Differential Forms for Target Tracking and Aggregate Queries in Distributed Networks

Rik Sarkar Jie Gao
Computer Science Department, Stony Brook University, Stony Brook, NY, USA.
{rik, jgao}@cs.sunysb.edu

ABSTRACT

Consider mobile targets moving in a plane and their movements being monitored by a network such as a field of sensors. We develop distributed algorithms for in-network tracking and range queries for aggregated data (for example returning the number of targets within any user given region). Our scheme stores the target detection information locally in the network, and answers a query by examining the perimeter of the given range. The cost of updating data about mobile targets is proportional to the target displacement. The key insight is to maintain in the sensor network a function with respect to the target detection data on the graph edges that is a *differential one-form* such that the integral of this one-form along any closed curve C gives the integral within the region bounded by C.

The differential one-form has great flexibility making it appropriate for tracking mobile targets. The basic range query can be used to find a nearby target or any given identifiable target with cost $O(d)$ where d is the distance to the target in question. Dynamic insertion, deletion, coverage holes and mobility of sensor nodes can be handled with only local operations, making the scheme suitable for a highly dynamic network. It is extremely robust and capable of tolerating errors in sensing and target localization. Due to limited space, we only elaborate the advantages of differential forms in tracking of mobile targets. The same routine can be applied for organizing many other types of informations, for example streaming scalar sensor data (such as temperature data field), to support efficient range queries. We demonstrate through analysis and simulations that this scheme compares favorably with existing schemes that use location services for answering aggregated range queries of target detection data.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network protocols—*Routing protocols*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

MobiCom'10, September 20–24, 2010, Chicago, Illinois, USA.

General Terms

Algorithms, Design, Theory

Keywords

Multi-Target Tracking, Range Queries, Differential Forms, Sensor Networks

1. INTRODUCTION

Tracking mobile targets is a common application scenario in modern society. People in motion need to maintain connectivity, thus requiring location management. Other applications, for example monitoring of traffic require real-time assessment of environments of mobile devices. Mobile targets can be identifiable, for example possessing unique identifiers or unique signal signatures, or non-identifiable, for example for privacy concerns. Queries on mobile targets may be about locating the current position of a mobile identifiable target, or aggregated information such as the count of targets in a user specified region. There is often a connected communication infrastructure spanning the space in which targets move. A major technical question is centered around the representation of target motion that will allow the users easy and effective access to the data. The possible solutions can be tailored to different system requirements and assumptions.

Take an example of the location management schemes in cellular systems. The problem is to find the current location of the mobile user when receiving a call. There are two atomic operations, called paging and update respectively. Paging is used when the system searches the cellular towers looking for the user. Update refers to mobile users informing the system of their current locations. The full scheme uses a combination of paging and update, based on user mobility patterns and call frequencies. This solution assumes the cooperation of the mobile users/targets and that the query is for individual identifiable targets.

Targets may not always be so cooperative or capable of direct communication with the system. In such cases the task of locating, tracking and querying for mobile targets is entirely on the communication infrastructure spanning the region. The targets may not be individually identifiable, but being able to detect the number of targets in any region can still supply valuable information. This is motivated by the recent advances of large scale wireless sensor networks. As sensor networks intrude into the space where people live and work, they form a sensing and communication infrastructure that can provide real-time assessment of the living environment and the mobile objects therein. Indeed, tracking of mobile targets is identified as a major motivating application for sensor networks [4, 15, 17, 19, 27, 28, 30] from the very beginning. We use sensor network as a simple model for a distributed tracking infrastructure but the solution is independent of the particular network underneath. For example, wireless enabled devices can be tracked by wireless access points or other wireless devices. In this case, the wireless infrastructure acts as the sensor network.

Consider the following scenario of wide-area deployment of sensors along major roads to track and monitor moving vehicles. A suitable sensor can detect the position and velocity of a target within its sensing range [21], the navigation system in a car may also communicate directly with the sensors. A target may or may not have identifiable signatures. The moving vehicles come in swarms as in the typical case of medium to heavy traffic situation. A user may use hand-held devices (smartphones, PDAs, etc) or the car's GPS system to communicate with nearby sensors or other portals and inquire for the target distribution. Of particular interest to us are *range* queries for *aggregated* data, for example, the level of traffic congestion in a specified neighborhood and its evolution over time. Formally, we ask a counting range query: what is the number of targets in any user-specified region R? The topic for this paper is to develop an efficient data processing and query scheme for such applications. A desirable solution should have low query delay, low communication costs, as well as low maintenance cost as the targets move rapidly.

In sensor networks, the most adopted target tracking approach, arguably, is the sensors to record the detection events in the data logger or report to a base station. The base station assembles target trajectories for post-experiment analysis. This solution bears the common problems of having a central server (bottleneck and a single point of failure, not resilient to attacks), and in particular, the data collection step makes it inappropriate for applications with stringent delay requirements. In many practical scenarios, movements of targets are relevant only in the local region and for a short period of time. For example, some cars turning on a particular by-road is a relevant traffic information only while they are in the neighborhood. It is difficult to justify the high communication and storage costs of updating a remote server for high volumes of such fleeting pieces of data. Very often, users may be in a neighborhood of where the relevant data is generated. A centralized solution would require both the data and query from the users to be delivered to a (possibly) remote server. This leads to unacceptable delay and unnecessary network traffic.

Alternatively, the sensor in the proximity of a target can detect the target and can locally cache the detection event. This scheme has low maintenance cost as data is stored locally and only local updates are needed when target moves. But with such raw detection data stored directly in the network it is not easy to answer range queries. One has to flood all the nodes inside the range R to find out the total number, the communication cost of which is proportional to the area of R, $\mathrm{A}(R)$.

The solution we propose in this paper uses local maintenance, but instead of storing raw detection data, stores target movements implicitly. Counting range queries have costs proportional to the perimeter of range R, $\mathrm{P}(R) \ll \mathrm{A}(R)$. For this we use a novel notion of differential one-form on the network. The key insight is to maintain in the sensor network a function on the edges that is a *co-vector* field with respect to target detection data, which means that the integral along any closed curve C gives the integral of the region bounded by C. Thus our scheme naturally supports efficient range queries by touring along the boundary of the region. This idea is introduced below.

Our approach: differential one-form. A differential form is commonly considered on smooth manifolds, where it is easier to write explicit expressions for smooth forms. In this paper we use a formulation which can be considered an implicit representation corresponding to smooth forms. This representation allows us to consider the concepts in a more discrete manner that is suitable for computations and dynamic modifications. This discrete differential form is defined on a cell complex, for example, a decomposition of the plane into non-overlapping faces by a planar graph. This particular representation of differential forms, while not common in mathematics, its hints can be found in literature [10, 16].

Consider the simplest case. We have a planar graph embedded in the plane, and one target lies within a face f_0 and has a weight of w, e.g., representing its size or other metrics of interest. The differential one-form is represented by a function ξ on directed edges. The value for $\xi(ab)$ must be the negation of the value for $\xi(ba)$. We maintain the property that for the face f_0, the summation of all the values of the edges on its boundary, in clockwise order, is w, and the summation of all the values of the boundary edges on each other face is 0. This ensures that any cycle containing the face f_0 will have a total summation of w, and any cycle not containing f_0 will have a sum of 0. In other words, one is able to answer range queries by simply integrating the differential one-forms along the range boundary. The weight on an edge signifies we have created a differential form whose integral over the edge sums to that value.

The basic definition for one target can be generalized to multiple non-identifiable targets – such that the integral of a face is the total weight of the targets within the face. This way range query can be done for a swarm of targets with the same query cost. Using range queries we can implement the query for locating a nearby target or a given identifiable target. The idea is to use exponentially enlarging range around the query node and once the range includes the target, reduce the range by using divide and conquer. The cost for such is bounded by $O(d)$, where d is the distance to the target in question, representing locality sensitivity.

The differential one-form has great flexibility that allows low maintenance cost under both network dynamics and target movements. When a target moves from one face f_0 to an adjacent face f_1, we only need to update the differential one-form on the edge ab common to f_0, f_1. In particular, $\xi(ab) \leftarrow \xi(ab) - w$, for a target of weight w. This ensures that the property of the one-form is maintained. The cost for the update is a constant and can be done locally. Network dynamics such as link addition and removal, or node insertion and removal, can be handled in constant time. We also show that the differential one-form can be initialized in linear communication cost, i.e., constant cost per node. Further, this aids in energy management. Sensors only need to be active if there are targets nearby. A region of the network where there are no targets need not perform any communications to maintain tracking data, and can sleep or go to low power mode for extended periods.

