DIST: A Distributed Spatio-temporal Index Structure for Sensor Networks

Anand Meka and Ambuj K. Singh
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA.
{meka,ambuj}@cs.ucsb.edu

ABSTRACT

We consider the general problem of tracking moving objects in sensor networks. The specific application we consider is that of tracking a chemical plume moving over a large infrastructure network. We present a distributed index structure *DIST* that stores and updates distributed summaries as the plume moves. We present algorithms for range queries on the history of the plume. DIST localizes information with respect to time and space using a hierarchy that scales with the plume size. The highlight of our work is an *analytical model* to predict the cost of query algorithms based on the query location, query size, and plume's spatio-temporal distribution. Using this model, our adaptive scheme chooses the optimal scheme. Experimental results show that DIST outperforms alternative techniques in query, update, and storage costs, and scales well with the number of plumes.

1 Introduction

Consider the following historical event [2]: *The explosions that started the Chernobyl incident sent large amounts of radioactivity into the atmosphere. The wind carried the radioactivity towards Central Sweden. A plume reached Central Europe and another was blown over the Balkans. Monitoring stations around the world began intensive sampling programs.* During this disaster, governments needed to know the shape, location, and direction of the plume at every time instant. Monitoring sites processed the global plume information gathered from passive sensors.

Nowadays, organizations are deploying large scale sensor networks for real-time monitoring applications, such as military sensing and tracking, traffic control, and seismic measurements. A new generation of tiny, and computationally active sensors is being developed not only to overcome the scalability issues but also for deployment in inhospitable terrains. Though these sensors provide continuous monitoring, their limited resources (processing, communication, and storage) render the conventional data management tools and techniques ineffective.

Distributed protocols and algorithms [8, 16, 17] are being employed to store, process, and query high level semantically rich data from the individual, raw and noisy sensor signals. But, as the power consumption for communication is up to three orders of magnitude higher than that for computation on a sensor node (such as a Crossbow Mica2 mote [1]), a goal of sensor research has been to lower data transmission while designing such distributed techniques.

We develop distributed techniques for answering queries regarding the movement of an object in a sensor network. Our query of interest is: *Was the object present in a given region during a given time interval?* Such a *spatio-temporal range query* should return all object locations where the object's spatial extent overlaps with the query's spatial range and its temporal duration overlaps with the query's temporal range.

Specifically, we consider the case of a chemical plume moving over a sensor network. This is more challenging since a plume's shape can change in addition to its translations and rotations. Sensors collaborate to detect the shape of a plume periodically. As the plume moves, the movement has to be sensed, and the location of the plume should be updated. The resulting spatio-temporal summary of a plume's movement can be used to answer queries regarding its location at different time intervals, e.g., *which plume contours were present in a given region during a given time period?*

We focus on efficient distributed indexing of a plume's movement in order to answer such queries with a small communication overhead. We assume that each sensor is stationary and knows its own location and that of its neighbors [11, 16]. Our contributions are:

1. We address the important problem of plume tracking (in general, tracking of a mobile object) in a sensor network. We present efficient algorithms for storing summaries, and for answering spatio-temporal range queries using a distributed index structure with a small communication overhead.
2. We design an analytical model to evaluate the expected cost of our query algorithms based on the query location, query size, and plume distribution. Using this model, our adaptive scheme chooses the optimal scheme on a per-query basis.
3. Our experimental results on real-world plume mobility-models indicate that DIST performs two to three times better than alternative techniques in update, querying, and storage costs, and scales well with the number of plumes.

2 Spatio-temporal Indexing

The sensor network is hierarchically decomposed into levels and a quad-tree like partitioning (called *cells*) at each level. A distributed indexing scheme exploits the plume's locality in space and time using a hierarchical index. The movement of a plume is observed at different spatial resolutions: indices at a level of resolution are updated *only* if the movement occurs across cells at that or a coarser level. Next, we discuss the spatial decomposition of a network into cells, and the propagation of location information of a plume as it moves through the network.

2.1 Spatial decomposition of the network

The key idea behind *DIST* is to observe the movement of the plume at different spatial and temporal scales. To this end, the entire topology is decomposed into *cells* at different spatial resolutions (similar to a quad-tree). Consider the bounding rectangle $[L \times L]$ of the entire N node network on the x-y plane. A cell is obtained by dividing the network recursively into four quadrants. The number of recursive divisions defines the *level* of a cell. The root node is at level 0 and ϵ denotes the highest level of decomposition. An example cell structure is illustrated in Fig. 1. We assume a dense deployment of sensors at level ϵ, i.e., one or more sensors in every cell at level. Every cell has a leader. Leader election is done with the help of a localized and scalable hierarchical leader election algorithm [8][1]. The leader of a cell c is denoted by $l(c)$. A hierarchical structure is imposed by connecting a cell leader to its four child quadrant leaders. All the actions at a cell are executed by its leader. In the development of our algorithms, we blur the distinction between a cell and its leader.

Depending on its current size, a plume is mapped to a specific level of the hierarchy; this level is denoted by $\alpha\ (\le \epsilon)$. If D denotes the diameter of the axis-aligned minimum bounding rectangle of the plume, then $\alpha = i$ if $L/2^{i+1} < D \le L/2^i$. A plume can also be mapped to a specific set of cells at level α that contains it; this set is denoted by $\mathcal{S}$. The size of set $\mathcal{S}$ is more than one if the plume straddles more than one cell. Note that $|\mathcal{S}| \le 4$. Since α and $\mathcal{S}$ can change dynamically, we index them by a time instant, as in $\alpha(t)$ and $\mathcal{S}(t)$. We make two simple continuity assumptions about the plume movement for simplicity of presentation. First, α does not change by more than one in two consecutive time instants, i.e., $|\alpha(t+1) - \alpha(t)| \le 1$. Second, the plume does not skip across the neighbors of a cell between two consecutive time instants, i.e., $\alpha(t) = \alpha(t+1) \wedge c \in \mathcal{S}(t+1) \wedge c \notin \mathcal{S}(t) \Rightarrow (\exists d \in N(c) :: d \in \mathcal{S}(t))$, where $N(c)$ denotes the neighbors of cell c.

Observe the motion of the plume in Fig. 1. For the range query $q : [42, 65] \times [42, 48] \times [t_5, t_{11}]$ shown in the figure, its spatial range overlaps cells F, G, and H. The plume was present in cell F at time instants t_8 and t_9 and in cell G at time instants t_{10} and t_{11}. These time instants are within the query's time interval $[t_5, t_{11}]$. Therefore, any algorithm should return the plume summaries stored in F and G for this query.

2.2 Shape summaries & update propagation

Every leader stores an index or a set of disjoint time intervals over which the plume was inside its cell. Each time interval has a *begin* and an *end* time instant. If the plume is currently in a cell, then its most recent interval has ∞ as the *end* value. Such an index is called *open*. Otherwise, an index is called *closed*. We maintain the following invariant: (*Invariant I*) an index at a cell (leader) is open *iff* it is on the path from the root to a cell belonging to $\mathcal{S}$.

