y\notin Subtree(v)}} w_G(x,y) \right), \tag{2}$$

where $Subtree(v)$ is the subtree of T rooted at node v and $p(v)$ is the parent of v.

Proof. This can be proved by observing which events will be reported along an edge in T. Given $(p(v),v) \in E_T$, for any $(x,y) \in E_G$ where $x \in Subtree(v)$ and $y \notin Subtree(v)$, since the lowest common ancestor of x and y must not in $Subtree(v)$, any event generated on (x,y) will be transmitted from v to $p(v)$. Otherwise, no message will be transmitted from v to $p(v)$. This leads to the theorem. $\square$

From Eq. 1 and Eq. 2, we make three observations about $U(T)$:

- Eq. 1 contains the factor $dist_T(u, lca(u,v))$. Its minimal value is $dist_G(u, lca(u,v))$, which denotes the minimum hop count between sensor u and sensor $lca(u,v)$ in G. Therefore, we would expect that $dist_T(u, sink) = dist_G(u, sink)$ for each $u \in V_G$; otherwise, we say that

u ***deviates from*** its shortest path to the sink. If $dist_T(u, sink) = dist_G(u, sink)$ for each $u \in V_G$, we say that tree T is a ***deviation-avoidance*** tree. Fig. 3 shows four possible object tracking trees for the graph G in Fig. 1(b). The one in Fig. 3(b) is not a deviation-avoidance tree since $dist_T(E, A) = 3 > dist_G(E, A) = 2$. The other three are deviation-avoidance trees.

- Eq. 2 contains the factor $w_T(u, v)$. Its minimal value is 1 when $u \neq v$. Consequently, it is desirable that each sensor's parent is one of its neighbors. Only the tree in Fig. 3(d) satisfies this criterion. By selecting neighboring sensors as parents, the average value of $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))$ in Eq. 1 can be minimized. For example, the average values of $dist_T(u, lca(u, v))$
$+ dist_T(v, lca(u, v))$ are 3.591, 2.864, and 2.227 for the trees in Fig. 3(a), 3(c), and 3(d), respectively.

- In Eq. 1, the weight $w_G(u, v)$ will be multiplied by $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v))$. For two edges (u, v) and $(u', v') \in E_G$ such that $w_G(u, v) > w_G(u', v')$, it is desirable that $dist_T(u, lca(u, v)) + dist_T(v, lca(u, v)) < dist_T(u', lca(u', v')) + dist_T(v', lca(u', v'))$. Combining this observation with the second observation, an edge (u, v) with a higher $w_G(u, v)$ should be included into T as early as possible and $p(v)$ should be set to u if $dist_G(u, sink) < dist_G(v, sink)$, and vice versa. We call this the ***highest-weight-first*** principle.

Based on above observations, we develop our algorithm DAT. Initially, DAT treats each node as a singleton subtree. Then we will gradually include more links to connect these subtrees together. In the end, all subtrees will be connected into one tree T. The detailed algorithm is shown in Algorithm 1, where notation $root(x)$ represents the root of the temporary subtree that contains x. To begin with, E_G is sorted into a list L in a decreasing order of links' weights. Based on the third

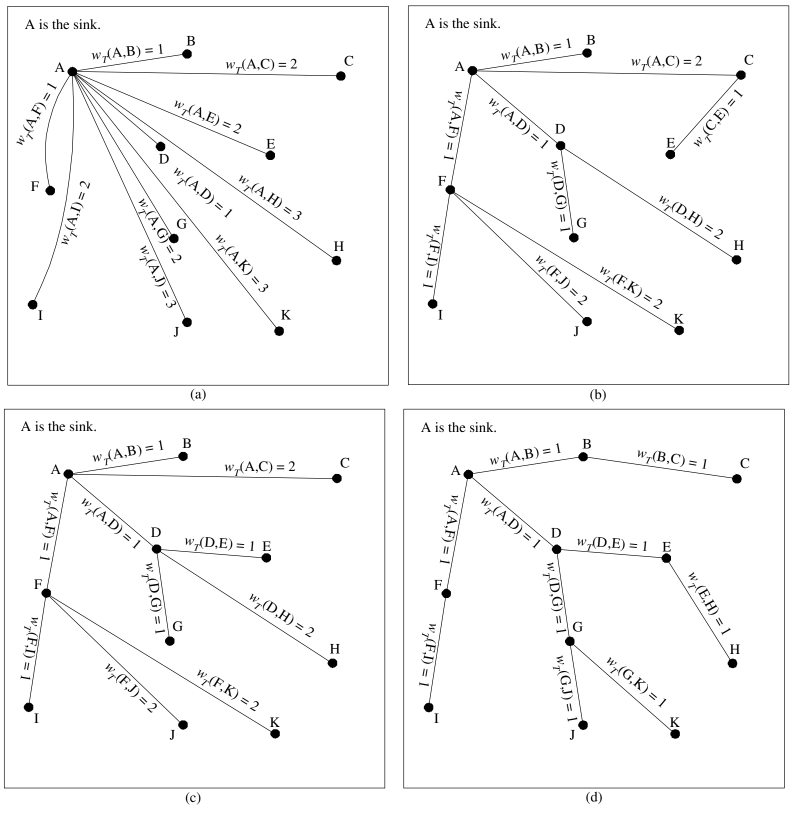


Figure 3: Four possible location tracking trees for the graph in Fig. 1(b).

observation, algorithm DAT will examine edges in L one by one for possibly being included into tree T. For each edge (u, v) in L being examined by algorithm DAT, (u, v) will be included into T only if u and v are currently located in different subtrees. Also, (u, v) will be included into T only if at least one of u and v is currently the root of its temporary subtree and the other is on a shortest path in G from the former node to the sink (these conditions are reflected by the ***if*** statements in lines 5 and 7). An edge in G passing these checks will then be included into T. Note that without these conditions, deviations may occur. It can be seen that T is always a subgraph of G and $w_T(u, v) = 1$ for all $(u, v) \in E_T$. For example, Fig. 4(a) is a snapshot of an execution of DAT. When (F, G) is examined by DAT, it will not be included into T, because neither F nor G is the root of its temporary subtree. Another snapshot is shown in Fig. 4(b). When (B, D) is examined, it will not be included into T. Although D is the root of its temporary subtree, B is not on the shortest path from D to A, i.e., $dist_G(D, A) \neq dist_G(B, A) + 1$. (A, D) will be then examined after (B, D). (A, D) can be included into T, because D is the root of its temporary subtree and A is on the shortest path from D to A.