The method is built on a planar decomposition of the sensing region. The planar decomposition can come from a planar subgraph of the communication graph, or just as a virtual decomposition of the domain, as long as the sensors maintain the counts on these virtual edges based on their sensing data. The method automatically handles sensing holes — relatively large faces in the planar graph. If a target moves deep inside a hole and is not detected by any sensor, its contribution to the total count of a region enclosing the hole is still correct. This is in contrast with the naive approach of storing the target detection data locally where the range query by summing up all sensor detections is incorrect and misses all the targets that are 'lost' in the hole.

Although we present as the major application of differential forms the tracking of targets in swarm, the same routine can be applied for organizing streaming scalar sensor data (such as temperature data field), to support efficient range queries.

The rest of the paper is organized as follows. We review prior related work on range queries of mobile target and elaborate how our scheme fits and compares with the state of the art schemes. Then we introduce the definition of differential one-form on the network. The algorithms for computing and maintaining the one-form are described afterwards. We report simulation evaluations and comparisons with prior work at the end.

2. RELATED WORK

There are a lot of previous works on tracking mobile targets and on range queries of sensor data. We briefly review these work and compare with our approach.

Range queries. For a typical range query, we are given a query region plus possibly a range of the sensor data, and then ask for all the the sensors in the query region whether any sensor data is within the data range. This is a problem that has been studied a lot in computational geometry. Centralized data structures for geometric range query on static points [3] or motion data [2], have been developed. But they are obviously not a good fit for a distributed sensor network setting. Various distributed schemes have been proposed. In the case of a scalar field, one solution is to partition the information about large geographic regions into subsets according to smaller ranges of the field value, and store these subsets in different nodes. This is the approach taken in the DIFS system [14]. In the DIM system [23] a locality preserving hash function is used to map portions of a multidimensional attribute space to sensors so that all data needed to answer a range searching query can be located conveniently. In the fractional cascading approach [12], information is stored so that more detailed information is available about data obtained in the spatio-temporal locality of the sensor where the query is injected—but without sacrificing the ability to query distant regions or times as well.

All of these schemes are designed to support range queries for static sensor data and essentially use a quadtree-type hierarchical space decomposition. For mobile data, constant updates to a fixed space partitioning make these schemes too costly — small movement of a target may lead to updates up to a high level of the quadtree and possibly updates on *all* sensors, if the mobile target happens to cross a high-level boundary. Location services, as described below, alleviate some of the shortcomings of quadtree based schemes and are more appropriate for mobile data. For this reason we only compare with location services in the simulation section. In addition, these range query schemes are mainly for rectangular ranges only. Ranges of other shape must be first partitioned into smaller rectangular ranges, which are queried separately.

Location services. Existing solutions for tracking and searching for mobile targets, termed as *location services*, focus on the tracking and searching of a single target. The earliest work is by Awerbuch and Peleg [5] and followed up in [1, 9, 22] to fine tune the system. The location of a mobile target is updated to a carefully selected set of nodes, called the location servers, whose spatial density cascades exponentially as we move away from the target. This allows 'locality-sensitive' queries, i.e., the cost of a query is proportional to the distance to the target. When a target moves, information is updated on a location server, with the frequency inversely proportional to the distance to the target. The information of a nearby location server is more up-to-date. Forwarding pointers are left at the old position pointing to the current position of the target. A query far away from the target may first obtain outdated information pointing to a past location, from where the query can be delivered to the current position by following the forwarding pointers. This family of schemes focused on the tracking and searching of an individual, identifiable target. Location services have amortized update cost of $O(d \log d)$ when a target moves a distance d, and a query cost of $O(d')$ if the query node is of distance d' away from the target's current location. In comparison, we have better asymptotic bounds. Our update cost is worst case $O(d)$ and query cost is no more than $O(d')$. In addition, location services do not support range queries very well. If there are multiple targets, they are handled separately. For range queries or aggregated queries (such as density) one has to search for location servers for all potential targets within the range, which can be highly inefficient. Note also that this method requires tracking data to be sent and stored at far away nodes. Thus, even if targets are concentrated only one region of the network, other nodes have to stay awake for storage and communication of the tracking data. In this paper we evaluate the performance of using location services and using our method for range queries in the simulation section. We show that for both update and query cost, our method is substantially better.

Information gradients. The third approach is to define a potential field centered at the target. Such information potential fields can be either the natural gradients of physical phenomena, since the spatial distribution of many physical quantities, e.g., temperature measurements for heat, follows a natural diffusion law [6–8, 25], or built explicitly on a mobile target. One scheme in this family uses harmonic function to build such information strength field [24], which satisfies the Laplace's equation $\nabla^2\Phi(x) = 0$ with proper Dirichlet boundary condition (1 at the target location and 0 at the network boundary). Such an information field is guaranteed to be free from local minima. Thus every node can follow the local *information gradient* to arrive at the target. This works for both identifiable (information fields are maintained separately) and non-identifiable targets (a single information field is maintained for all targets). In addition, the divergence-free property of harmonic gradients and Faraday's law of induction imply an easy solution for counting range queries — touring the boundary of a given range and summing up the difference of the potential values on the edges across the region boundary provide the number of targets in the interior of the range. When a target moves, the information field needs to be updated to ensure the harmonic function property. The limitation of the scheme is that updating the potential field for mobile target is costly by the global nature – nodes far away from the target have to update their information strength, while ideally we hope to restrict the updates to be within a small neighborhood of the target. If we 'rotate' the gradient vectors by 90°, the result is a differential harmonic one-form. In our scheme we do not require the differential one-form to be harmonic – thus one can not as easily navigate towards the target as in the scheme in [24]. However, the benefit of using a relaxation as simply a differential one-form is to allow quick maintenance of the one-form under target motion. As we have shown, the update is completely restricted to the target neighborhood.

To summarize, the scheme proposed in this paper complements the state of the art data processing methods in a sensor network by providing low-maintenance, low cost range query scheme for a large number of non-identifiable mobile targets.

3. DIFFERENTIAL ONE-FORM ON CELL COMPLEXES

The differential one-form is defined on a *cell complex*, induced by a planar graph G in the plane in our case. The vertices, edges and faces of the planar graph are the $0, 1$ and 2 dimensional elements created by the planar graph. In algebraic topology these as called the 0-cells, 1-cells and 2-cells respectively. See Figure 1 for examples. The composition of the different dimensional cells covering the deployment region is called a *cell complex*. The idea of a cell complex extends up to k-cells for arbitrary k. A more detailed treatment of cell complexes can be found in [16].

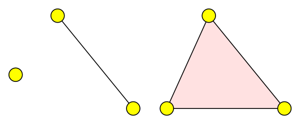


Figure 1. 0, 1, 2-cells.

Our focus is to track targets in the plane as they move between faces (2-cells) of the planar graph – which is a 2-complex in the plane. We assign and update weights of the edges (1-cells) of the complex. The idea however extends to suitable complexes of higher dimensions.

For ease of explanation, we assume for now that the targets are accurately tracked by nearby sensors. Various target detection schemes and signal processing primitives have been developed in the literature [21]. In the algorithm and simulation sections we address the issues of sensing holes and target detection errors. Our strategy assigns values to edges of the planar graph, and changes these values as the target moves. We introduce the following definitions and notations to represent the related faces, edges and values.

3.1 Boundaries and Boundary Chains

A face is demarcated by the edges or 1-cells that surround it. Such a set of edges form the *boundary* of the cell. For an edge pq, we use the ordered pair (p, q) to represent a directed edge whose direction or orientation is from p to q. We use $-(p, q)$ to represent the same edge with orientation (q, p). For brevity, we can represent (p, q) and (q, p) as e and $-e$ respectively. In a diagram, when an edge is labeled simply as e, an arrowhead is used to represent the intended orientation. The opposite orientation will naturally correspond to $-e$.

Definition 3.1. Edge chain or 1-chain. *Suppose $a, b, c \dots$ are oriented edges or 1-cells, then a chain on these edges is a formal sum $\lambda_1 a + \lambda_2 b + \lambda_3 c + \dots$, where each λ_i is an integer.*

This chain simply signifies λ_1 occurrences of a, λ_2 occurrences of b etc. The advantage of the summation notation will be clear in a short while. Note that in many cases we consider, the edges will be adjacent to each other and form a connected path. But this is not necessary in general, and the edges in an edge chain can in fact be any set of edges from the complex.