We do not address the problem of detecting and summarizing the shape of a plume. We rely on the current boundary estimation and edge detection methods [3] for this task. We assume that a plume's shape is continuously tracked and stored at specific sensor nodes called *repository nodes*. These nodes communicate with the cell leaders after gathering plume information.

Information about a plume's location is maintained as follows. At each time instant t, a repository node senses the plume and com-

[1]For query routing, the node closest to the centroid of a cell is elected as its leader.

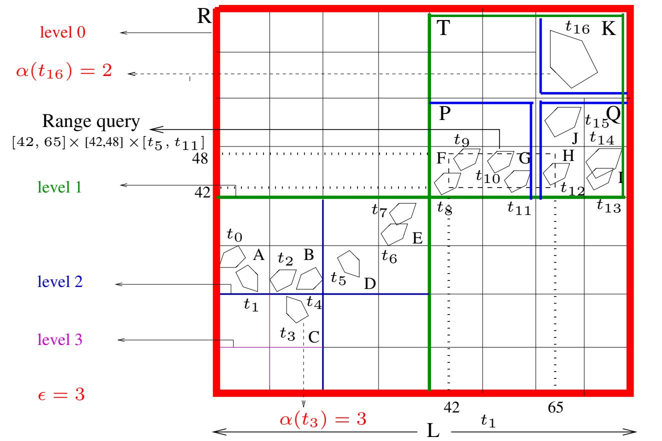

Figure 1: Movement of a plume across cells at various levels from time instants t_0 to t_{16}. A, B, . . . , J represent level 3 cells. P, Q, K are level 2 cells. T is a level 1 and R is a level 0 cell.

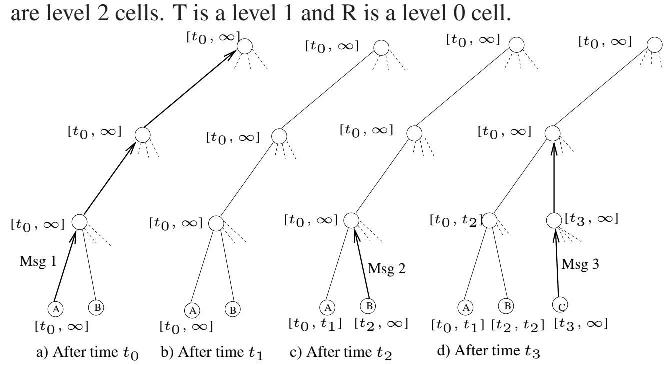

Figure 2: Update propagation for the plume movement shown in Fig. 1. Every node represents a cell leader.

```
▷ Update propagation at node x (0 ≤ x ≤ N)

// ShapeList : List of shape summaries stored at a node.
// C : An index at a node.
// IndexList : A list of indexes at a node.

upon detecting the shape (at a repository_node):
    ShapeList ← ShapeList ∪ ⟨shape, current_time⟩
    z ← leader of cell at level h
    send ⟨update; current_time, x⟩ to z

upon receiving ⟨update; t, y⟩:
    if plume was in the cell at t − 1
        if y is a repository_node
            C.list ← C.list ∪ {y}
    else
        C.start_time ← t
        send ⟨update; t, x⟩ to parent
        for each neighboring cell leader n
            send ⟨set_end_time; t⟩ to n

upon receiving ⟨set_end_time; t⟩:
    if plume was in the cell at t − 1 but not at t
        C.end_time ← t − 1
        IndexList ← IndexList ∪ {C}
    else send the msg a level up and a level down.
```

Figure 3: Propagation of updates.

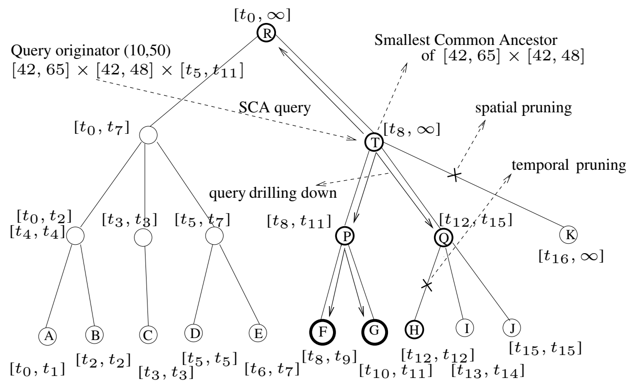

Figure 4: *SCA* query trace for the example in Figure 1.

putes $\alpha(t)$ and $\mathcal{S}(t)$. It then sends a message (id, t) to the leader of each cell $c \in \mathcal{S}(t)$, where id is the repository node id. The indices at all these nodes and their parents need to be open in order to satisfy invariant I. Node $l(c)$ stores the pair (id, t). If the index at cell c was open at $t-1$ (or $c \in \mathcal{S}(t-1)$), node $l(c)$ does nothing else. Otherwise, $l(c)$ inserts a new time interval into its index, and propagates the received information to its parent and the neighboring cells. If necessary, the parent uses this information to insert a new time interval (recording of the id is not needed) and propagates the information up the tree. Any neighboring cell d of cell c that had an open index at time $t-1$ and that does not belong to $\mathcal{S}(t)$ ends its most recent time interval by inserting $t-1$. It also propagates the information up to the root so that its parent can also take the appropriate actions in order to maintain the invariant. If none of the neighboring cells had an open index at time $t-1$, then this implies that the level of the plume has changed (our continuity assumptions). In this case, the tree is searched a level up and a level down to determine $\alpha(t-1)$ and $\mathcal{S}(t-1)$. The indices at the cells belonging to $\mathcal{S}(t-1) - \mathcal{S}(t)$ are closed.

The number of update messages that are sent at a given time instant t depend on the changes in α and $\mathcal{S}$. If neither of them changes, updates are not generated. If $\mathcal{S}$ changes but α does not, then update messages are confined to a small part of the tree. Changes in α lead to more messages; however, these events are relatively infrequent. The index at a cell leader at level k maintains plume information about all those instants when the plume was at a level at least k. The stored set of (id, t) values, on the other hand, maintains intersection information about all those instants when the plume was at level k. The complete update algorithm is presented in the full paper [18].

For the movement of the plume shown in Fig. 1, we obtain the execution trace shown in Fig. 2. At time instant t_0, a repository node p senses that $\alpha = 3$ and $\mathcal{S} = \{A\}$, and sends the message (p, t_0) to leader $l(A)$. Node $l(A)$ stores this tuple, and inserts the interval (t_0, ∞) in its index, and propagates the message to its parent. Since this is the initial message, it propagates all the way up to the root, and every leader on the way also sets its index to (t_0, ∞). Fig. 2(a) shows the resulting tree. At time instant t_1, a repository node again sends a similar message to $l(A)$. Since $A \in \mathcal{S}(t_0)$, $l(A)$ does not propagate any update messages to its parent, as shown in Fig. 2(b). At time instant t_2, the plume has moved to a new cell B. A repository node q senses that $\mathcal{S} = \{B\}$, and sends the message (q, t_2) to node $l(B)$. Node $l(B)$ opens a new interval, and propagates the message to its parent and to every neighboring cell leader. As a result, node $l(A)$ closes its open interval and propagates the message up the tree. The parent does nothing as its index was open at t_1, as shown in Fig. 2(c). Fig. 2(d) shows the tree at time t_3.

