Geometric Methods of Information Storage and Retrieval in Sensor Networks

Rik Sarkar
Department of Informatics
The University of Edinburgh
United Kingdom
rsarkar@inf.ed.ac.uk

August 20, 2013

Abstract

Sensor networks collect data from their environment. Locations of the sensors are important attributes of that information and provide a context to understand, and use sensor data. In this chapter, we will discuss geometric ideas to organize sensor data using their locations. We will consider distributed methods for managing queries about isolated events, queries about mobile objects, and queries for general signal fields. Location based methods often operate on simple geometric domains, and we will discuss how they can be adapted to networks with complex shapes.

1 Introduction

The data collected by a sensor has to be available to others, since the sensor that produces a piece of datum is not always the one that uses it. The consumer may be far from the source of data and have no idea of how to find the one relevant source in a large network. When the consumer asks for aggregate information, for example, sum or average or maximum, we face a similar problem of handling data from a large group of sensors at a large communication cost.

Methods of data pre-processing distribute hints about sensor information across the network. When done properly, pre-processing can make it easier to answer consumer queries – search and aggregation – and avoid the large costs.

Locations are a useful tool in pre-processing and an important aspect of sensor data, since physical events are naturally associated with coordinates instead of ids. Locations let us associate events with the physical world – they give us an *index* of the environment; thus locations are essential to *Cyber Physical Information.* Interpreting and processing data by locations gives a geometric structure to the otherwise amorphous sensor readings. As an added bonus, basic network operations of routing, communication, scheduling and many others can benefit from use of locations. Since wireless communication works only between nearby nodes, their locations have a close relation to the overall structure of a network.

In this chapter, we will examine how geometric ideas can leverage the locations of nodes for better utilization of sensors. We will see that geometry plays a role in processing the sensor data as well as in answering queries.

These location based methods can be divided into two categories by their applicability. In Section 2, we discuss ways of managing data in a static scenario where sensors measure general physical quantities or events such as temperature, pressure or occurrence of fire. In section 3 we will consider methods for processing data about mobile objects. With the increasing popularity of mobile devices, this is an important sensing category, and carries interesting relations to the general case. In each of Section 2 and 3, we will consider two different styles of data organization – hierarchic data structures, and data distribution in a flat structure. In Section 4, we will discuss how these methods are adapted to complex networks by construction of virtual coordinates and segmentation of networks.

Our goal is to keep the methods as general as possible, so that they are applicable to the widest variety of scenarios. Therefore we will treat our network as general communication-capable sensors distributed in a plane, without any assumption on their specific sensing capabilities or other features. We will, however, mention relevant example applications in each case to illustrate the methods and their uses.

In the rest of this section we discuss the general model and scenario used to describe the different methods. Locations and distributions of nodes are important to storage schemes, as is routing. Let us briefly review the aspects of these topics that will be relevant to our main discussion. Readers generally familiar with location based algorithms and routings methods may wish to skip ahead to the next section.

1.1 Distribution and Location of Nodes

Finding locations of nodes is a challenge on its own and much research has been devoted to it. The methods and protocols vary by the requirements and the infrastructure available, and the theoretical questions related to localizing nodes are often intractable problems [10, 4]. For more related works on this topic see a recent survey [15]. From a practical point of view, GPS is becoming more affordable, and localizing sensors by collaborating with nearby and passing GPS enabled devices is often a possibility. Wireless and cellular signal based localizations are also becoming and common and fairly reliable. Very accurate localization will not be important to our discussions, we therefore leave the question of localization here and assume that some form of approximate locations are available.

We do need some idea of how sensors are distributed in the domain. For our discussion, let us assume that the nodes are distributed over large area and with bounded density: the number of neighbors of any node is bounded by some constant number. This is a reasonable assumption, since in a small region we would like to have only a limited number of sensors. Too many devices in close proximity increase costs and reduce communication efficiency [16], but cannot provide corresponding sensing benefits.

For simplicity, we can discuss the performance of algorithms with respect to a specific sensor configuration. Let us suppose n sensors are distributed in a large enough square and with uniform density as discussed above. If communication ranges of sensors are bounded above and below by some constants, this would mean that the sides of this square are of length $\Theta(\sqrt{n})$, and the diameter of the network is of the same order. The expected distance between any two random nodes in the network is also $\Theta(\sqrt{n})$.[1] We will discuss more general types of networks in section 4.