We can also associate orientations with 2-cells or faces. These correspond to traversing the boundary cycle of a face in some direction, clockwise or counter-clockwise. In this paper we assume that all faces are oriented in the clockwise direction. Such a consistent orientation of cells is made possible by the fact that the 2-dimensional plane is *orientable* [20]. Thus, given a cell σ represented as an ordered tuple $\sigma = (p, q, r, s, t)$, as shown in Figure 2, we understand that the order corresponds to a clockwise traversal of edges (p, q), (q, r), (r, s), (s, t) and (t, p). Correspondingly, $-\sigma$ is the same cell with the opposite orientation, $-\sigma = (t, s, r, q, p)$. Observe that the orientation of a cell implies a specific orientation for each edge on its boundary.

Definition 3.2. Boundary operator ∂. *The boundary operator ∂ acts on a 2-cell or a face σ to produce a chain $\partial(\sigma) = a + b + c \dots$ where $a, b, c \dots$ are the edges on the boundary of σ, with orientations inherited from the clockwise orientation of σ. For a set of faces $U = \{\sigma, \tau \ \dots\}$, we extend ∂ to operate on it as $\partial U = \sum_{\sigma \in U} \partial\sigma$.*

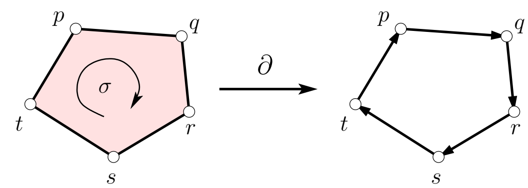


Figure 2. Action of boundary operator on a face σ will give a chain of its boundary edges with orientations inherited from the orientation σ.

The idea behind this definition is shown in Figure 3. The two neighboring faces σ and τ have boundaries $\partial\sigma = a + b + c$ and $\partial\tau = d + e + (-c)$, respectively. Note that a shared edge like c must always appear with opposite orientation, and therefore have opposite signs for the two faces. Thus the resultant boundary $\partial\{\sigma, \tau\} = a + b + d + e$ is exactly the boundary of the union of two faces. This applies more generally to any set of faces. We refer the reader to [20] for more details on the algebra of chains.

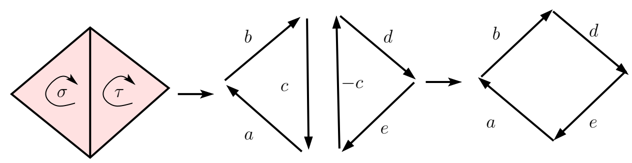


Figure 3. Action of the boundary operator ∂ on faces σ and τ produces the boundary of the union of the two.

3.2 One-Forms and Tracking Forms

In this subsection we define functions over edge chains and show how they help in tracking a target.

We consider a function f that assigns a value to each directed edge in the planar graph P. The function is defined to have the property that $f(-e) = -f(e)$. We extend this function to edge chains by making it distributive over summation: $f(a + b + c + \dots) = f(a) + f(b) + f(c) + \dots$. Let us refer to such functions as 1-*forms* or *edge forms*. A 1-form f can be extended to a 2-form df on the faces of the planar graph, if we let it take the value on the boundary of that face, that is, $df(\sigma) = f(\partial\sigma)$.

Now suppose there is a single target T of weight w in the domain. Then at any given time this target resides in single unique face of the planar graph P [1]. Then we define a one-form on the faces and edges such that it is non-zero on this face and is zero on every other face:

Definition 3.3. Tracking form ξ. *A tracking form ξ for a target T of weight w is a one-form such that*

$$d\xi(\sigma) = \begin{cases} w & \text{if } \sigma \text{ contains } T \\ 0 & \text{otherwise} \end{cases}$$

Remember that on the face σ the form is defined to take a value equal to its sum on the boundary edges, $d\xi(\sigma) = \xi(\partial\sigma)$. We can extend the form to a set U of faces by simple summation : $d\xi(U) = \sum_{\sigma \in U} d\xi(\sigma)$.

[1] The degenerate cases of the target being on an edge or a vertex can be resolved locally by a predetermined policy between the local nodes to assign the target to a face. Therefore, we ignore these cases to keep our discussion simple.

As a direct consequence of this definition, we know that to evaluate the presence of the target within a subset U of faces, it suffices to add the extended tracking-form $d\xi$ on the faces in U. If a face in U contains the target T, then $d\xi(U)$ sums to w, else it sums to zero. The following lemma implies that it is sufficient to sum the form ξ only on the edges that form the boundary of the set U to obtain $d\xi(U)$.

Lemma 3.4. *The sum of the form on the faces in a set U equals its sum applied only to the boundary of U, that is: $d\xi(U) = \xi(\partial U)$.*

PROOF. This follows directly from the definitions that

$$\begin{aligned} d\xi(U) &= \sum_{\sigma\in U} d\xi(\sigma) && \text{from definition 3.3} \\ &= \sum_{\sigma\in U} \xi(\partial\sigma) && \text{from defnition of } d\xi \\ &= \xi\left(\sum_{\sigma\in U}(\partial\sigma)\right) && \text{by distributivity of } \xi \text{ over } + \\ &= \xi(\partial U) && \text{by definition 3.2} \end{aligned}$$

□

This lemma is equivalent to Stokes' theorem [10]. Its significance becomes clear in Figure 4. Given any cycle L in P, it is possible to detect if the target T is inside the loop or not, by simply adding the tracking form along L. If T is in the interior, then $\xi(L) = w$, and if T is not in the interior, then $\xi(L) = 0$. In either

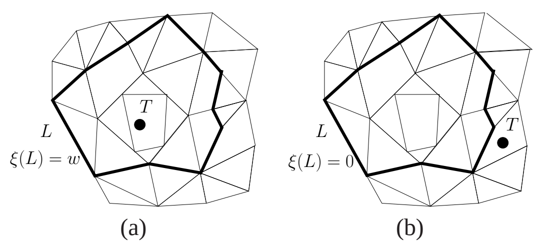


Figure 4. Query for a target T inside L. (a) T is inside L, therefore $\xi(L) = w$. (b) T is not inside L, therefore $\xi(L) = 0$.

case, the query does not need to visit the nodes in the interior of L. A simple walk on the loop suffices to find the answer. Further, this works exactly the same way for any arbitrary loop L and position of the target T.

Multiple Targets. This idea extends to any number of targets in the domain. Suppose targets $T_1, T_2, \ldots, T_k$ of weights $w_1, w_2, \ldots, w_k$, individually give rise to tracking forms ξ_1, $\xi_2, \ldots, \xi_k$. Then we can construct a combined tracking form as the sum of these $\xi = \xi_1 + \xi_2 + \cdots + \xi_k$ on each edge. Given any loop L, the sum $\xi(L)$ will provide the total weight of targets inside L.

The weights assigned to targets can be adjusted to suit the needs of the system. For example, if all weights are equal, then $\xi(L)$ provides the count of targets inside. If each individual target T_i is given weight 2^i, then from $\xi(L)$ it is possible to deduce exactly which ones are located inside L. This is equivalent to maintaining a form for each individual target. It is possible to imagine other scenarios where targets are assigned different weights according to their importance, for example, objects can be classified according to needs and weights assigned according to their types.

Given the weights and target locations, it is always possible to create a suitable tracking form. In the next section we will describe an efficient algorithm.

Updating tracking forms for mobile targets. When a target moves from one face to another, we need to update the tracking form by changing its value on the directed edges. Without loss of generality,

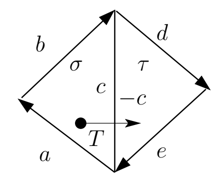


Figure 5. Target T of weight w moves from face σ to face τ. Modify $\xi(c) \leftarrow \xi(c) - w$ to obtain the new form.

we consider the example in Figure 5, where T moves from face σ to an adjacent face τ. Let us say, the shared edge that was crossed by T appears as c in $\partial\sigma$, and as $-c$ in $\partial\tau$. In the initial configuration, we had $d\xi(\sigma) = w$ and $d\xi(\tau) = 0$. After the move, we need to have a final configuration with $d\xi(\sigma) = 0$, and $d\xi(\tau) = w$. This is achieved by the following simple modification to the form on the shared edge:

$$\xi(c) := \xi(c) - w. \tag{1}$$

The same assignment can alternately be written from the point of view of τ as:

$$\xi(-c) := \xi(-c) + w. \tag{2}$$

Evidently, these two are the same operation, since $\xi(-c) = -\xi(c)$.

The following theorem says that this indeed is the correct operation that achieves the desired result.