3 Range Query Algorithms

In this section, we first describe two basic range query algorithms. The *Smallest Common Ancestor (SCA)* algorithm uses the hierarchy to prune spatial and temporal extents of a query. The other algorithm, *Direct*, directly queries the cells within the spatial extent of the query at level ϵ and its ancestors. Then, we discuss an *adaptive* algorithm that uses cost estimates to invoke either the *SCA* or the *Direct* scheme on a per-query basis.

3.1 SCA query algorithm

The query originator uses its knowledge of the quad-tree like network cell structure to determine the spatial cells at the level ϵ that are intersected by the query. From these cells and the spatial tree hierarchy, it determines the smallest common ancestor *sca* of these cells. It then transmits the query to the *sca* using an underlying location-based communication model such as GPSR[2] [11], a greedy geographic routing protocol. For example, for the query $q : [42, 65] \times [42, 48] \times [t_5, t_{11}]$ shown in Fig. 1, the querying node at location $(10, 50)$ decomposes the query's spatial extent into cells F, G, and H, and determines that cell T is the *sca* of these cells.

Upon receiving the query message, the *sca* needs to accumulate spatio-temporal intersection information from all levels in the tree. In order to find all those instants when the intersecting plume was at a lower level than the level of *sca*, the query message is sent up the tree. An ancestor node evaluates if its index intersects with the query's time interval. If so, it uses the stored (id, t) pairs to find the pairs that intersect temporally with the query. It then checks with the relevant repository nodes for exact spatial intersections. The *sca* cell leader then finds intersecting plume contours at its own level by consulting the stored (id, t) pairs in an analogous manner. Finally, the *sca* evaluates if the query's time interval overlaps with any of the time intervals in its index. If so, it propagates the query to those children whose spatial extent overlaps the query's spatial range. This forwarding down the tree is carried on recursively, pruning in both the spatial and temporal domains simultaneously until the query reaches level ϵ. All relevant repository nodes transmit the plume contours back to the querying node.

Fig. 4 shows the query q's trace after it is transmitted to the leader of cell T, the *sca* of the query. A message is sent up the tree to parent R. There are no stored (id, t) pairs corresponding to plumes either at R or at T. Then, the downward propagation of the query begins. As the query drills down, we observe that the branch to K is pruned spatially, whereas the branch from Q to H is pruned temporally. The query finally reaches nodes F and G at level ϵ. These nodes now forward the query to all the the relevant repository nodes. This scheme not only employs the pruning advantages of a tree, but also alleviates any root node bottlenecks since queries can begin at different levels in the tree.

Sometimes, domain knowledge can be used to infer bounds (α_l, α_h) on the values of α. In such cases, propagation up the tree from the *sca* stops when level α_l is reached and propagation down the tree stops when level α_h is reached. For example, under some mobility models, a plume only rotates and translates but retains its size (i.e, $\alpha_l = \alpha_h = k$ where k is a constant). If the plume is known to be at a specific level k, then the searching for intersecting plumes is carried out from the *sca* down to level k.

[2]GPSR delivers the packet to the node closest to the destination location. As a cell leader is the node closest to the cell centroid, GPSR ensures that a packet destined for the centroid is delivered to the leader. A query originator locally computes the centroid of the *sca* cell and employs GPSR to transmit the query to the *sca* leader.

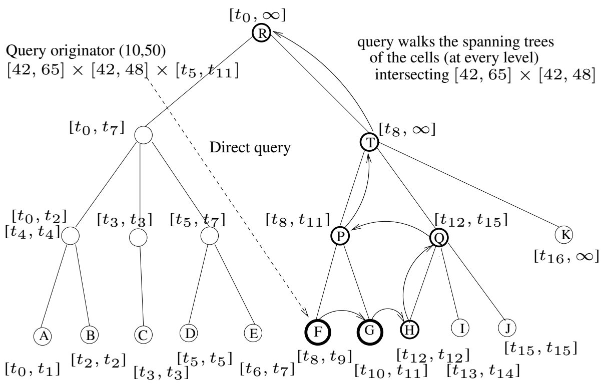

Figure 5: Direct query trace for the example in Figure 1.

3.2 Direct query algorithm

In the *Direct query* algorithm, the query originator decomposes the query's spatial extent into cells at level ϵ (as in the SCA scheme), and directly queries these cells and all their ancestors. All levels in the query's spatial extent need to be queried since there is no *a priori* knowledge regarding a plume's level, and a cell at level k maintains intersection information for only a level k plume. This algorithm is useful when domain knowledge limits the plume to a few levels. In that case, only cells within the query's spatial extent at the relevant levels need to be queried. Specifically, if the plume is limited to just one level k, then only cells at level k that are within the query's spatial extent are queried. If α_h is known *a priori*, the query can be decomposed into cells at level α_h instead of ϵ. The cost of maintaining α_h is amortized if a plume's level changes infrequently.

The propagation of a query to the relevant cells in the hierarchy can be made efficient by constructing a spanning tree (*ST*) at each level. Consider each cell leader within the query's spatial extent to be a *vertex* and place an *edge* between two such vertices if they are physically adjacent. In this manner, the query originator constructs a communication graph and finds a *ST*. The query is propagated along the *ST* at each level and then transmitted to the next level. Each leader on the *ST* forwards the query to those repository nodes that intersect temporally with the query. All the results from the repository nodes are sent back to the querying node.

For example, the query $q : [42, 65] \times [42, 48] \times [t_5, t_{11}]$ in Fig. 1 is decomposed into cells F, G, and H at level $\epsilon = 3$. The graph has a vertex for each cell leader F, G, and H, and edges $F \rightarrow G$ and $G \rightarrow H$ because F and G are physically adjacent, as are G and H. In this particular case, the *ST* is the original graph itself. The *ST* at level 2 is the edge $Q \rightarrow P$; at level 1, it is the node T; at level 0, it is the node R. Fig. 5 shows how the above query is handled. The originator at location $(10, 50)$ routes the query to the leader of cell F, the closest node in the *ST* at level ϵ. The leader of F forwards the query to the leader of G, and the latter forwards it to the leader of H. Now, since F and G's temporal ranges are within the query's temporal range, the leaders of these cells transmit the query to the relevant repository nodes in their cells, whereas the leader of H does not. Fig. 5 shows how the query is recursively sent up to the *ST* at the next level.

3.3 Adaptive querying

Both the *SCA* and Direct query algorithms have their advantages and disadvantages. In the case of a query with a large spatial range, the *Direct* query algorithm does not scale well, since it is propagated to a large number of cells. Here, *SCA* is effective as it prunes the search space in both space and time. But, for a plume with a few levels, and a query with a small spatial extent that overlaps multiple cells, *SCA* performs poorly as the smallest common ancestor (*sca*) lies high up in the hierarchy leading to an expensive drill-down operation. Direct querying ameliorates this effect by forwarding the query only to the small number of relevant cells. It is possible to overcome the disadvantages of both the schemes by adopting the better of the two schemes depending on the query location, query size, and plume distribution on a per-query basis. This motivates the adaptive query scheme presented next.