1.2 Communication and Routing

To make best use of sensor networks, a sensor needs to communicate with other nodes; it needs support from the network to forward its messages. Multi-hop routing in wireless and sensor networks is a widely studied subject that we will not attempt to survey here, but will mention briefly a few concepts important to our later discussion.

Flooding. This is perhaps the simplest communication technique, where the message is sent to all neighbors of the source, and a receiver always sends it to all its own neighbors. As a result, the message reaches all nodes in the connected network, including the intended destination. Once the first message is delivered, one of the paths along which it traveled can be used for further communication between source and the destination. This is the basis of classic ad hoc routing protocols such as AODV [35] and DSR [18]. The cost of such a protocol is $\Theta(n)$ per communication pair, since the first message goes to all nodes.

Geographic routing. To make routing more efficient in sensor networks, several methods make use of node locations. These schemes preprocess the network to compute a planar graph whose edges consist of communication links in the network (see [2]). The routing itself follows a two phase method. Suppose node s currently has a message for location t, then s uses one of the following tactics:

1. *Greedy routing:* Node s checks all its neighbors and finds neighbor w that is nearest to t. If $|wt| < |st|$, that is, w is nearer to t, then s sends the message to w.

2. *Perimeter mode routing:* If no such w is available, the routing enters perimeter mode, where the message moves along the face of the planar graph containing s, until it finds w with $|wt| < |st|$.

See Figure 1 for an example.

[1]A function $f(n)$ is said to be $\Theta(g(n))$, if there are constants a, b, N such that for all

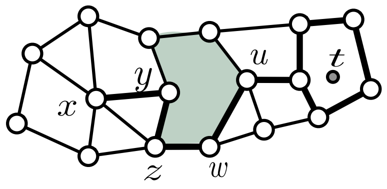


Figure 1: Greedy routing and face routing. Message path is shown in bold, and goes through the following steps. A message for t starts from x and reaches y by a greedy step. Greedy step fails at y, so perimeter mode is initiated for the shaded face. Message reaches w while traversing the face. Greedy mode resumes at w since $|wt| < |yt|$. The message traverses the face containing location t looking for the node nearest to t.

There are several methods [2, 19, 22, 25, 26], that are variations of this essential strategy. The most popular among these is GPSR [19], which we will use as our standard reference for this class of greedy plus perimeter mode routing. We will assume for simplicity that the sensors are distributed with sufficient density that there are no large "holes" in the square network. As such, we can say that if two nodes are distance d away in the plane, the GPSR path between them has a length $O(d)$ – which is the communication cost between these two sensors. If there is no node at the destination location t, then GPSR traverses the face containing t, and arrives at the node closest to t.

Virtual coordinates and handling network shapes. A real sensor network will typically not be so simple as the square domain we have assumed. It will have an unknown shape, and will likely have coverage holes where there are no sensors. Is it still possible to apply the routing methods and the geometric data storage methods to these networks?

Protocols have been designed to compute *virtual coordinates* – assignment of a logical or virtual location to nodes in an abstract plane [36, 34, 5, 6, 39, 40, 45]. These methods morph the virtual network into more standard shapes, making it easier to use routing and geometric data storage methods. A different approach is proposed in [46, 47] - to decompose a complex shape into pieces, each of which is relatively simple, and easy to apply geometric methods.

We will discuss these methods of handling complex shapes in Section 4.

$n > N$, $a \cdot g(n) \leq f(n) \leq b \cdot g(n)$. That is, for large enough n, $f(n)$ behaves like $g(n)$ to within constant factors. More details can be found in books on algorithms. See for example [7].

2 Information Brokerage and Range Queries

An important sensor network question is searching for particular pieces of information. Sensors detect and store significant *events*, and sometimes we need to find a particular type of event – for example, a tourist on safari may wish to find the elephants[2]. In this case, we have a sensor network monitoring the park, and some sensor P near a group of elephants has useful information – it has stored an “elephant detection” event. But sensor P is not aware where an interested tourist may be. Symmetrically, node C is in communication with the tourist, but has no idea where P is.