Theorem 3.5. *If σ and τ are adjacent faces with shared edge c, and $d\xi$ has values $d\xi(\sigma) = u$ and $d\xi(\tau) = v$, then the modification described in equation (1) or (2) results in $d\xi(\sigma) = u - w$ and $d\xi(\tau) = v + w$.*

PROOF. Suppose, the boundary of σ is $\partial\sigma = e_1 + e_2 + \cdots + c + \cdots$. In the initial configuration we had $d\xi(\sigma) = \xi(e_1) + \xi(e_2) + \cdots + \xi(c) + \cdots = u$. After the modification, we have $d\xi(\sigma) = \xi(e_1) + \xi(e_2) + \cdots + (\xi(c) - w) + \cdots = u - w$.

Similarly, after the modification, we have $d\xi(\tau) = \xi(e_k) + \xi(e_{k+1}) + \cdots + (\xi(-c) + w) + \cdots = v + w$. □

In the proof above we take the initial values to be u and v instead of w and zero so that the same proof applies to scenarios with multiple targets, and any preexisting weights on the faces and edges. For a system with a single target, the final values are $\xi(\sigma) = 0$ and $\xi(\tau) = w$, as required. In general, the weight of T is removed from the weight of σ and added to the weight of τ.

4. ALGORITHMS

In this section, we describe the algorithms for constructing the tracking form, and for supporting range queries and other queries.

4.1 Planar graph for tracking

As a first step we compute a planar graph. The planar graph can be either a subgraph of the communication graph of the sensors, or a virtual graph chosen for the tracking application.

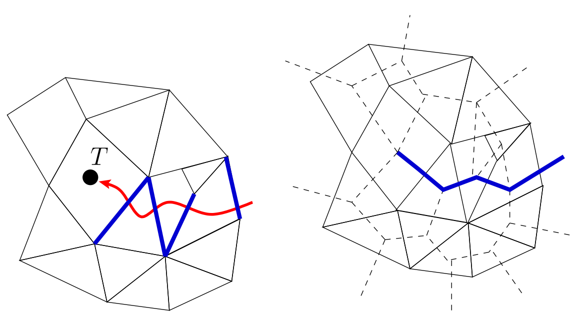
In the first case, consider the sensor network as the nodes embedded in a region in the plane, and an associated communication graph G. We obtain a planar subgraph $P \subseteq G$ that contains all the nodes, but is drawn in the plane without crossing edges. We can apply planarization techniques to extract a planar graph from the network connectivity graph. Such methods have been developed in the past [11, 13, 26, 29]. Any such algorithm can be used for our purpose.

Alternatively, we can also consider a virtual planar graph chosen for the tracking application. For example, the virtual planar graph can represent any convenient space decompositions, such as streets and blocks, any other meaningful districts, or simply a global grid overlayed on the region. For each virtual edge we can appoint a nearby sensor or all the nearby sensors (e.g., those whose sensing ranges cover part of the edge) to 'maintain' the value on the edge. In this case we only assume that a target crossing an edge of the virtual graph can be detected by at least one sensor and the new differential form value is updated. Such virtual planar graphs can be made to create finer subdivisions as required. When the mobile entities can detect their own locations, they can on their own notify the system when they cross an edge of the graph.

4.2 Constructing one-form

In this subsection, we show how to initialize a tracking one-form in the network. First, we describe the simple case where the network is empty of targets to start with, and all targets enter through the outer boundary. Next we will see that the ideas from this case provide a mechanism for initializing the more general case where targets may be present at the time of initialization.

Starting with an empty field. In this case, we initialize all edges to zero, that is for every edge $e \in P, \xi(e) = 0$. Now, suppose that a target T of weight w enters the network. It crosses the edge $c \in \partial\tau$ to enter the face τ. Then we modify $\xi(c) := \xi(c) + w$. Clearly, after this modification, $d\xi(\tau) = w$. As T moves, we can adaptively modify the form according to equation (1) or (2).



(a) (b)

Figure 6. The entry of a target T into the network. (a) As it moves from face to face, it leaves a trail of edges that it modified - shown in bold blue. (b) The trail in the dual graph. The edges of the dual graph are shown as dotted lines, and the dual trail of the target as a solid blue path.

The process is shown in Figure 6(a). As the target moves from face to face, it modifies ξ on the shared edges between adjacent faces. Creating a trail of edges with non-zero values.

Now, let us look a complex $\bar{P}$ that is the dual complex of P. A vertex (say $\bar{\sigma}$) in $\bar{P}$ corresponds to a face (σ) in P. An edge $\bar{e}$ between vertices in $\bar{P}$ represents the shared edge e between corresponding faces of P. The trail of edges in P thus results in a dual trail, which is a path in $\bar{P}$, shown in Figure 6(b). For a more complete picture, we can regard the region outside of the planar graph as a *face at infinity*, and then the dual trail of T is a path from this face to the current position of T.

Initializing a field with targets. The idea of the dual trail directly leads to a simple algorithm to initialize targets in the field. We take a dual path to the face at infinity and add the suitable weight to edges of P whose dual are on the path.

More formally, for a target T, we select any simple directed path α in $\bar{P}$ from the current face of T to the face at infinity. If $\bar{e} = (\bar{\sigma}, \bar{\tau})$ is on α, and $e \in \partial\sigma$, then we do the following modification:

$$\xi(e) := \xi(e) + w, \tag{3}$$

where w is the weight of T. Quite clearly, any simple directed clockwise loop that contains T passes through one such edge. In cases where the loop has more than one such edges, the additional edges appear in oppositely oriented pairs and the values on them cancel out each other.

The following theorem shows that the algorithm above creates a correct tracking form.

Theorem 4.1. *Suppose $d\xi(\sigma) = u$, then after the algorithm above is executed,*

1. *If a face σ contains target T, then $d\xi(\sigma) = u + w$,*
2. *Else $d\xi(\sigma) = u$.*

PROOF. Suppose $T \in \sigma$, then $\bar{\sigma} \in \alpha$ and has an outgoing edge $\bar{e}$. Therefore, after the algorithm is executed, ξ changes on $e \in \partial\sigma$ by $\xi(e) := \xi(e) + w$. All other edges on $\partial\sigma$ remain unchanged. Therefore, after the modification, $\xi(\sigma) = u + w$. This proves the first claim.

Suppose $T \notin \sigma$, if $\bar{\sigma}$ is not on the trail α, then of course nothing changes, and $d\xi(\sigma) = u$. So, the only case we need to consider is when $\bar{\sigma}$ is on the path α. We know that α is a path from the current face of T to the face at infinity, and σ is neither of these. Therefore, $\bar{\sigma}$ has degree exactly 2 in α. Suppose the incoming and outgoing edges are $\bar{e}_1$ and $\bar{e}_2$ respectively. Then the algorithm will have made the following modifications : $\xi(-e_1) = \xi(-e_1) + w$ and $\xi(e_2) = \xi(e_2) + w$. Therefore, the original sum $d\xi(\sigma) = a + \cdots + \xi(e_1) + \xi(e_2) + \cdots = u$ remains unchanged : $d\xi(\sigma) = a + \cdots + (\xi(e_1) - w) + (\xi(e_2) + w) + \cdots = u$. This proves the second claim. □

Once again, the proof works for domains with multiple targets. We execute this once for each target in the domain or for each face containing targets with the total weight of these targets. Thus producing the correct form for initialization. The same procedure can be executed in case a target appears in the middle of the network at any time during the operation.

In cases where there are many targets in the field, creating a trail to the boundary for each can be expensive. In such cases, we perform the initialization as a sweep on the network. We discuss this further in section 4.8.

4.3 Containment queries

Given a one-form on the planar graph, we can query the number of targets inside any loop on the planar graph. This subsection extends it to queries of a geometric range. In the following we use the example of user specified squares. Other geometric ranges can be handled in a similar manner.

For now, let us assume that the network is sufficiently dense so that every point within it is covered (sensed) by one or more sensors, in particular that every point in a face is within a small constant distance δ of some vertex of the face. Let us also assume that the density is bounded, that is, inside any disk of radius 1 the number of nodes is bounded by some constant k. This is not a very restrictive assumption. In a very dense network, we can select a sample of bounded density that still covers the region. We assume geographic face routing [18] is used to follow the faces that intersect a given geometric curve.

Let us use the notation $S_p(r)$ to denote the square of side length $2r$, centered at point p. We sometimes use p to denote both a node and its location. We define the *size of* $S_p(r)$ to be r. The goal is to compute the weight of targets inside this box, or equivalently, compute the sum of the tracking form on the boundary $\partial[S_p(r)]$.

Consider the faces of P that intersect this boundary. By the assumptions above, there are at most a constant number of these within a unit distance of any point on $\partial S_p(r)$. Therefore, the number of faces intersected by the boundary is $O(|\partial S_p(r)|)$ or $O(r)$.