We build an analytical model to compute the expected communication costs of the *SCA* and Direct query algorithms. Using this model, the query originator locally computes the expected costs of Direct query, $\lambda_d(q)$, and of *SCA* query, $\lambda_s(q)$, and then chooses the scheme with the lower cost for the query q denoted by $q : (x_l, x_r) \times (y_b, y_t) \times (t_1, t_2)$.

The expected cost of a query q in Direct query, $\lambda_d(q)$, can be decomposed into the expected cost of sending the query from the originator to the closest node on the spanning tree at level ϵ, $IC_d(q)$, the expected cost of traversing the spanning trees at different levels for the query, $\phi_d(q)$, and the expected cost of sending the query from the spanning tree nodes to the repository nodes, $\psi_d(q)$.

$$\lambda_d(q) = IC_d(q) + \phi_d(q) + \psi_d(q) \tag{1}$$

The expected cost of a query q in the *SCA* query, $\lambda_s(q)$, can be decomposed into the expected cost of sending the query from the originator to the *sca*, $IC_s(q)$, the expected cost of traversing the hierarchical tree from *sca* up and down the tree, $\phi_s(q)$, and the expected cost of sending the query to the repository nodes, $\psi_s(q)$.

$$\lambda_s(q) = IC_s(q) + \phi_s(q) + \psi_s(q) \tag{2}$$

Since both the query algorithms are guaranteed to return all the answers for a query, $\psi_d(q) = \psi_s(q)$. So, from Equations 1 and 2, we get

$$\lambda_d(q) - \lambda_s(q) = (IC_d(q) + \phi_d(q)) - (IC_s(q) + \phi_s(q)) \tag{3}$$

The query originator, using its own location and inspecting q's spatial extent, determines the closest node on the spanning tree at level ϵ and hence computes $IC_d(q)$. Similarly, *sca* is determined using q's location and spatial extent; this is used to compute $IC_s(q)$. The level of the plume is constrained to be in the range $[\alpha_l, \alpha_h]$. If these values are not known *a priori* at query time, then $\alpha_l = 0$ and $\alpha_h = \epsilon$.

Next, we consider the computation of $\phi_d(q)$. The width w_h of a level h cell is $L/2^h$. The number of cells n_h at level h intersected by query q, is given by

$$n_h = (\lceil x_r/w_h \rceil - \lfloor x_l/w_h \rfloor) * (\lceil y_t/w_h \rceil - \lfloor y_b/w_h \rfloor) \tag{4}$$

The cost of traversing an edge of spanning tree, i.e., the number of hops to travel between two adjoining cell leaders[3] at level h, is w_h/r, where r is the communication radius. Let this be denoted by c_h. Since the number of edges in the spanning tree is one less than number of nodes, the contribution to $\phi_d(q)$ by nodes at level h is

$$\phi_d(q, h) = (n_h - 1) * c_h \tag{5}$$

Summing this cost over all levels, along with the additional inter-level communication costs we obtain

$$\phi_d(q) = \sum_{h=\alpha_l}^{\alpha_h} \phi_d(q, h) + \sum_{h=\alpha_l}^{\alpha_h - 1} c_h/\sqrt{2} \tag{6}$$

[3]This holds for dense sensor networks [21]. For accurate estimations it should be replaced by w/L_{hop} [15], where L_{hop} is the average euclidean distance covered per hop. $L_{hop} = r\,\alpha(\rho)$ where ρ is the density of the nodes and $0 \leq \alpha(\rho) \leq 1$.

Finally, we analyze $\phi_s(q)$. Let $p(x, y, t)$ be a spatio-temporal density function that assigns the probability of plume membership to a point (x, y) at time t. Then,

$$\int_{\text{All } X \times Y} p(x, y, t) \, \mathrm{d}x \, \mathrm{d}y = 1. \tag{7}$$

Let the probability that the point (x, y) is inside the plume during the time interval $[t_1, t_2]$ be given by $p(x, y, [t_1, t_2])$. Let $P_{i,h}[t_1, t_2]$ denote the probability that the plume was at a cell i at level h during the time interval $[t_1, t_2]$. Now, if cell i has an extent $[x_1, x_2]$ and $[y_1, y_2]$ then:

$$P_{i,h}[t_1, t_2] = \int_{x_1}^{x_2} \int_{y_1}^{y_2} p(x, y, [t_1, t_2]) \, \mathrm{d}x \, \mathrm{d}y \tag{8}$$

In $\phi_s(q)$, the cost of traversing up the tree from the *sca* node is determined only by the level k of the *sca*, whereas the cost of drilling down the tree from the *sca* depends on the probability of intersection of the plume with every cell on the path. Hence,

$$\phi_s(q) = \sum_{h=\alpha_l}^{k-1} c_h/\sqrt{2} + \sum_{h=k}^{\alpha_h - 1} \sum_{i \in V_h} P_{i,h}[t_1, t_2] \cdot |\xi(i)| \cdot c_h/\sqrt{2} \tag{9}$$

where V_h is the set of cells at level h that intersect the query and $\xi(i)$ is set of children of cell i that belong to V_{h+1}. The first term is the cost of propagating up the tree. The second term is the cost of drilling down the tree: at every level h, we multiply the probability that the cell $i \in V_h$, intersects the plume during the query time interval (t_1, t_2), $P_{i,h}[t_1, t_2]$, with the cost of propagating the query from cell i to all its children that intersect the query $(|\xi(i)| \cdot c_h/\sqrt{2})$.

The computation of $p(x, y, t)$ depends on the plume mobility model. A rich body of literature has shown the efficacy of two such stochastic models, the *cloud model* [6], and the *Gaussian plume model* [7], in resembling real-world distributions. Equations 8–9 have a closed form solution for the cloud model. For the Gaussian model, numerical integration techniques are employed. The details are discussed in the full paper [18].

In summary, the adaptive query protocol followed by the query originator is:

1. Use equations 4–6 to compute $\phi_d(q)$.
2. Use equations 8–9 to compute $\phi_s(q)$.
3. Use equation 3 to find the optimal scheme.

Profile analysis [18] showed that the computational cost of this analytical model is up to three orders of magnitude smaller than the accrued communicational benefits for the MICA2 mote [1].

3.4 Fault tolerance

Robustness to node failures is an important design consideration. Absence of periodic beaconing messages from neighboring nodes within the radio range can detect node failures. DIST can handle node failures by two kinds of replication: *local* replication where the index information is replicated randomly on a node (called the *sibling*) within a radius of two hops; *mirror* replication for resilience towards concurrent failure of contiguous nodes where the data is stored on geographically complementary nodes (as proposed in DIMS [16]). Both the *sibling* and *mirror* nodes also store the original node's parent and child ids. Updating of indices can be made resilient to packet losses by a simple ACK mechanism. In order to handle query (and response) packet losses, the query originator can set a conservative timeout, after which the querying node *selectively* retransmits the queries. If a node has decided that its cell leader has died then it can either elect the *sibling* as the new leader or reinitiate the leader election algorithm [8]. The new leader can restore all the index information from the *sibling* or *mirror* nodes.