Algorithm 1 DAT(G)

```
1:  Let $T = (V_T, E_T)$ such that $V_T = V_G$ and $E_T = \phi$
2:  Sort $E_G$ into a list $L$ in a decreasing order of their event rates.
3:  for each $(u, v) \in E_G$ in $L$ do
4:      if $(root(u) \neq root(v))$ then
5:          if $(u = root(u)) \wedge (dist_G(u, sink) = dist_G(v, sink) + 1)$ then
6:              Let $E_T = E_T \cup (u, v)$ and let the root of the new subtree be $root(v)$.
7:          else if $(v = root(v)) \wedge (dist_G(v, sink) = dist_G(u, sink) + 1)$ then
8:              Let $E_T = E_T \cup (u, v)$ and let the root of the new subtree be $root(u)$.
9:          end if
10:     end if
11: end for
```

Theorem 2. *If G is connected, the tree T constructed by algorithm DAT is a connected deviation-*

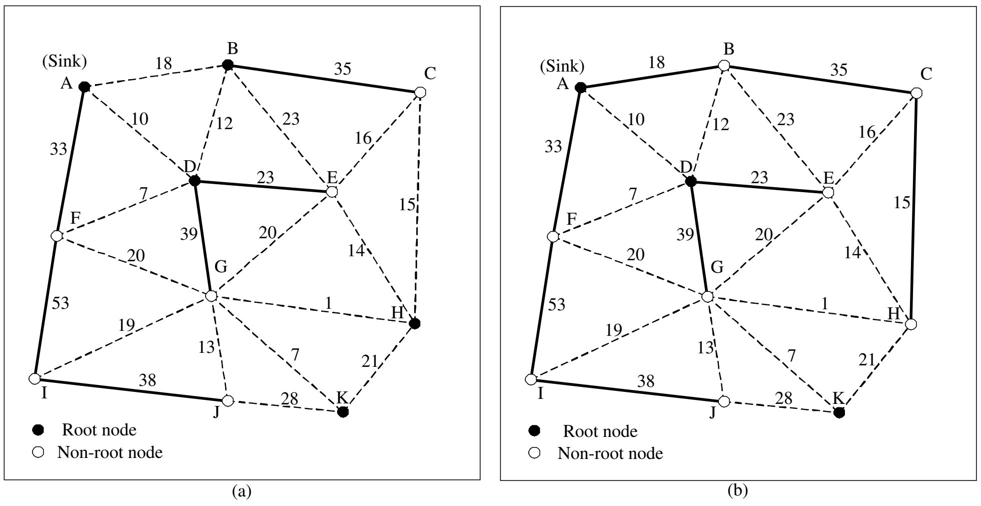


Figure 4: Snapshots of an execution of DAT. Solid lines are those that have been included into T.

avoidance tree rooted at the sink.

Proof. First, we show that T is connected. Each sensor is the root of a singleton subtree in the beginning and we will prove that only one senor will be the root in the ending. Since G is connected, when a sensor $x \neq sink$ is the root of a subtree (i.e., $x = root(x)$), it always can find a neighboring sensor y such that $dist_G(x, sink) = dist_G(y, sink) + 1$. It is clear that $root(y) \neq x$, because $dist_G(root(y), sink) \leq dist_G(y, sink)$. Hence, edge (x, y) can be included into T, and x will not be the root anymore. By repeating such arguments, T must be connected and rooted at the sink. Second, we show that T is a deviation-avoidance tree. This can be derived from two observations. First, when an edge (u, v) is included into T, DAT will choose v as the child of u if $dist_G(v, sink)$ is larger than $dist_G(u, sink)$, and vice versa. Therefore, if the path from the sink to sensor u is one of the shortest paths, the path from the sink to sensor v is also one of the shortest paths. Second, assuming $dist_G(v, sink) = dist_G(u, sink) + 1$, DAT will include (v, u) only when

v itself is the root of a subtree. This guarantees that all descendant nodes in $Subtree(v)$ will not deviate from their shortest paths to the sink. Hence, the theorem follows. $\square$

3.2 Algorithm Z-DAT (Zone-based Deviation-Avoidance Tree)

The Z-DAT is derived based on the following locality concept. Assume that u is v's parent in T. According to Eq. 2, for any edge $(x, y) \in E_G$ such that $x \in Subtree(v)$ and $y \notin Subtree(v)$, arrival/departure events between x and y will cause a message to be transmitted on $(p(v), v)$, thus increasing the value of $\sum_{(x,y) \in E_G \wedge x \in Subtree(v) \wedge y \notin Subtree(v)} w_G(x, y)$. Therefore, the perimeter that bounds the sensing area of sensors in each $Subtree(v)$ will impact the update cost $U(T)$. A longer perimeter would imply more events crossing the boundary. For example, in the three subtrees in Fig. 5, although all subtrees have the same number of sensors, the perimeter of the subtree in Fig. 5(a) is smaller than that in 5(b), which is in turn less than that in 5(c). In geometry, it is clear that a circle has the shortest perimeter to cover the same area as compared with other shapes. Circle-like shapes, however, are difficult to be used in an iterative tree construction. As a result, Z-DAT will be developed based on square-like zones.

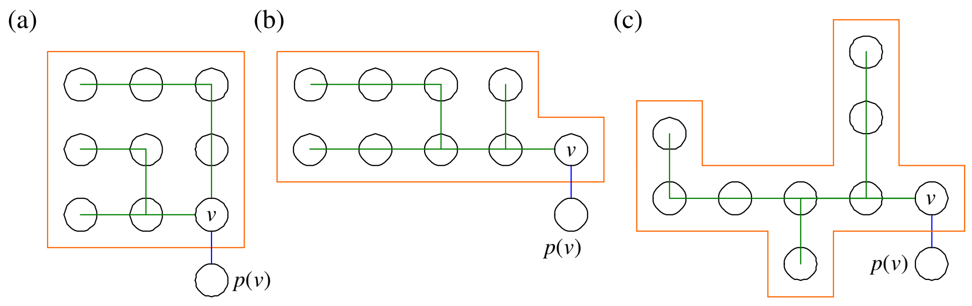


Figure 5: Possible structures of subtrees with nine sensors.