Let Q represent this set of faces at the boundary. For a sufficiently large box queried, Q is an annulus and ∂Q has 2 different connected components — say $\partial Q = \beta + \gamma$ where each is a connected edge chain, in fact a cycle. One of these, say γ lies outside $S_p(r)$ and β lies inside. We say that γ and $-\beta$ respectively form the outer and inner approximations of $\partial S_p(r)$. The reason for taking $-\beta$ is that β by default is oriented counter clockwise, therefore we reverse the orientation to match our conventions. $\xi(-\beta)$ gives a lower bound on the weight of targets inside the box, while $\xi(\gamma)$ gives an upper bound.

We can now find the answer to our query. First, we find $\xi(-\beta)$. Next, for every face $\sigma \in Q$, we manually check the total weight of targets inside $\sigma \cap S_p(r)$. The sum of these values with $\xi(-\beta)$ gives the answer.

Note that this entire computation can be done in a distributed manner by a single walk along the cycle $\partial S_p(r)$. The size of the sub-complex induced by Q and therefore the cost of this computation is $O(r)$.

4.4 Search queries

In this section, we build an algorithm to answer queries of the type "Find the target T starting from p." It is assumed that a differential form is maintained for the identifiable target T, that can be used to search for T, Similar ideas apply to find a target nearest to p.

We search in two stages. First, we find the smallest box $S_p(2^i)$ that contains T. This is done by successively checking $S_p(2^i)$ for $i = 0, 1, 2, 3, \ldots$. Suppose the T is at a distance d, then the size of the largest box tested in this process is $2^{\lceil \lg(d) \rceil}$. Denote this box as $B_p(r)$. From section 4.3, the cost of checking a box of size r is bounded by ar for some constant a. Then the total cost of the test above is $a \sum_{i=0}^{\lceil \lg(d) \rceil} 2^i = O(d)$.

In the second step, we search within the box $B_p(r)$ recursively for the actual location of the target. We partition the box $B_p(r)$ into four quads, each of size $r/2$, and check each of these for the presence of a target. Each test costs $ar/2$, therefore, the total test for 4 quads costs $2ar$. This is done recursively until we arrive at a node that 'sees' the target. Clearly, the cost of this recursive search is $4ar(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots) = O(r)$. Since r is at most $2^{\lceil \lg(d) \rceil}$, we have that the total cost of finding the nearest target is $O(d)$, that is of the order of the distance to the target.

Our query cost is sensitive to the distance to the target. Notice that whether we simply want to deliver a message to the target or obtain its location, the cost is $\Omega(d)$. Thus our query cost is asymptotically optimal.

4.5 Update costs

The network incurs a certain cost in updating the tracking form as a target moves. To be precise, every time the target moves from one face of P to another, the form on that edge has to be updated. Therefore, the total cost of the update equals the number of faces traveled by the target. By the arguments in section 4.3 as a target moves along a straight line segment of length d, the system requires $O(d)$ updates at nodes. If updating an edge requires communication between the endpoints, then the communication cost is also $O(d)$. Note that in some cases this may not be necessary. If both the sensors can detect a target entering a face, which can happen for example if the sensing range covers the entire edge, then the target is sensed by both these sensors, and each can update their view of the edge without any mutual communication. In such cases, the update is carried out without any communication at all.

One can consider adversarial behavior, for example where a target repeatedly crosses an edge back and forth to induce many updates in the nearby sensors. However, this sort of behavior is easy to detect, and can be handled separately. If we would like to reduce maintenance cost, we can stop updating that edge for some time. That is, the edge is assumed not to exist in P for that duration. Note that this 'hole' in the graph does not affect anything in the rest of the network at all. Updates and queries can proceed as usual and the query result is not affected unless the query happens to use this edge. The edge can be reinstated when target movement is infrequent.

In general, when a part of the network is very active with many and frequent movements, it may not be economical to track all such changes. Our scheme is sufficiently flexible and robust that tracking can be turned off in such regions without any loss to other parts or any overhead. Alternatively, it is possible to reduce the tracking resolution in that region by selectively removing nodes and edges so that the faces are larger and therefore incur fewer updates.

4.6 Network holes, fault tolerance and network dynamics

If a network has coverage holes, a target entering the hole might be lost – no sensor detects its location. However, our range query result is not affected if the query range is either outside the hole or encloses the hole completely. If the query range happens to cut through the hole, this is a pathological case that no method can accurately tell whether the target is inside or outside the range, due to limited sensing coverage. We can however get upper and lower bounds (such as $\xi(\gamma)$ and $\xi(-\beta)$ in section 4.3) by computing the weights inside such uncovered faces. When initializing a network with large holes, these are simply disregarded, that is, the corresponding vertex does not exist in the dual. The dual trail for the initialization therefore never goes through the hole.

The scheme is also fault tolerant and adaptive to network dynamics. If some nodes fail, or all nodes in a region fail even including those near the target, that does not affect the correctness of the tracking form. Thus, this permits dynamic networks where nodes can be turned off arbitrarily. There is no overhead on maintaining the tracking form on surviving sensors. Nodes can also be inserted into the network. This only requires refining the planar graph and the tracking form locally. See Figure 7 for an example.

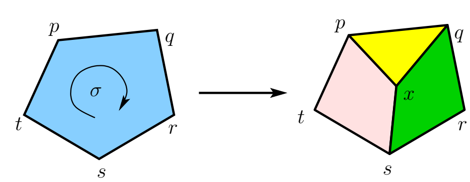


Figure 7. Suppose a node x is inserted inside a face $\{p, q, r, s, t\}$ of total weight w and the face is partitioned into three faces $\{p, q, x\}$, $\{q, r, s, x\}$, $\{p, x, s, t\}$, where the total weights within these faces are w_1, w_2, w_3 respectively, $w_1 + w_2 + w_3 = w$. We simply set the values of the edges $\xi(x, p) = 0$, $\xi(x, q) = \xi(p, q) - w_1$, $\xi(x, s) = \xi(p, q) + \xi(q, r) + \xi(r, s) - w_1 - w_2$. One can verify easily that these values conform to the definition of a tracking form.

The effect of sensing noise is local. Suppose an edge gets updated incorrectly due to sensing or communication failure. This only affects the evaluation of loops that actually pass through that edge. All other loops still produce the correct results. In our simulation sections we evaluate the tracking results when sensing is inaccurate.

4.7 Tracking without target locations

Up to this point, we have assumed that the location of the target can be sensed by the nearby sensors. We now show how to modify the tracking scheme so that it can work without target localization.

Start from the simple case when the target T is detected by exactly one sensor at a time. We initialize this scenario as follows. Suppose s is the sensor detecting T. Remove s (and all incident edges) from P to get a new planar graph P'. Then in P', T is assumed to reside in the new face with the neighbors of s on the boundary. Now, we can initialize the form as usual on the dual of P'. When the target moves from s to a neighboring node t, we first remove t from P' and then reinstate s and its edges using the method for inserting vertices.

The method naturally extends to cases where a target is detected by a set of sensors. In this case, we just remove all the detecting nodes, and when the target moves, we reinstate those that no longer detect it.

4.8 Aggregation of signal over all nodes

Beyond tracking moving targets, differential forms can also be used to compute aggregates of arbitrary functions sampled by sensor network. Suppose h is such a function. Since we have a method for computing sums of values defined over faces of P, we adapt to make use of that existing method. For any node s, we apply small perturbation to the location. That is, the value $h(s)$ is assumed to exist as an added weight in a face σ incident on s, that is $d\xi(\sigma) \leftarrow d\xi(\sigma) + h(s)$. Each node remembers to which face its value was delegated.

First, we have to initialize the form over all faces. For every face σ, we have to find a path α to the face at infinity, in the dual graph $\bar{P}$. To build these paths, we construct an aggregation tree T in $\bar{P}$, rooted at the vertex for the face at infinity. The path for sigma is then the path in T between $\bar{\sigma}$ and the face at infinity.

Next, starting at the leaves of $\mathcal{T}$, we compute an aggregate at each interior node by summing its value with those of its children in the the aggregation tree. Let us denote this function on the dual nodes as μ. For every node $\bar{\sigma} \in \mathcal{T}$, consider the edge $\bar{e}$ to its parent in the aggregation tree $\mathcal{T}$ and its dual e in the original graph P. We set $\xi(e) = \mu(\bar{\sigma})$. This initialization can be executed as a single aggregation sweep on the tree $\mathcal{T}$. Therefore, it can be computed at a total communication cost of $O(n)$.