4 Performance Evaluation

In this section, we compare the performance of our index structure and query algorithms against alternative choices for implementing spatio-temporal range queries. We evaluate the performance in terms of average query and update costs. Query cost is the average number of messages required to route a query to the relevant nodes in the network. Update cost is the number of messages required to update the distributed index as a plume moves in the network. Even though there are currently no existing spatio-temporal indexing schemes in sensor networks, we present two alternatives.

- **Geocast [12]:** *Geocast* is a location-based multicast approach. It does not maintain any index structure. A range query is forwarded to the group of nodes within the query's spatial range. Nodes that have the requisite data reply back. Therefore, the algorithm incurs an insertion cost of 0 and a query cost of $O(\sqrt{N} + |Q_s|)$, where $|Q_s|$ is the query's spatial extent and N is the network size.

- **DIMS [16]:** Another alternative is to extend DIMS [16], a multi-dimensional index structure for sensor networks. DIMS hashes multi-dimensional events to the nodes in the network. So, in order to hash a plume (an area), the repository node decomposes its area into squares of size Q_l (lower bound on the query size), and hashes the (centroid, time instant) to a node in the network. A range query is similarly hashed to a region of nodes and is routed to the leader of that region. Then, it is progressively split into smaller sub-queries and sent along subtrees. Leaf nodes contact the repository nodes and relevant plumes are subsequently returned to the query originator. The insertion cost is $O(\frac{A}{Q_l}\sqrt{N})$, where A is the area of the plume, and the query cost is $O(\sqrt{N} + |Q|)$, where $|Q|$ is the size of the mapped region resulting from the hashing of the query.

Next, we present the simulation and mobility models, and then present the results of an extensive evaluation of our techniques against the alternatives in terms of update, query and storage costs.

4.1 Simulation and mobility models

We built a discrete event simulation model. Experiments were conducted at network sizes ranging from 1000 to 7000 nodes and node densities ranging from 0.6 to 0.8. Nodes were placed randomly with a uniform distribution. Each node has on the average 4 nodes within its radio range. Two nodes in the network communicate with each other using GPSR [11]. We considered two stochastic models for the plume's mobility:

- In a non-dispersive model called the *cloud* model [6], the center of mass of the plume performs a random walk. The starting point of the cloud is chosen from a stationary distribution. Its translation along the two axes is sampled at every time instant from the uniform distribution $U(-1, 1)$.

- In the standard *Gaussian plume dispersion* [7] model, the concentration of the plume perpendicular to the direction of the wind velocity follows a Gaussian distribution. A point source is randomly placed inside the network, and a node is considered to be inside the plume if its concentration relative to the source exceeds a threshold of 0.03. The speed of the wind is sampled from a normal distribution with a mean 2, whereas its direction is sampled from a random distribution ranging between -30° and $+30^\circ$ with the x-axis. The rate of dispersion is sampled from a normal distribution with a mean 0.3.

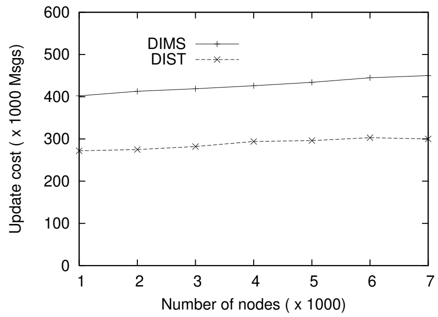

Figure 6: Update cost for the cloud model.

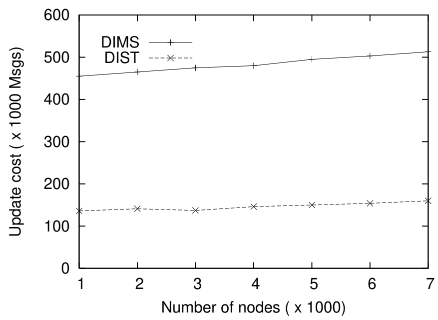

Figure 7: Update cost for the Gaussian model.

For the first model, α is fixed at 5 and is maintained by every node and this (one-time) cost is amortized; for the second, it is allowed to vary during the course of the experiment with a mean around 2 and is not maintained by the network nodes. Query sizes are determined using *exponential* distribution and the location of the query is sampled from a uniform distribution. In experiments where query cost is measured for varying spatial query sizes, the temporal query size is fixed at 10 % *Exp_Time* (the total time of Experiment). For experiments with varying temporal query sizes, the spatial query size is fixed at 10 % *Grid_Size* (the total area of the network).

4.2 Update costs

Figs. 6 and 7 plot the average update costs for the *cloud* and the *Gaussian* models. We see that DIST scales well with the network size. Since Geocast has zero update cost, it is not shown. In Fig. 6, plume localization and the hierarchical structure allows DIST to reduce the number of updates. DIMS update costs are high due to the expensive hash involved in inserting a set of points. Therefore, DIST performs 2 times better than DIMS. In Fig. 7, we see a further reduction in DIST's update cost. This is attributed to the high degree of localization of the plume. DIMS incurs higher costs than before due to the large plume size. Here, DIST performs 3 times better than DIMS.

4.3 Query costs

In this section, we first show how adaptive query overcomes the disadvantages of both the SCA and Direct querying schemes and lowers the average communication cost on a per-query basis. Then, we show that the adaptive query scheme performs two to three times better than DIMS and Geocast.

4.3.1 Adaptive querying

In the Adaptive scheme, the query originator chooses the optimal scheme depending on the query location, query size and plume distribution (Section 3.3) on a *per-query* basis. Every point shown in the following plots is averaged over a set of 100 queries with the same query size but placed at different locations.

Figs. 8 and 10 show the query cost of the Adaptive scheme as compared to the *Direct* and *SCA* algorithms for the cloud model.

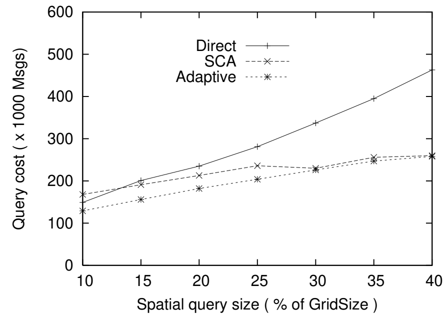

Figure 8: Query costs for varying spatial query size for the cloud model.

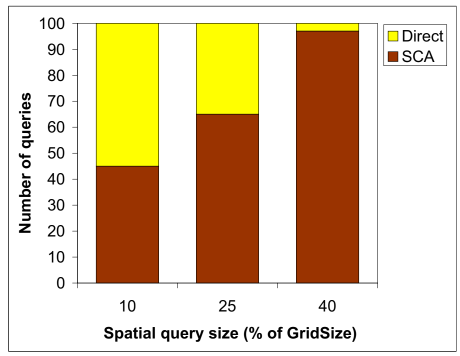

Figure 9: Adaptive query's preference to SCA and Direct for the cloud model for varying spatial query sizes.