Z-DAT is derived based on the deviation-avoidance principle and the above locality concept. The algorithm builds T in an iterative manner based on two parameters, α and δ, where α is a power of 2 and δ is a positive integer. To begin with, Z-DAT first uses $(\alpha - 1)$ horizontal lines to divide the sensing field into α strips. For each horizontal line between two strips, we are allowed to further move it up and down within a distance no more than δ units. This gives $2\delta + 1$ possible locations of each horizontal line. For each location of the horizontal line, we can calculate the total event rate that objects may move across the line. Then we pick the line with the lowest total event rate as its final location. After all horizontal lines are determined, we then further partition the sensing field into α^2 regions by using $(\alpha - 1)$ vertical lines. Following the adjustment as above, each vertical line is also allowed to move left and right within a distance no more than δ units and the one with the lowest total event rate is selected as its final location.

After the above steps are completed, the sensing field is divided into α^2 square-like zones. First, we run DAT on the sensors in each zone. This will result in one or multiple subtrees in each zone. Next, we will merge subtrees in the above α^2 zones recursively as follows. First, we combine these zones together into $\frac{\alpha}{2} \times \frac{\alpha}{2}$ larger zones, such that each larger zone contains 2×2 neighboring zones. Then we merge subtrees in these 2×2 zones by sorting all inter-zone edges (i.e., edges connecting these 2×2 zones) according to their event rates into a list L and feeding L to steps $3 \sim 11$ of the original DAT algorithm. Second, we further combine the above larger zones together into $\frac{\alpha}{4} \times \frac{\alpha}{4}$ even larger zones, such that each even larger zone contains 2×2 neighboring larger zones. This process is repeated until one single tree is obtained. The algorithm is summarized in Algorithm 2. An illustrated example is shown in Fig. 6.

To summarize, Z-DAT is similar to DAT except that it examines links of E_G in a different order. By partitioning the sensing field into zones, each subtree in T is likely to cover a square-like region, thus avoiding the problem pointed out in Fig. 5. Also, by using the parameter δ to fine-tune the lowest-level zones, Z-DAT tends to avoid high-weight links becoming inter-zone edges. In fact, this is a consequence of the the highest-weight-first design principle.

Algorithm 2 Z-DAT(G, α, δ)

```
1: Divide the network into $\alpha \times \alpha$ zones based on parameters $\alpha$ and $\delta$.
2: Run DAT on the sensors in each zone.
3: $i \leftarrow 1$
4: while $\frac{\alpha}{2^i} \neq 0$ do
5:     The network is divided into $\frac{\alpha}{2^i} \times \frac{\alpha}{2^i}$ zones.
6:     Run DAT on each zone to merge its subtrees.
7:     $i \leftarrow i + 1$
8: end while
```

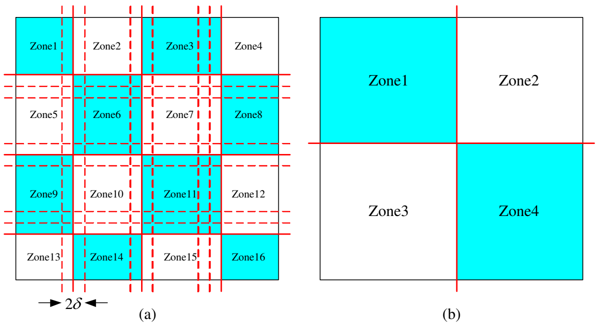


Figure 6: An example of the Z-DAT algorithm with $\alpha = 4$. (a) In the first iteration, we divide the field into $\alpha \times \alpha$ zones and adjust their boundaries according to δ. (b) In the second iteration, each 2×2 neighboring zones is combined into a larger zone.

Theorem 3. *If G is connected, the tree T constructed by algorithm Z-DAT is a connected deviation-avoidance tree rooted at the sink.*

Proof. Z-DAT will examine all links of G, but in a different order from DAT. However, the proof of Theorem 3 is independent of the order of the links being examined for being included into T. Therefore, the same proof is still applicable here. □

3.3 Algorithm QCR (Query Cost Reduction)

The above DAT and Z-DAT only try to reduce the update cost. The query cost is not taken into account. QCR is designed to reduce the total update and query cost by adjusting the object tracking tree obtained by DAT/Z-DAT. To begin with, we define the query rate $q(v)$ of each sensor v as the average number of queries that refer to objects within the sensing range of v per unit time in statistics.

Given a tree T, we first derive its query cost $Q(T)$. Suppose that an object x is within the sensing range of v. When x is queried, if v is a non-leaf node, the query message is required to be forwarded to v since $p(v)$ only indicates that x is in the subtree rooted at v. On the other hand, if v is a leaf node, the query message only has to be forwarded to $p(v)$, because sensor $p(v)$ knows that the object is currently monitored by v. The following equation gives $Q(T)$ by taking into account the number of hops that query requests and query replies have to travel on T.

$$Q(T) = 2 \times \left(\sum_{\substack{v \in V_T \wedge \\ v \notin \text{leaf node}}} q(v) \times dist_T(v, sink) + \sum_{\substack{v \in V_T \wedge \\ v \in \text{leaf node}}} q(v) \times dist_T(p(v), sink) \right), \qquad (3)$$

We make two observations on $Q(T)$. First, because $dist_T(p(v), sink)$ is always smaller than $dist_T(v, sink)$, Eq. 3 indicates that placing a node as a leaf can save the query cost instead of placing it as a non-leaf. For example, when query rates are extremely high, it is desirable that every node will become a leaf node and T will become a star-like graph. Second, the second term in Eq. 3 implies that the value of $dist_T(p(v), sink)$ should be made as small as possible. Thus, we should choose a node closer to the sink as v's parent (however, this is at the expense of the update cost).

Based on the above observations, QCR tries to adjust the tree T obtained by DAT or Z-DAT. In QCR, we examine T in a bottom-up manner and try to adjust the location of each node in T by the following operations.