Now we reconsider the way the function h is handled. We had perturbed h and shifted the value $h(s)$ to a neighboring face σ. This perturbation can cause query results to be erroneous. However, this is easily rectified. Suppose L is the loop that bounds the closed area over which we wish to compute the aggregate. Observe that for a loop not passing through s, the contribution of $h(s)$ is estimated correctly – since then both s and σ are either both inside or both outside the loop. We only need to adjust carefully for loops passing through s. In this case, we need to see whether σ is inside or outside the query region. If σ is inside the region then $h(s)$ is already incorporated in $\xi(L)$. If σ is outside, then the value of $h(s)$ is manually added to $\xi(L)$.

If L is traversed clockwise, then faces on the right of the path are inside, else the faces on the left are inside. Therefore the challenge is to find the orientation along which L is traveled. This we do by means of another differential form, calculated on the fly. Let us say e is the first edge traveled along L, and say σ_1 and σ_2 are the faces adjoining e. Now, we choose arbitrary points $p_1 \in \sigma_1$ and $p_2 \in \sigma_2$ respectively. As we walk along L, we maintain two other one-forms η_1 and η_2, these are the *winding numbers* around p_1 and p_2 respectively.

For any edge (u, v) on L, we add the clockwise angle $\angle u p_i v$ to η_i. By *clockwise angle* we mean that if $\angle u p_i v$ is oriented clockwise, we add its positive value, else we add its negative value. Suppose p_1 is on the exterior and p_2 is on the interior of the region bounded by L, then we have $\eta_1(L) = 0$. The value of $\eta_2(L)$ will be either 2π or -2π depending on the orientation of L.

Thus we can reliably find the sum of values inside a closed loop L in the planar graph P.

Changing values. Unlike the case of mobile targets, if an arbitrary function h changes with time, local updates may not suffice. In particular, the local update scheme works only when the function has certain local conservation properties, such as when a change of δ in a face always causes a change $-\delta$ in an adjacent face.

Instead we simply re-initialize the form at regular intervals or on sufficient changes. With an initialization of cost $O(n)$, we create a network-wide one-form with which we can find the aggregate in any region of the network.

4.9 Completely mobile networks

Consider a network where all nodes are mobile. That is, beyond the targets, the sensors themselves are mobile. Our method naturally extends to such scenarios. As a sensor moves, it may cross an edge of the planar graph. Suppose that s crosses an edge e to enter a face τ. Then we update the network simply by first discarding all edges incident on s, then by inserting s into τ as in Figure 7. Many existing planarization algorithms work for mobile networks [13]. We can use such methods to maintain the graph. In all cases, the removal of an edge will not incur a cost, the insertion of an edge will be made according to the idea in Figure 7.

Care needs to be taken in cases where we are considering forms to monitor values defined on nodes. For example, when a mobile network tracks its own nodes to be able to answer aggregate counts and weighted sums inside regions. Suppose in such a case s crosses an edge $e \in \partial\tau$ to enter τ. Then along with the usual insertion, the value $h(s)$ must be reassigned to one of the new faces, for example by $\xi(e) := \xi(e) + h(s)$, as in section 4.8.

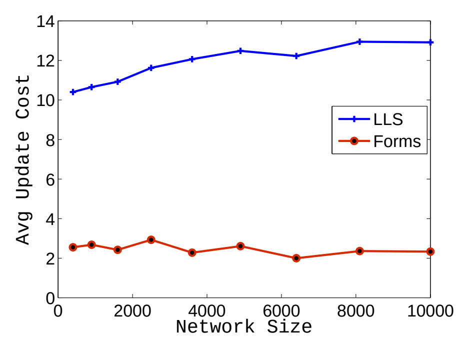


Figure 8. Average update cost per move.

5. SIMULATIONS

We conducted extensive simulation tests to see how the theoretical guarantees of our algorithm translate to a network graph and compare with LLS [1] in performance, particularly in terms of communication costs. In addition, we conducted simulations to test the

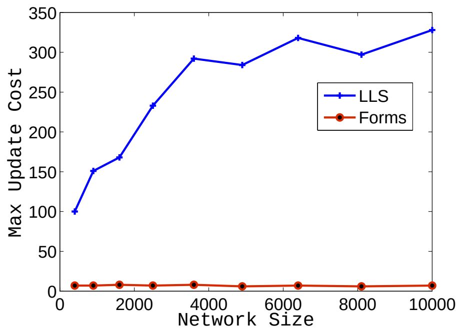


Figure 9. Max update cost for any move.

robustness of the algorithm to sensing failures and inaccuracies. This section describes the findings.

The simulations were done with networks that are quasi unit disk graphs[2] of inner radius $1/\sqrt{2}$. This choice of parameters allows local planarization algorithms [11, 26] to be used. The underlying sensor networks have nodes in a perturbed grid distribution, where the node is placed uniformly randomly in the grid box assigned to it. We consider networks without any significant coverage holes. In all cases, the average degree was about 10, and the size of the network was varied between 400 nodes and 10,000 nodes to test the scaling properties.

To evaluate the update costs, we introduce moving targets to the network domain. At each step, a target selects a random direction and moves up to a unit distance in that direction. After the move, the initial and final position are compared and updates are made.

5.1 Comparison with LLS

LLS scheme. This is a locality aware location service for mobile networks. The principle here is to use location servers at different levels. At each level $i = 0, 1, 2, 3, \ldots$ the network region is tiled by squares of side 2^i. The squares are aligned so that a square at level i is precisely covered by exactly 4 squares of level $i - 1$. In each square at each level, one node is designated to be the location server for that square, and keeps track of more precise locations of nodes in the square.

Location updates are performed in a certain lazy manner. Suppose mobile node p was in a square S_i at level i, and moves to a neighboring square at that level. The scheme does not update the location of p to the respective location servers. Instead, it waits until p has left this surrounding neighborhood of S_i before it actually performs an update. Thus, around S_i there is a ring of 8 squares moving where does not cause an update. As a compensation, LLS keeps its location information at the location servers of these nodes in addition to S_i. The idea here is to delay updates to avoid unnecessary communication. On average, if a node moves a distance d, then this scheme can be shown to have update costs of $O(d \log d)$. The cost is amortized. That is, the average cost is guaranteed to be low, but the update cost at a particular step can be arbitrarily high compared to the movement at that step.

The location search for a particular node starts at some other node in a network, and proceeds by searching nearby location servers at increasing levels. This goes on until some location server at the current or neighboring square for the current level claims to know the target location square at that level. Then the search proceeds in that square, successively searching lower levels. Of course, it is possible that due to the lazy update scheme, a server claiming to have the target is in fact in error. However in such a case, the target is guaranteed to be in one of the neighboring squares. It can be shown that this does not incur too high a cost. In fact, if the distance to the target is d, then the search finds the target at a cost of $O(d)$.

We compared costs with LLS in updates and query response. The following are the important observations:

- **Update costs.** Our algorithm adapts to node movements very efficiently. It has an average cost of about 2 messages per each unit distance move of the target, as compared to a cost of 10 to 12 messages for LLS. The maximum update cost for our scheme is about 7, while that for LLS is orders of magnitude higher — at 200 or 300 or more messages for a single small move. Most importantly, the costs of our scheme are independent of the network size, making it scalable to very large networks.

- **Search queries.** In answering queries where the one node searches for a specific target, our scheme performs slightly worse — consuming about 2 times the messages compared to LLS.

- **Aggregate range queries.** Given a geometric region such as a rectangle or ellipse, this query asks for the number of targets inside it. On this sort of queries, our scheme outperforms LLS by an order of magnitude.

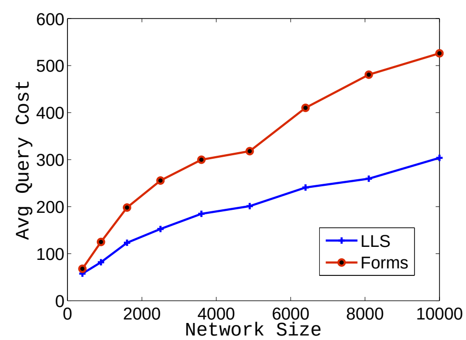


Figure 10. Average cost per search query.