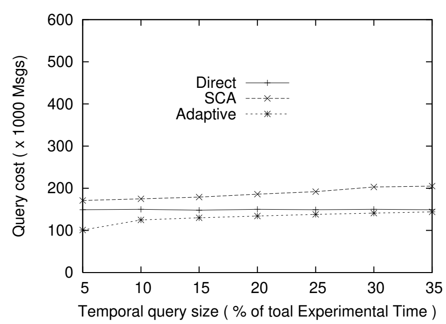

Figure 10: Query costs for varying temporal query size for the cloud model.

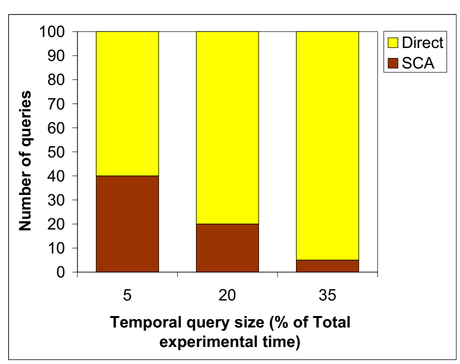

Figure 11: Adaptive query's preference to SCA and Direct for the cloud model for varying temporal query sizes.

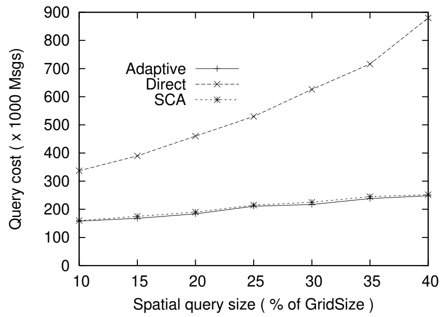

Figure 12: Query cost for varying spatial query size for the Gaussian model.

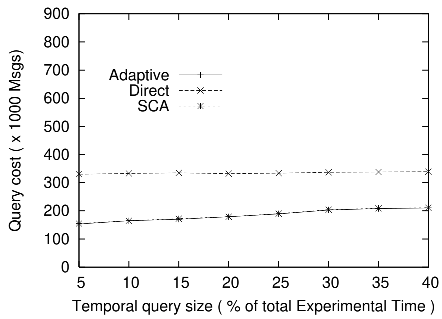

Figure 13: Query cost for varying temporal query size for the Gaussian model.

Fig. 8 illustrates the query costs for varying spatial query sizes. We see that *SCA* performs better than *Direct* because of its spatial pruning in choosing the smallest common ancestor, and spatio-temporal pruning while drilling down. Even though α is fixed, the cost of Direct query grows linearly with the size of query because it is routed to all the cell leaders within the query's spatial range. Because the Adaptive scheme chooses the minimal cost scheme for every query, its average cost (over the 100 queries) is lower than the average costs of *SCA* and *Direct* schemes. As the cost difference between the two algorithms increases, the Adaptive scheme converges to the better algorithm, i.e., *SCA*. For the set of 100 queries, Fig. 9 shows the preference of Adaptive scheme towards the SCA compared to the Direct. Though at small spatial query sizes Direct is preferred, at large query sizes SCA dominates the query set.

Fig. 10 depicts the query costs for varying temporal query sizes. We see that the performance of the Direct algorithm is independent of the temporal query size (as expected). The *SCA* scheme performs well because of its pruning, but *Direct* performs better due to the small spatial query size. Therefore, the Adaptive scheme converges to *Direct* in this case. For the set of 100 queries, Fig. 11 illustrates the preference of Adaptive scheme towards the Direct compared to the SCA. At large temporal query sizes, SCA cannot prune as well temporally, whereas Direct retains its fixed and small cost. Note in both the experiments, Adaptive querying converges to the optimal scheme.

Figs. 12 and 13 plot the average query cost for the Gaussian model. In Fig. 12 we notice the superior performance of the SCA over the Direct querying due to two reasons. First, as α is not known *a priori*, the Direct scheme propagates the query across spanning trees at all levels intersecting the query leading to a huge communication cost. Second, the temporal pruning employed by the SCA decreases the drill-down cost. For the set of 100 queries, we noticed that Adaptive querying employed the SCA for (almost) all the queries. Fig. 13 once again shows the independence of the Direct algorithm with time. Although SCA's pruning performance

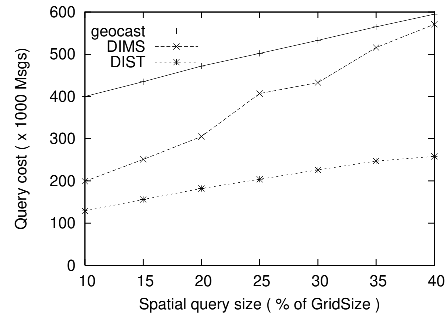

Figure 14: Query cost for varying spatial query size for the cloud model.

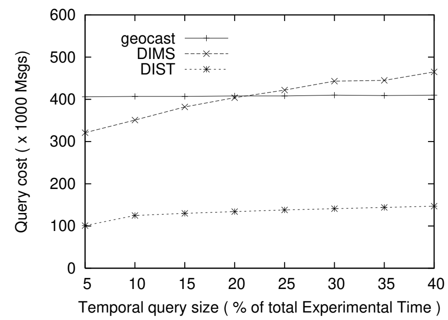

Figure 15: Query costs for varying temporal query size for the cloud model.

decreases at large temporal query sizes, its costs are much lower compared to the Direct scheme.

4.3.2 Computational cost analysis

Profiling [21] on an Atmel 128L micro-controller used in the motes running TinyOS [1] shows that the worst case computational cost of calculating equations 4–9 is around $800,000$ CPU instructions. The cost of transmitting a bit is equivalent to executing $12,000$ operations on a Mica2 mote [1]. This implies that the computational cost of Adaptive query scheme is recovered even if a single byte is saved over a single hop. We measure the *per-query* communication benefits of our adaptive query scheme by taking the difference between the average cost of Adaptive query scheme and the minimum of the average cost of *Direct* and *SCA* schemes over a set of 100 queries. Figs. 8 and 10 show that the communication savings of the adaptive query scheme is in the order of 10000 messages where the message size was 8 bytes. Therefore, the communication benefits are 10^3 times the accrued computational costs.

4.3.3 Comparison with alternatives

Here, we compare the performance of our adaptive query scheme (*DIST*) against DIMS and Geocast. Figs. 14 and 15 plot the average query costs for the cloud model. In Fig. 14, DIST performs better than the other schemes because it employs *SCA* query to prune spatio-temporal query while drilling down the hierarchy. DIMS and Geocast do not scale with the query size. This is because Geocast forwards the query to all the nodes in the query region and DIMS hashes the query to a large region of nodes. In Fig. 15, Geocast and DIST (which now employs *Direct*) are independent of the temporal query size (as expected). DIST performs 3 times better than DIMS on average. In case of DIMS, its hash based on temporal dimension grows linearly with the size of query. Please refer to the full paper [18] for performance gains (up to 2 times) of DIST over DIMS and Geocast for the Gaussian model.