1. If a node v is not a leaf node, we can make it a leaf by cutting the links to its children and connecting each of its children to $p(v)$. (Note that we can do so because T is regarded as a logical tree.) Let T' be the new tree after modification. We derive that

$$C(T) - C(T') = Q(T) - Q(T') + U(T) - U(T') = 2 \times \left(q(v) + \sum_{\substack{i \in children(v) \wedge \\ i \in \text{leaf node}}} q(i) \right) - \sum_{\substack{i \in neighbors(v) \\ \wedge i \in Subtree(v)}} w_G(x,i) - \sum_{i \in children(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(i) \\ \wedge x \in Subtree(i)}} w_G(x,y) \right) + \sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(v) \\ \wedge x \in Subtree(v) \wedge x \neq v}} w_G(x,y). \quad (4)$$

The derivation of Eq. 4 is in Appendix A. If the amount of reduction is positive, we replace T by T'. Otherwise, we keep T unchanged. Fig. 7 illustrates this operation.

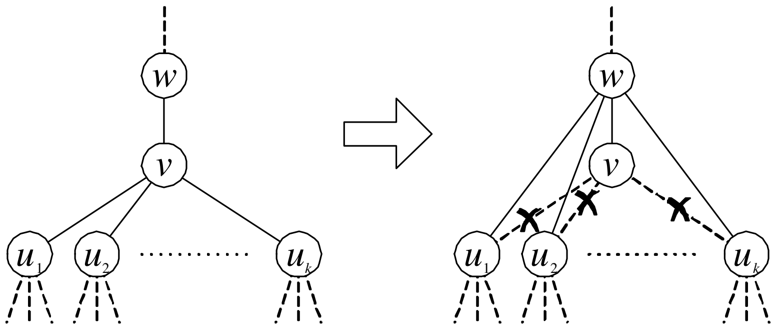


Figure 7: Making a non-leaf node v a leaf node.

2. If a node v is a leaf node, we can make $p(v)$ closer to the sink by cutting v's link to its current parent $p(v)$ and connect v to its grandparent $p(p(v))$. Let T' be the new tree. We derive that

$$C(T) - C(T') = Q(T) - Q(T') + U(T) - U(T') = 2 \times (q(v) + q'(v)) - \left(2 \times \sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(v) \wedge \\ x \in Subtree(v) \wedge y \in Subtree(p(v))}} w_G(x,y) \right), \quad (5)$$

where

$$q'(v) = \begin{cases} 0 & \text{if } p(v) \text{ has more than one child in } T \\ q(p(v)) & \text{otherwise} \end{cases}.$$

The derivation of Eq. 5 is in Appendix A. If the amount of reduction is positive, we replace T by T'. Otherwise, T remains unchanged. Fig. 8 illustrates this operation.

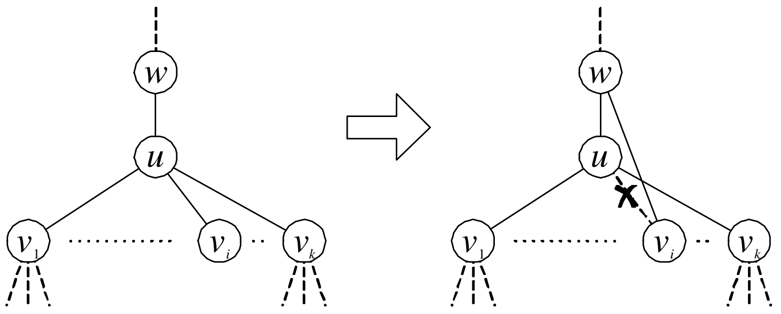


Figure 8: Connecting a leaf node v_i to $p(p(v_i))$.

Note that Eq. 4 and Eq. 5 allow us to compute the reduction of cost without computing $U(T')$ and $Q(T')$. This saves computational overhead. Also note that T is examined in a bottom-up manner in a layer-by-layer manner. Nodes that are moved to an upper layer will have a chance to be reexamined. However, to avoid going back and forth, nodes that are not moved will not be reexamined.

For example, suppose that we are given a DAT tree in Fig. 9(a) (which is constructed from Fig. 1(b)), where the number labelled on each node is its query rate. When examining the bottom layer, we will apply step 2 to sensors H, J, and K and obtain reductions of 1974, -62, and -6, respectively. Hence, only H is moved upward as shown in Fig. 9(b). When examining the second layer, we will apply step 1 to sensor G and I and apply step 2 to sensors C, E, and H. Only when applying to sensor H, it will result in a positive reduction of 1970. This updates the tree to Fig. 9(c). Finally, sensors B, D, and F are examined. Only D has a positive reduction of 1842.

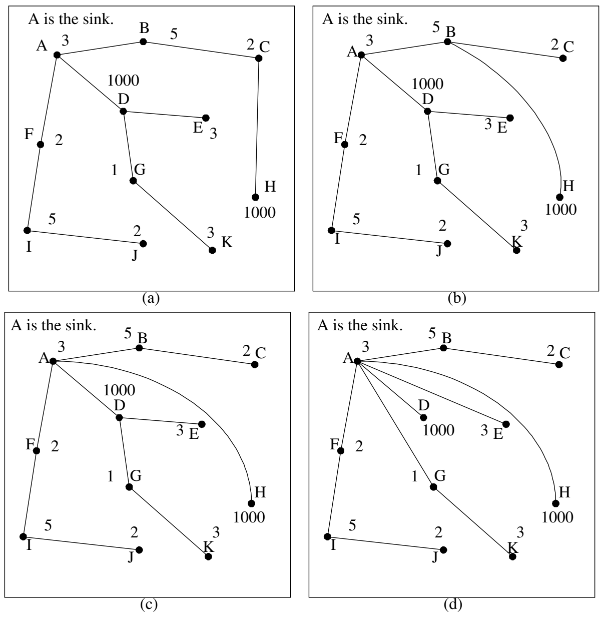


Figure 9: An execution example of algorithm QCR.

Thus, D will become a leaf and all its children are connected to D's parent as shown in Fig. 9(d). Overall, the cost is reduced from 7121 to 5150, 3180, and then 1338 after each step respectively.