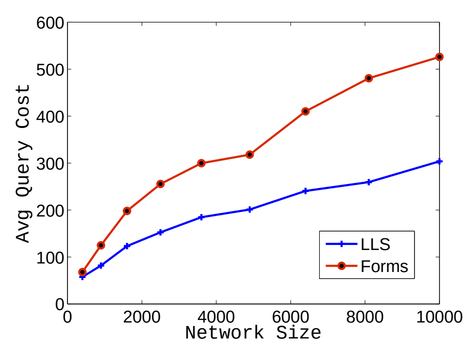


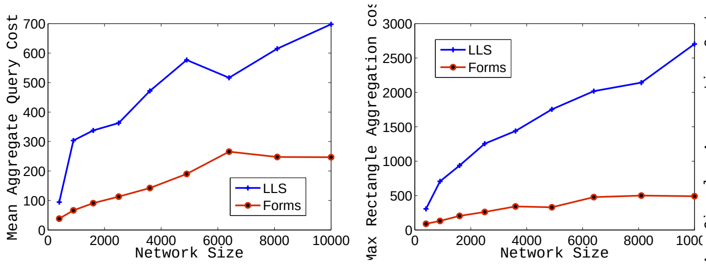
Figure 11. Max cost for any search query.

5.1.1 *Update costs*

As a target moves, the tracking system has to update its data to be consistent with the current target position. LLS does this by suitably sending updates to it location severs, while our scheme changes the weights on the edges crossed by the target.

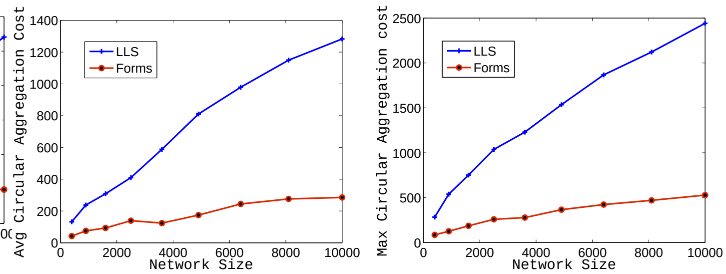
The results are shown in Figure 8. Our scheme is extremely efficient, since a small move does not cross too many edges, and the mean cost is about 2 per move. LLS is designed so that on certain

[2] A quasi unit disk graph is one where nodes more than unit distance away do not have an edge, nodes less than a distance r away always have an edge, and for other distances, the presence of an edge is uncertain.



(a) (b)

Figure 12. Aggregation query costs for random rectangle regions. (a) Average Costs, (b) Max costs



(a) (b)

Figure 13. Aggregation query costs for random circular regions. (a) Average Costs, (b) Max costs

moves, it does not require any updates. However, when the target has undergone sufficient displacement, it has to update several nearby lower level location severs - this incurs a reasonable cost. Later on, after further displacement, a move may require higher level servers further away to be updated, increasing the cost for that move, as well as the mean cost. The distance of the farthest server that may be tracking a target is proportional to the network diameter. After a proportional displacement this server will need to be updated as well. Thus, the update costs of LLS depend on the network size, though the amortized cost of LLS is still quite manageable, at about 10 to 12 messages per move.

The worst case behavior of LLS is poor. This is because the strategy of avoiding updates until necessary means that the updates build up and on certain moves neighboring servers and servers at several levels of hierarchy need to be updated. Thus the update cost of a single move can go into several hundred messages (shown in Figure 9). Our scheme, on the other hand, never has to update more than 8 edges.

Note that the costs in our scheme are taken to be proportional to the number of edge updates needed. In certain scenarios, where the target sensing does not require any communication, and when there is agreement among nodes on monitoring different parts of edges, it is possible to perform the updates at zero cost.

5.1.2 Search Costs.

Location service schemes are designed to answer queries that ask for the location of a specific mobile target, or to deliver a message to the target. Our scheme of tracking forms on the other hand was designed with aggregate queries pertaining to groups of targets in mind. Nevertheless, we find that it is a good instrument for search of specific targets, and has performance comparable to the location service scheme. We can maintain a tracking form ξ_i for each target T_i and then use that to search for it starting from the query node. The scheme is described in section 4.4.

In this experiment, we chose random query nodes, and random mobile targets. We execute a search for the target starting at the query node. The two schemes use analogous methods of searching exponentially growing regions for presence of the target, and in the suitable region searching exponentially smaller subregions until reaching the target. The asymptotic costs are the same for the two schemes. The simulation results in Figures 10 and 11 show that with tracking forms it costs about twice that of LLS to search.

In mobile environments, since updates are much more frequent than queries, the higher search costs of our method are compensated by the significantly lower update costs.

5.1.3 Aggregate Range Queries.

Given a region $\mathcal{R}$, say a rectangle or an ellipse, we wish to find the number of targets inside the region. With tracking forms, this is easy to do by summing the form in walk around the boundary. The details of the methods are in section 4.3. With a location sever scheme, the process is a little more complicated.

LLS maintains a quad-tree hierarchy, and recursively tracks nodes inside the quads at different levels. To find the aggregate, we need to look at quads of different levels that intersect with $\mathcal{R}$. In particular, if a quad Q intersects the boundary $\partial\mathcal{R}$, that means sub-quads of Q need to be analyzed further, to see which targets inside Q are actually inside $\mathcal{R}$. Therefore, the method boils down to finding quads at all levels that contain targets and intersect $\partial\mathcal{R}$. This turns out to be reasonably costly.

Figure 12 shows the costs when $\mathcal{R}$ is a random rectangle inside the network region. Figure 13 shows the corresponding costs when $\mathcal{R}$ is a random circle. Clearly, location server based schemes incurs a substantial cost in this type of query. Note that for target searching LLS actually uses a different quadtree hierarchy for each target. This would be impractically expensive in this sort of query, where the presence of each target in $\mathcal{R}$ will then have to be checked individually, driving the costs very high. We therefore used a common hierarchy where a location server can provide information about all targets in its quad region. Even with this modification, the costs of our scheme are still much lower, in principle only proportional to the size of the boundary of $\mathcal{R}$.

5.2 Effects of Target Detection Errors

Monitoring of mobile targets is not easy. Sensing errors and failures in communication can create difficulties for any tracking algorithm. Such failures occur at the physical layer and in effect supply the algorithm with incorrect input. A tracking algorithm should be robust, so that its performance degrades gracefully and slowly with increasing sensing errors.

This subsection tests the effects of such failures on the quality of aggregate results returned by our method. As targets move we compute the aggregate in arbitrary ranges using the tracking form and compare with the true aggregate of the range. We consider two types of errors:

1. **Failure to detect a target crossing an edge.** For example, a sensor monitoring the edge fails to detect the target passing. This can also happen when targets are responsible for supplying their own tracking information. For example, a targets crosses an edge into a new face, but its message notifying this move gets lost. In such cases, the tracking form on the edge will not be updated, and certain queries may return incorrect results.

2. **Incorrect Estimation of Target Location.** The location of a target computed by the system may be incorrect. For example, signal strength based localization may be erroneous, or even GPS based location computed by a target itself may be off by several meters. In such cases, the object will be

estimated to be inside a different face than where it really is, and will contribute an error to the computed aggregate.

In these simulations we consider a variable number (between 20 and 300) of targets moving in the plane, and are tracked by a differential form on a 100×100 unit grid. A target takes steps in random directions and within a unit length as before. As targets move, we execute queries to count the number of targets within a unit square chosen randomly within the grid. For each such query, we take as error the difference of the computed result with the actual number of targets in the range. This error has a dependence on the number of targets in the system. We measure the *relative error* – the ratio of the error to the number of targets and see how that changes with increasing number of targets.

To simulate the first type of errors, we select a probability p as the probability that a target is not detected when crossing an edge. The parameter p in that sense represents the sensing accuracy of the system. We vary p over a wide range of values from 0.05 to 0.70 that is, we vary it upto the the case where 70% of edge crossings are missed. For each p and number of targets we execute 100 range queries on random axis-aligned squares. We let the targets make 2 moves between successive queries.

The results are shown in Figure 14. The values of the errors are

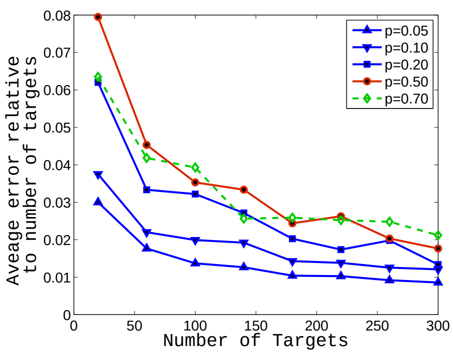


Figure 14. Error induced by failure to detect targets crossing edges. The error in counting relative to the total number of targets, plotted against the total number of targets; for counting number of targets in random axis aligned squares. The parameter p is the probability that a targets crossing and edge is not detected.

very small. Even for severe values of p reaching upto 50% or 70%, the counting error is less than 8% of the target count, and drops rapidly to less than half of that for 100 targets or more. For more reasonable values of p such as $10\% - 20\%$, the errors are just a few percents.