Figs. 16 and 17 plots the average query cost for the Gaussian model. In Figure 16, average level of the plume was around 2. As a

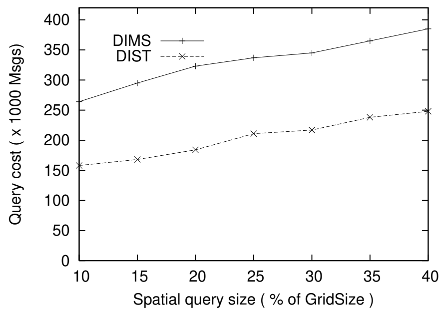

Figure 16: Query cost for varying spatial query size distribution for the Gaussian model.

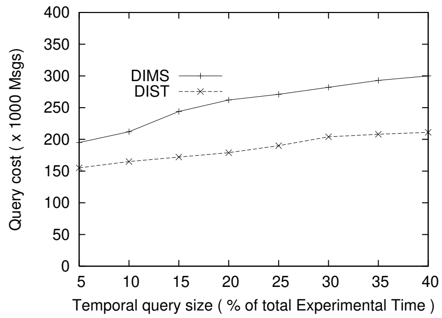

Figure 17: Query cost for varying temporal query size distribution for the Gaussian model.

result, we observed that most of spatial queries were contained in a cell of level 2. As seen in Figure 12, DIST mostly employs the *SCA* algorithm because of its superior spatial pruning performance when α is not known *apriori*, to perform two times better than DIMS. As shown in Figure 17, DIST outperforms DIMS again since DIMS does not scale with the temporal query size. Since the Geocast scheme is independent of the plume size and is only dependent on spatial query size, its performance results were similar to the previous experiment.

4.4 Storage consumption

The average storage distribution on the nodes was computed experimentally on a network of size 8000. Storage unit reflects the storage for index. DIST uses 2 units for its open/closed time intervals and 2 units for (id, t) tuples; whereas DIMS uses 2 units for its index (centroid, time-stamp). Geocast is not shown because it does not maintain any indices. DIST used less storage compared to DIMS (as shown in Table 1) because of two reasons. First, update pruning created few open or closed time intervals at the higher level nodes in the DIST hierarchy. Second, at every time instant the number of events DIMS inserts is proportional to the plume size. Storage consumption for the Gaussian model is shown in Table 2. In this case, the plume is bigger and localized better. DIMS created

Table 1: Storage consumption for the cloud model

DIST		DIMS	
% of nodes	storage units ($\times 10^5$)	% of nodes	storage units ($\times 10^5$)
30	below 10	20	below 75
20	10–30	40	75–110
35	30–40	10	110–140
15	40–60	30	140–180

Table 2: Storage consumption for the Gaussian model

DIST		DIMS	
% of nodes	storage units ($\times 10^5$)	% of nodes	storage units ($\times 10^5$)
45	below 30	75	below 30
30	40–70	25	220–280
25	70–100		

hot spots due to the strictly increasing temporal dimension.

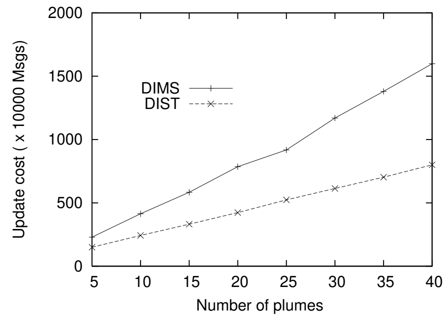

Figure 18: Scalability for the cloud model.

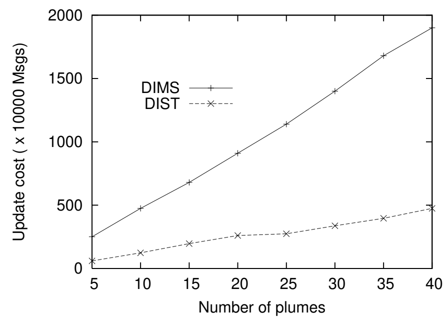

Figure 19: Scalability for the Gaussian model.

4.5 Multiple plumes

Our techniques can be generalized to multiple plumes, where the id of the plume is stored along with each index. The identity question "if a node n senses a plume at time t, and it senses another plume at time $t + 1$, how does it determine whether it is sensing the same plume twice, or two different plumes" can be addressed based on movement patterns [24]. Figs. 18 and 19 show that DIST's update cost scales well with the number of plumes. For the cloud model (Fig. 18), DIST outperforms DIMS due to the pruning of update messages for every plume. Geocast has zero update cost and hence is not shown. Fig. 19 illustrates the scalability for the Gaussian plume model. We see a further improvement in DIST because of a greater localization. Large plume sizes and DIMS's expensive update scheme result in a poor scalability. Although the query costs for both scale with the number of plumes, DIST performs better message pruning due to adaptive querying.

5 Related Work

The problem that the paper addresses—*spatio-temporal range queries*—has been greatly studied in spatio-temporal databases. Motion Objects Spatio-Temporal (MOST) models [24] support queries regarding the past, present and future locations of mobile objects. There have been many spatio-temporal indexing techniques using R-Trees [10] and its variants, by transforming the location-time plane into the dual velocity-intercept plane. TPR-tree [20] which keeps a previous history of each object is update intensive. Sun et al [23] use multi-dimensional histograms to count objects in a partition of 2D-grid. These schemes operate in a centralized, infrastructure network with no memory or power limitations. In [9], a two-tier architecture was proposed employing location preprocessors to index motion-sensitive bounding boxes of objects to efficiently predict and answer continual queries.

Peer-to-peer DHT systems such as Chord [22] and CAN [19] distribute indices across the network to answer match queries. DIST's hierarchy is similar to CAN, but our logical overlay is consistent with the physical topology. Further, adapting a decentralized P2P spatial index (e.g., P-tree [4]) to handle spatiotemporal queries may lead to root bottlenecks as all updates and queries pass through it.

Kung et al [13] use a publish-subscribe system to track point objects in a sensor network. GLS [14] provides location service to mobile users and uses geographic forwarding similar to us. But, none of them build distributed index structures to support spatio-temporal range queries. Madden et al use Semantic Routing Trees (SRT) [17] to support single key range queries. Li et al proposed DIMS [16], a distributed hashing technique to resolve multi-dimensional range queries. DIMS uses the spatial hierarchy for hashing, whereas DIST uses it to exploit plume *localization*. BBQ [5] proposes probabilistic modelling at a central location to optimize sampling costs.