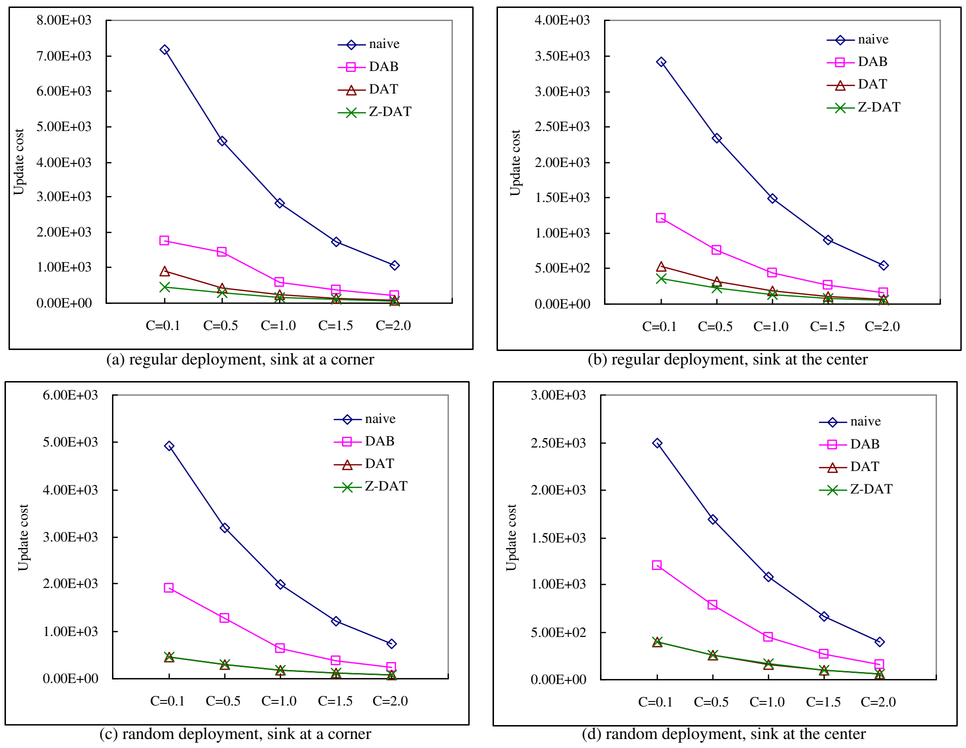
4 Simulation Results

We have simulated a sensing field of size 256×256. Unless otherwise stated, 4096 sensors are deployed in the sensing field. Two deployment models are considered. In the first one, sensors are regularly deployed as a 64×64 grid-like network. In the second model, sensors are randomly deployed. In both models, the sink may be located near the center of the network or one corner of the network.

Event rates are generated based on a model similar to the *city mobility model* in [8]. Assuming the sensing field as a square of size $r \times r$, the model divides the field into 2×2 sub-squares called *level-1* subregions. Each level-1 subregion is further divided into 2×2 sub-squares called *level-2* subregions. This process is repeated recursively. Given an object located in any position in the sensing field, it has a probability p_1 to leave its current level-1 subregion, and a probability $1 - p_1$ to stay. In the former case, the object will move either horizontally or vertically with a distance of $r/2$. In the latter case, the object has a probability p_2 to leave its current level-2 subregion, and a probability $1 - p_2$ to stay. Again, in the former case, the object will move either horizontally or vertically with a distance of $r/2^2$, and in the latter case it may cross level-3 subregions. The process repeats recursively. The probability p_i is determined by an exponential probability $p_i = e^{-C \cdot 2^{d-i}}$, where C is a positive constant and d is the total number of levels. In fact, the above behavior only formulates how objects move in the sensing field. After sensors are deployed in the network (no matter the sensors are deployed in a regular or random way), the movement patterns of these objects will generate event rates between neighboring sensors. Also, objects are queried by the sink with the same probability. Since objects may be located at different sensors with different probabilities, the query rates may vary in different sensors.

We compare our schemes with a naive scheme and the DAB scheme [8]. In the naive scheme, any update is sent to the sink (i.e., there is no in-network processing capability.) In this case, the query cost is always zero, so it is preferable when the query rates are relatively high. For the DAB scheme, all sensors are considered leaf nodes, and a logical structure is used to connect these leaf nodes. When two subtrees are merged into one, the root of the subtree which is closer to the sink will become the root of the merged tree (note that this may still cause deviation).

First, we observe the advantage of using in-network processing to reduce update cost. Fig. 10 shows the result under different values of C for regular and random sensor deployment. As can be seen, a larger C implies a higher moving locality, thus leading to a lower update cost. The naive scheme has the highest update cost, which is reasonable. By exploiting the concept of deviation



(a) regular deployment, sink at a corner (b) regular deployment, sink at the center

(c) random deployment, sink at a corner (d) random deployment, sink at the center

Figure 10: Comparison of update costs. In the Z-DAT scheme, $\alpha = 8$ and $\delta = 0$.

avoidance and taking the physical topology into account, DAT and Z-DAT further outperform DAB.

Next, we investigate the effect of deployment models. By comparing, the graphs in Fig. 10, we see that Z-DAT outperforms DAT under regular deployment, but the advantage is almost negligible under random deployment. This is because maintaining the shapes of subtrees in Z-DAT is difficult. For example, Fig. 11 shows snapshots of DAT trees and Z-DAT trees under regular and random deployments. As can be seen, Z-DAT does exploit the locality of sensors by partitioning sensors into zones under regular deployment. However, this is not true for the random case.

To get further insight into the performance of Z-DAT, we vary α and δ, and show the results in Fig. 12, where a 4096- and a 2500-node sensor networks are simulated. Note that when $\alpha = 1$