The curve for $p = 70\%$ fits the pattern less tightly than the others. Its high error rate causes it to fluctuate and behave more unpredictably at low number of targets. As number of targets increases, it stabilizes better, and ends with a higher relative error rate than the other curves with lower p values, as expected.

The relative error decreases with increasing number of targets. This is because statistically the effects of over counting and under counting cancel each other, and this happens more reliably with larger number of targets.

In simulation of the second type of errors, we assign each target a location different from its true location and compare the true and computed counts as before. The assigned location is intended to simulate the estimated and possibly incorrect location of the target. The estimation cannot be very far from its true location, since the location of sensors or access points that detect the target can be used to restrict the region within which the target must lie. Therefore we use a parameter *localization radius (LR)* which limits the maximum distance from the true location within which the estimated location must lie. The estimated location is taken to be a random point within this radius. We vary LR from 0.1 to 5.0 units. And as before, we carry out 100 random queries for each LR and different number of targets, with the targets moving twice between successive queries. The results are shown in Figure 15. Once again,

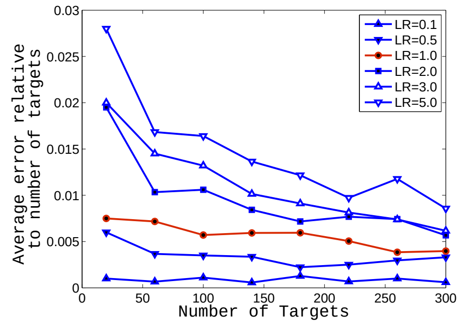


Figure 15. Error induced by incorrect localization of targets. The error in counting relative to the total number of targets, plotted against the total number of targets; for counting number of targets in random axis aligned squares. The parameter LR is the maximum distance between true and estimated locations of targets.

the we find that the relative error drops with increasing number of targets. In this case, the error rates are even lower, staying below 3%, and in most case at about $1\% - 2\%$ or lower.

The overall conclusion is that the method is extremely robust to failures and sensing noises of different types. On average it incurs only small output errors even with large probabilities failures. The errors degrade gracefully with increase in failures. This is largely the result of the local nature of the tracking mechanism: if an edge is not updated, that failure does not affect a query unless the edge lies at the boundary of the query region.

6. DISCUSSIONS

Networks Without Locations. A *range or neighborhood* is a topological concept, and so is a range query. A differential form is a topological construct and can be defined abstractly without use of coordinates. Therefore, this minimal scheme is applicable without the use of locations. It is possible to obtain a planar graph without using node locations [29]. After that it is possible to determine a consistent orientation and create a tracking form abstractly. The ideas from subsections 4.7 and 4.8 can then be used to track and query the form inside any given loop.

Geometric data such as the locations of nodes and description of the range can be helpful is executing a query, but not essential. Existing methods [1, 12] commonly use hierarchical quadtree type partitions that rigidly depend on a geometric processing of the data. This makes such schemes unsuitable for use in a coordinate free environment.

Mobility Models of Targets. Throughout the paper we have assumed that the target can move in an *arbitrary* manner. Since updates are completely local, the cost is bounded by the total distance traveled by the targets, not how they move, assuming that small oscillating motions are handled in an efficient way as in section 4.5. The performance of LLS is affected in some degree by the mobility patterns of the targets. In particular, linear motion will again drag the squares along leading to the worst-case update cost of $\Theta(d \log d)$ where d is the total distance moved. But local oscillating type of motion when a target does not move too far from its original location will keep the updates limited to local location servers. Thus the maintenance cost can be lower than the upper bound.

Network Power Management. In a network with mobile entities, it can be expected that targets move often. Our scheme handles the movements very efficiently and locally. There is never any need to send updates to a distant point. This is also significant from power management point of view. If a target of interest is present in a part of the network, nearby nodes can be expected to be awake and actively monitoring it. If all movements are handled locally, then relatively distant nodes can sleep or go to low power mode to save energy without fear of interruptions.

Schemes that recruit distant location servers or a global central server for target tracking will need to keep most of the network on for target update at far away location servers and routing to them.

7. CONCLUSIONS

In this paper we presented the use of differential one-form in the application of target tracking and range queries. The method is simple, has low maintenance cost under target movement, is extremely flexible and robust to network changes and node mobility. The performance of our method is orders of magnitude better than previous location services schemes for tracking mobile targets. We expect that more applications can be found that use the differential one-form for a diverse set of queries of aggregated data, which we will investigate in the future.

Acknowledgement: Jie Gao and Rik Sarkar acknowledge the support from NSF CAREER Award CNS-0643687. We thank Somnath Basu for some very helpful discussions on differential forms, and the anonymous reviewers and Lili Qiu for their many suggestions on revising the paper.

8. REFERENCES

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: a locality aware location service for mobile ad hoc networks. In *DIALM-POMC '04: Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pages 75–84, 2004.

[2] P. Agarwal and C. M. Procopiuc. Advances in indexing for mobile objects,.

[3] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[4] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161, 2003.

[5] B. Awerbuch and D. Peleg. Concurrent online tracking of mobile users. In *SIGCOMM '91: Proceedings of the conference on Communications architecture & protocols*, pages 221–233, 1991.

[6] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *Int'l J. High Performance Computing Applications*, 16(3):90–110, 2002.

[7] J. Faruque and A. Helmy. RUGGED: Routing on fingerprint gradients in sensor networks. In *IEEE Int'l Conf. on Pervasive Services (ICPS)*, pages 179–188, July 2004.

[8] J. Faruque, K. Psounis, and A. Helmy. Analysis of gradient-based routing protocols in sensor networks. In *IEEE/ACM Int'l Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 258–275, June 2005.

[9] R. Flury and R. Wattenhofer. MLS: an efficient location service for mobile ad hoc networks. In *MobiHoc '06: Proceedings of the seventh ACM international symposium on Mobile ad hoc networking and computing*, pages 226–237, 2006.

[10] W. Fulton. *Algebraic Topology: A First Course*. Springer, 1997.

[11] S. Funke and N. Milosavljević. Network sketching or: "how much geometry hides in connectivity? - part II". In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 958–967, 2007.

[12] J. Gao, L. Guibas, J. Hershberger, and L. Zhang. Fractionally cascaded information in a sensor network. In *Proc. of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 311–319, April 2004.

[13] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanners for routing in mobile networks. *IEEE Journal on Selected Areas in Communications Special issue on Wireless Ad Hoc Networks*, 23(1):174–185, 2005.

[14] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 163–173, Anchorage, Alaska, May 2003.

[15] L. J. Guibas. Sensing, tracking and reasoning with relations. *IEEE Signal Processing Magazine*, 19(2):73–85, March 2002.

[16] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.

[17] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Trans. Sen. Netw.*, 2(1):1–38, 2006.

[18] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 243–254, 2000.

[19] W. Kim, K. Mechitov, J.-Y. Choi, and S. Ham. On target tracking with binary proximity sensors. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 40, 2005.

[20] C. Kinsey. *Topology of Surfaces*. Springer, 1993.

[21] B. Kusy, A. Ledeczi, and X. Koutsoukos. Tracking mobile nodes using RF doppler shifts. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 29–42, New York, NY, USA, 2007. ACM.

[22] J. Li, J. Jannotti, D. Decouto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of 6th ACM/IEEE International Conference on Mobile Computing and Networking*, pages 120–130, 2000.

[23] X. Li, Y. J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 63–75, 2003.

[24] H. Lin, M. Lu, N. Milosavljević, J. Gao, and L. Guibas. Composable information gradients in wireless sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'08)*, pages 121–132, April 2008.

[25] J. Liu, F. Zhao, and D. Petrovic. Information-directed routing in ad hoc sensor networks. *IEEE Journal on Selected Areas in Communications*, 23(4):851–861, April 2005.

[26] R. Sarkar, X. Yin, J. Gao, F. Luo, and X. D. Gu. Greedy routing with guaranteed delivery using ricci flows. In *Proc. of the 8th International Symposium on Information Processing in Sensor Networks (IPSN'09)*, pages 97–108, April 2009.

[27] N. Shrivastava, R. M. U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 251–264, 2006.

[28] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 529–538, New York, NY, USA, 2007. ACM Press.

[29] F. Zhang, A. Jiang, and J. Chen. Robust planarization of unlocalized wireless sensor networks. In *Proc. of INFOCOM 2008*, pages 798–806, 2008.

[30] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration. *IEEE Signal Processing Magazine*, 19(2):61–72, 2002.