6 Conclusions

We considered the important problem of object tracking, spatio-temporal summarization and querying in sensor networks. For the case of deformable objects such as chemical plumes, we presented an efficient distributed index structure which *localizes* in both space and time and scales with the size of the plume. For this scheme, we presented two different query algorithms. The *Direct* query algorithm performs optimally at small query sizes, whereas the *SCA* query algorithm scales well with the query sizes. So, an analytical model was designed to evaluate the expected query cost for each algorithm based on the query size, query location and the plume distribution. Using the model, the adaptive scheme chooses the optimal algorithm on a per-query basis. Our experimental results showed that DIST performs 2-3 times better than existing techniques in update, querying and storage costs for different network, plume, and query sizes, and scales well with the number of plumes.

7 REFERENCES

[1] *Crossbow Inc. Wireless sensor networks*, http://www.xbow.com/.

[2] V. M. Chernousenko. *Chernobyl, Insight from the Inside*. Springer-Verlag, 1991.

[3] K. K. Chintalapudi and R. Govindan. Localized edge detection in a sensor field. *SNPA*, 2003.

[4] A. Crainiceanu, P. Linga, J. Gehrke, and J. Shanmugasundram. Querying Peer-to-Peer networks using P-Trees. *WebDB*, 2004.

[5] A. Deshpande, C. Guestrin, and S. Madden. Model-driven data acquistion in sensor networks. *VLDB*, 2004.

[6] H. V. Dop, F. T. M. Nieuwstadt, and J. C. R. Hunt. Random walk models for particle displacements in inhomogeneous unsteady turbulent flows. *Physics Fluids*, 1985.

[7] R. R. Draxler. Forty-eight hour atmospheric dispersion forecasts at selected locations in the united states. *NOAA ERL ARL-100*, 1981.

[8] D. Estrin, R. Govindan, and J. Heidemann. Next century challenges: Scalable coordination in sensor networks. *MOBICOM*, 1999.

[9] B. Gedik, K. L. Wu, P. Yu, and L. Liu. Motion adaptive indexing for moving continual queries over moving objects. *CIKM*, 2004.

[10] A. Guttman. R-trees. a dynamic index structue for spatial searching. *SIGMOD*, 1984.

[11] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. *MOBICOM*, 2003.

[12] Y. Ko and N. H. Vaidya. Geocasting in mobile networks: Location-based multicast algorithms. *WMCSA*, 1999.

[13] H. T. Kung and D. Vlah. Efficient location tracking using sensor networks. *WCN*, 2003.

[14] J. Li, J. Jannotti, D. DeCouto, D. Karger, and R. Morris. Geographic ad hoc routing. *MOBICOM*, 2000.

[15] Q. Li, M. DeRosa, and D. Rus. Distributed algorithms for guiding navigation across a sensor. *MOBICOM*, 2003.

[16] X. Li, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. *Sensys*, 2003.

[17] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Design of a acquisitional query processor for sensor networks. *SIGMOD*, 2003.

[18] A. Meka and A. K. Singh. DIST: A Distributed Index Structure for Tracking Plumes in Sensor Networks. *UCSB-TechReport*, 2005.

[19] S. Ratnaswamy, P. Francis, R. Karp, and S. Shenker. A Content-Addressable network. *SIGCOMM*, 2001.

[20] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. *SIGMOD*, 2000.

[21] V. Shynader, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. *Sensys*, 2004.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. *SIGCOMM*, 2001.

[23] J. Sun, D. Papadias, and B. Liu. Querying the past, present, and future in spatio-temporal databases. *ICDE*, 2004.

[24] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving object databases:Issues and solutions. *SSDBM*, 1998.

APPENDIX

A Computing $\phi_s(q)$:

For the gaussian plume model, the distribution $p(x, y, t)$ is given by:

$$\begin{aligned} p(x,y,t) &= \frac{1}{2\sqrt{\pi}\sigma_t} \mathrm{e}^{-\frac{(x-\mu_{x_t})^2+(y-\mu_{y_0})^2}{2\sigma_t^2}} \\ \text{where} \quad \mu_{x_t} &= \mu_{x_0} + a \cdot t \quad \text{and} \\ \sigma_t &= \sigma_0 + b \cdot t \end{aligned} \tag{10}$$

The above equation reflects that the mean translates due to wind convection in the x-direction and the variance changes due to dispersion perpendicular to the wind velocity. For the cloud model, the plume density is given by:

$$p(x,y,t) = \frac{1}{2\sqrt{\pi}\sigma_0} e^{-\frac{(x-\mu_{x_0})^2+(y-\mu_{y_0})^2}{2\sigma_0^2}} \tag{11}$$

Now, we show the calculations of $\phi_s(q)$ for the cloud model. Since the plume probabilities at t are defined given $p(x, y, t)$

$$1 - p(x, y, [t_1, t_2]) = \prod_{\forall t \in [t_1, t_2]} [1 - p(x, y, t)] \tag{12}$$

Taking the natural log on either side of the above equation and noting that this model is a steady state distribution leads to

$$\begin{aligned} \log[1 - p(x, y, [t_1, t_2])] &= \int_{t_1}^{t_2} \log[1 - p(x, y, t)]\, dt \\ &= \log[1 - p(x, y, t)]^{t_2 - t_1} \end{aligned} \tag{13}$$

Substituting Equation 11 on the right hand side of the above equation and using Taylor's first series approximation we get

$$p(x, y, [t_1, t_2]) \approx (t_2 - t_1) \frac{1}{2\sqrt{\pi}\sigma_0} e^{-\frac{(x-\mu_{x_0})^2+(y-\mu_{y_0})^2}{2\sigma_0^2}} \tag{14}$$

Substituting the above term in Equation 8, $P_{i,l}[t_1, t_2]$ becomes

$$\frac{(t_2 - t_1)}{2\sqrt{\pi}\sigma_0} \int_{x_1}^{x_2} \mathrm{e}^{-\frac{(x-\mu_{x_0})^2}{2\sigma_0^2}} \mathrm{d}x \int_{y_1}^{y_2} \mathrm{e}^{-\frac{(y-\mu_{y_0})^2}{2\sigma_0^2}} \mathrm{d}y,$$

which is equal to

$$\frac{\sqrt{\pi}(t_2 - t_1)}{8\sigma_0} \left[\mathrm{erf}\left(\frac{x - \mu_{x_0}}{\sqrt{2}\sigma_0}\right) \right]_{x_1}^{x_2} \left[\mathrm{erf}\left(\frac{y - \mu_{y_0}}{\sqrt{2}\sigma_0}\right) \right]_{y_1}^{y_2} \tag{15}$$

For the query shown in Figure 1, the query originator uses the above equation to calculate $P_{i,l}[t_1, t_2]$ for cells F, G, and H (at level 3 intersected by the query). It then uses Equation 8 to compute probabilities of cells P and Q at level 2, and of cell T, the *sca(q)*. Then, it finally computes $\phi_s(q)$ using equation 9. For the gaussian plume model in Equation 10, the query originator uses numerical integration techniques in order to compute the probabilities in Equation 8, and the rest of $\phi_s(q)$ computation follows as above. Note that every node is initialized with the real world parameters of μ_0 and σ_0 and a and b.