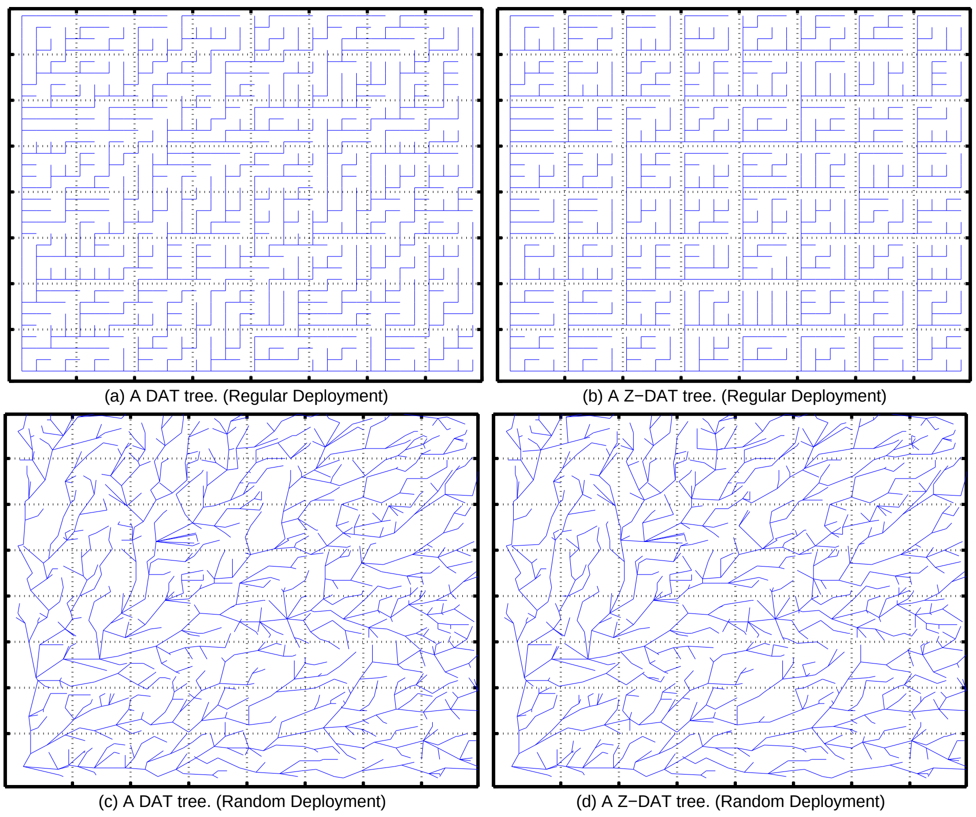

(a) A DAT tree. (Regular Deployment)
(b) A Z−DAT tree. (Regular Deployment)
(c) A DAT tree. (Random Deployment)
(d) A Z−DAT tree. (Random Deployment)

Figure 11: Snapshots of tree T obtained by DAT and Z-DAT under regular and random deployments. There are 1024 sensors with the sink at the lower-left corner. ($(\alpha, \delta) = (8, 0)$ for Z-DAT.)

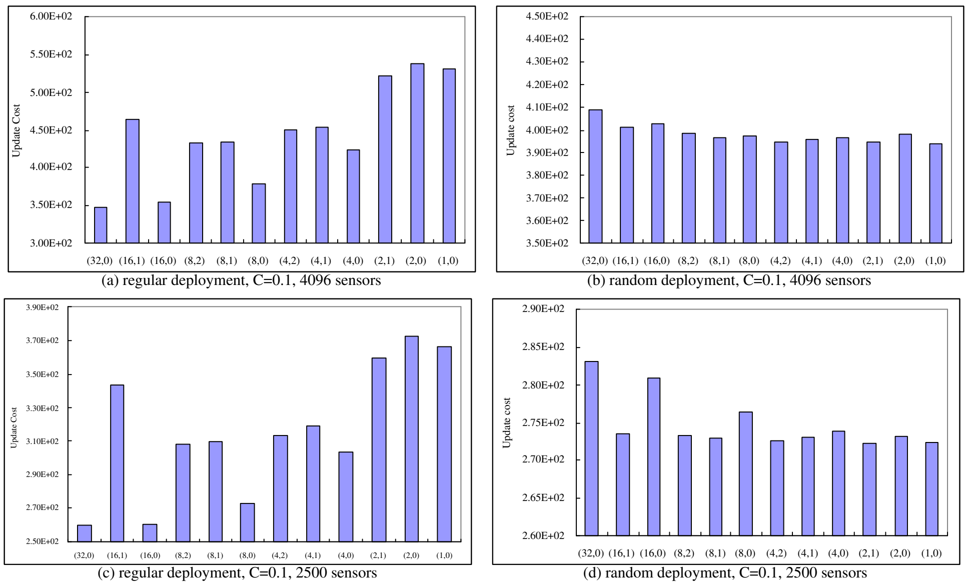
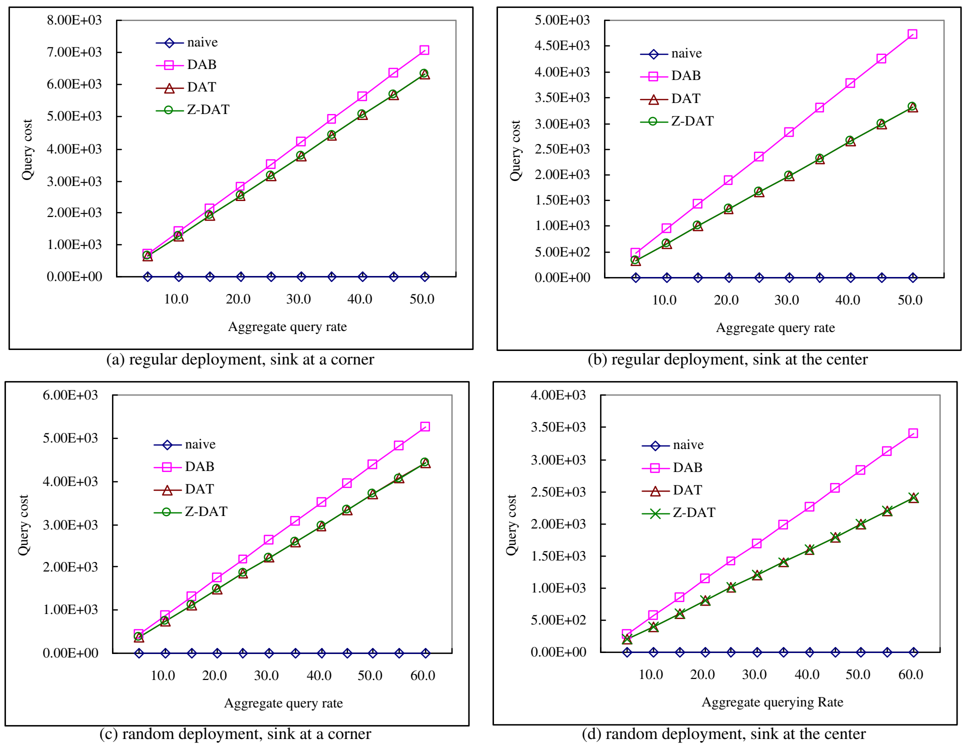


Figure 12: Comparison of update costs under different (α, δ) for Z-DAT. Sinks are located at the center of the network.

and $\delta = 0$, Z-DAT is equivalent to DAT. For regular deployment, Z-DAT performs well when α is larger than 4. However, for random deployment, the Z-DAT does not perform well, because maintaining the shapes of subtrees in Z-DAT is difficult. Furthermore, it can be seen that when $\delta = 0$, Z-DAT has better performance. This means that a square-like zone is better than a rectangle-like zone. Also, note that the trend in both 4096- and 2500-node sensors networks (the latter has a non-power-of-2 number of nodes) are quite similar.

Next, we examine the query cost. The result is shown in Fig. 13. In general, the query cost increases linearly with the aggregate query rate. As mentioned earlier, the query cost of the naive scheme is always zero. Both query costs for DAT and Z-DAT are lower than that of DAB. This is attributed to the fact that query messages are always transmitted along the shortest paths between the sink and sensors in DAT and Z-DAT. Also due to the similar reason, the query cost is independent of the shape of T; thus, DAT and Z-DAT perform similarly despite the deployment

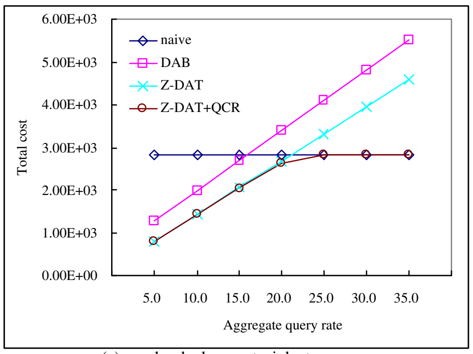


(a) regular deployment, sink at a corner (b) regular deployment, sink at the center (c) random deployment, sink at a corner (d) random deployment, sink at the center

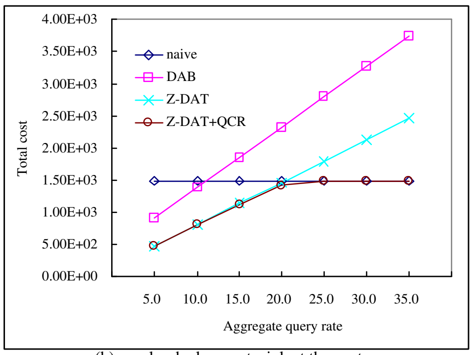
Figure 13: Comparison of query costs. ($C = 1.0$)

models.

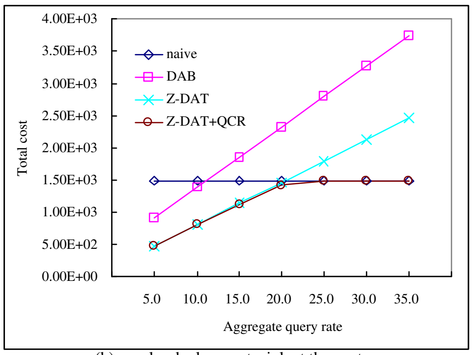
Finally, we examine the effectiveness of algorithm QCR by showing the total update and query costs of different schemes in Fig. 14. (For visual clarity, the cost of DAT are not shown.) The naive scheme has a constant cost because it is not affected by the query rate. The costs of DAB and Z-DAT increase linearly with respect to the query rate. As a result, they are outperformed by the naive scheme after the query rate reaches a certain level. Our Z-DAT with QCR scheme performs the best at all query rates. When the query rate is low, it performs close to Z-DAT. On the other hand, when the query rate increases, it works similar to the naive schemes. This verifies the advantage of the proposed DAT/Z-DAT with QCR schemes.



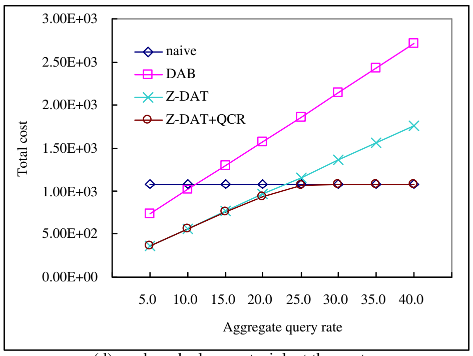
(a) regular deployment, sink at a corner



(b) regular deployment, sink at the center



(b) regular deployment, sink at the center



(d) random deployment, sink at the center

Figure 14: Comparison of total costs. ($C = 1.0$)

5 Conclusions

In this paper, we have developed several efficient ways to construct a logical object tracking tree in a sensor network. We have shown how to organize sensor nodes as a logical tree so as to facilitate in-network data processing and to reduce the total communication cost incurred by object tracking. For the location update part, our work can be viewed as the extension of the work in [8], and we enhance the work by exploiting the physical structure of the sensor network and the concept of deviation avoidance. In addition, we also consider the query operation and formulate the query cost of an object tracking tree given the query rates of sensors. In particular, our approach tries to strike a balance between the update cost and query cost. Performance analyses are presented with respect to factors such as moving rates and query rates. Simulation results show that by exploiting the deviation-avoidance trees, algorithms DAT and Z-DAT are able to reduce the update cost. By adjusting the deviation-avoidance trees, algorithm QCR is able to significantly reduce the total cost when the aggregate query rates is high, thus leading to efficient object tracking solutions.

References

[1] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, “Tracking a Moving Object with Binary Sensors,” in *Proc. of ACM SenSys*. ACM Press, November 2003.

[2] F. Aurenhammer, “Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure,” *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, September 1991.

[3] W.-P. Chen, J. C. Hou, and L. Sha, “Dynamic Clustering for Acoustic Target Tracking in Wireless Sensor Networks,” in *Proc. of IEEE International Conference on Network Protocols (ICNP)*, November 2003.

[4] D. Ganesan, R. Cristescu, and B. Beferull-Lozano, “Power-Efficient Sensor Placement and Transmission Structure for Data Gathering under Distortion Constraints,” in *Proc. of Int’l Workshop on Information Processing in Sensor Networks (IPSN)*, 2004.

[5] C.-F. Huang and Y.-C. Tseng, “The Coverage Problem in a Wireless Sensor Network,” in *Proc. of ACM Int’l Workshop on Wireless Sensor Networks and Applications (WSNA)*, September 2003.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks,” in *Proc. of 6th Annual International Conference on Mobile Computing and Networking (MobiCOM)*, August 2000.

[7] B. Krishnamachari, D. Estrin, and S. Wicker, “Modelling Data-Centric Routing in Wireless Sensor Networks,” in *Proc. of IEEE Infocom*, 2002.

[8] H. T. Kung and D. Vlah, “Efficient Location Tracking Using Sensor Networks,” in *Proc. of IEEE Wireless Communications and Networking Conference (WCNC)*, March 2003.

[9] S. Madden, R. Szewczyk, M. J. Franklin, and D. Culler, “Supporting Aggregate Queries over Ad-Hoc Wireless Sensor Networks,” in *Proc. of 4th IEEE Workshop on Mobile Computing and Systems Application*, 2002.

[10] K. Mechitov, S. Sundresh, and Y. Kwon, “Cooperative Tracking with Binary-Detection Sensor Networks,” *University of Illinois at Urbana-Champaign, Technical Report UIUCDCS-R-2003-2379*, 2003.

[11] G. J. Pottie and W. J. Kaiser, “Wireless Integrated Network Sensors,” *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[12] K. Sohrabi, J. Gao, V. Ailawadhi, and G. J.Pottie, "Protocols for Self-organization of a Wireless Sensor Network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16–27, October 2000.

[13] Y.-C. Tseng, S.-P. Kuo, H.-W. Lee, and C.-F. Huang, "Location Tracking in Wireless Sensor Network by Mobile Agents and Its Data Fusion Strategies," in *Proc. of Int'l Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.

[14] Y. Xu and W.-C. Lee, "On Localized Prediction for Power Efficient Object Tracking in Sensor Networks," in *Proc. of Int'l Workshop on Mobile Distributed Computing (MDC)*, May 2003.

[15] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communication*, vol. 3, no. 5, pp. 1689–1701, September 2004.

Appendix A

In this appendix, we show how to derive Eq. 4 and Eq. 5. To begin with, we present two implicit facts used in the following derivations. First, according to Theorem 1, we can conclude that if the members of $Subtree(v)$ are not changed, the number of messages transmitted on edge $(v, p(v)) \in T$ will be unchanged. Second, when a node v is being examined by QCR, $w_T(p(v), p(p(v)))$ must be 1. This fact holds because the input of QCR algorithm is a DAT/Z-DAT tree and the tree is examined in a bottom-up manner.

First, we derive the $Q(T) - Q(T')$ in Eq. 4. When v becomes a leaf and the queried object locates at the sensing field of v, the query only has to be sent to $p(v)$. In addition, when one of v's children, say i, is connected to $p(v)$ and i is a leaf, $p(v)$ also can reply the query if the queried object locates at the sensing field of i. Thus, we have

$$Q(T) - Q(T') = 2 \times \left(q(v) + \sum_{\substack{i \in children(v) \\ \wedge i \in leafnode}} q(i) \right).$$

Now we derive the $U(T) - U(T')$ in Eq. 4. The operation of QCR ensures that when one of v's children, say i, changes its parent to $p(v)$, the update cost will be increased by

$$\sum_{i \in children(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(i) \\ x \wedge \in Subtree(i)}} w_G(x, y) \right).$$

In addition, the events between v and i, where $i \in neighbors(v)$ and $i \in Subtree(v)$, will be reported to $p(v)$ rather than v when v becomes a leaf. Thus, v must forward an additional message to $p(v)$. The increased cost is

$$\sum_{\substack{i \in neighbors(v) \wedge \\ i \in Subtree(v)}} w_G(x, i).$$

However, when v becomes a leaf, the event across an edge $(x, y) \in E_G$ such that $y \notin Subtree(v)$, $x \in Subtree(v)$, and $x \neq v$ will not be transmitted on $(v, p(v))$. The cost is reduced by

$$\sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(v) \\ \wedge x \in Subtree(v) \wedge x \neq v}} w_G(x, y).$$

Combining above three factors, we have

$$U(T) - U(T') = -\sum_{\substack{i \in neighbors(v) \\ \wedge i \in Subtree(v)}} w_G(x, i) - \sum_{i \in children(v)} \left(\sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(i) \\ \wedge x \in Subtree(i)}} w_G(x, y) \right) + \sum_{\substack{(x,y) \in E_G \wedge y \notin Subtree(v) \\ \wedge x \in Subtree(v) \wedge x \neq v}} w_G(x, y).$$

Next, we derive Eq. 5. To see $Q(T) - Q(T')$, observe that when v changes its parent from $p(v)$ to $p(p(v))$, the saved query cost is $q(v)$. Furthermore, when $p(v)$ has only one child v, the adjustment of v will make $p(v)$ a leaf. This saves a query cost of $q(p(v))$. Therefore, we have

$$Q(T) - Q(T') = 2 \times (q(v) + q'(v)).$$

The value of $U(T) - U(T')$ is affected by three factors, when v changes its parent from $p(v)$ to $p(p(v))$. The update cost will be increased by

$$\sum_{\substack{(x,y)\in E_G \wedge y \notin Subtree(v) \\ \wedge x \in Subtree(v)}} w_G(x,y).$$

For edges that have one incident vertex in $Subtree(v)$ and one incident vertex is in $Subtree(p(v))$ but not in $Subtree(v)$, the events across these edges cannot be absorbed by $p(v)$ after v changes its parent from $p(v)$ to $p(p(v))$. The increased update cost will be:

$$\sum_{\substack{(x,y)\in E_G \wedge y \notin Subtree(v) \wedge \\ x \in Subtree(v) \wedge y \in Subtree(p(v)}} w_G(x,y).$$

However, for edges that have one incident vertex in $Subtree(v)$ and one incident vertex is not in $Subtree(p(v))$, the events across these edges will be transmitted on $(v, p(p(v)))$ rather than $(v, p(v))$ when we connects v to $p(p(v))$. The update cost will be decreased by

$$\sum_{\substack{(x,y)\in E_G \wedge x \in Subtree(v) \\ \wedge y \notin Subtree(p(v))}} w_G(x,y).$$

Combing these terms leads to the following equation

$$U(T) - U(T') = - \sum_{\substack{(x,y)\in E_G \wedge y \notin Subtree(v) \\ \wedge x \in Subtree(v)}} w_G(x,y) - \sum_{\substack{(x,y)\in E_G \wedge y \notin Subtree(v) \wedge \\ x \in Subtree(v) \wedge y \in Subtree(p(v))}} w_G(x,y)$$

$$+ \sum_{\substack{(x,y)\in E_G \wedge x \in Subtree(v) \\ \wedge y \notin Subtree(p(v))}} w_G(x,y) = - \left(2 \times \sum_{\substack{(x,y)\in E_G \wedge y \notin Subtree(v) \wedge \\ x \in Subtree(v) \wedge y \in Subtree(p(v))}} w_G(x,y) \right).$$