This general problem is called *Information Brokerage*: information *producer* P has some data, and information *consumer* C would like to learn this data. We need a general method by which producer and consumer can find each-other easily.

The conceptually simplest protocol to handle this problem is for the consumer to check every node in the network by flooding a query. This approach was taken in Directed Diffusion [17] and TinyDB [31]. It works well when data is updated often, but queries are occasional – a sensor stores updated information, and responds to the queries it receives. However, all other nodes in the network are also required to receive and forward the query, whether they have relevant data or not. When searches are frequent, this becomes an unnecessary load on the nodes, most of whom have nothing to do with the query.

We need an information brokerage scheme that makes better use of network resources – uses less communications, and balances the load across the network. We would like to avoid consuming energy at nodes that can not help us in our search. All nodes reporting their data to a single server achieves this in a way – it avoids flooding the network, but also overloads the server and the nodes close to it. All the updates and queries have to be forwarded by the few nodes leading to the server, which will quickly run out of battery. Information should be spread out to balance the storage and communication loads in the network.

2.1 Hashing Data to Points and Curves

One method to coordinate producers and consumers is by using consistent hashing over the entire network. The idea is used in [37]. All nodes know a hash function that can be applied to the data query or key, and it returns a location. Let us denote this function as h.

In our example, sensor P performs the hash $h(\text{“}\textit{Elephant}\text{”})$, and obtains a location – a pair of coordinates: (x, y). Node P sends a message to the sensor at (x, y) that “Elephant” or relevant data is available at P’s location. The consumer C performs the same hash when searching and obtains the same (x, y); thus C knows which sensor must have the information.

Location (x, y) need not be a sensor location at all, a random hash h will almost certainly give us an empty location. We can resolve this by storing the

[2]This commonly used example is from [37]

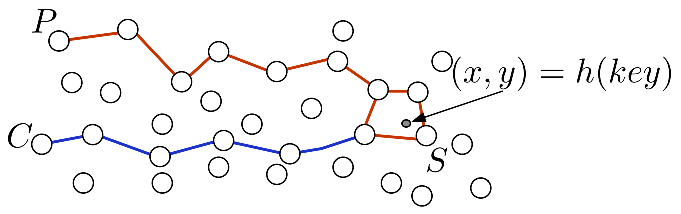


Figure 2: Geographic hash tables. The data producer P sends a message to location $(x, y) = h(key)$, using GPSR. The trajectory is shown as the path in red. The data may be stored at the node S nearest to the location, or on the entire perimeter around the hash location. The consumer also performs GPSR with (x, y) as target and arrives at the data.

data at the sensor nearest to the location (x, y) – see Figure 2. This nearest sensor is called the home node. We saw in Section 1.2 how the home node can be found using GPSR [19]. The consumer can also send a message via GPSR routing to (x, y) to arrive at this same sensor, and a return message retrieves the data. Since hash locations are completely random, the expected cost for a node – either a producer or a consumer – to send a message to the hash location is $\Theta(\sqrt{n})$. Therefore, the cost of a producer storing a piece of information at the hash location, or that of a consumer retrieving it are both asymptotically $\Theta(\sqrt{n})$ in expectation.

Storing a piece of data at only the home node is fragile – a failure of the node can destroy all the stored information. For better fault tolerance, GHT utilizes the GPSR's perimeter routing mechanism. While searching for the home node, GPSR traverses the perimeter of the face that contains the hash location, and GHT stores the data at all nodes on this home perimeter at no extra communication cost. Periodically, a probe is sent around the home perimeter, checking the health of the perimeter nodes. If some nodes have failed and the perimeter changed, then the new perimeter nodes are given a copy of the data.

The perimeter mode storage of data has an additional advantage that the consumer can get the data as soon as the search message hits some part of the perimeter curve. If we had gone a step ahead and stored the data at all the nodes on the path from the producer up to the home perimeter, and sent the consumer's search message in a path likely to touch the storage path, it could have collected the data without visiting the home region.

Can we stretch this idea further? Maybe we can deliberately send the producer's message along a path that the consumer can find. If done suitably, this method may have several additional benefits. In Figure 2, the producer and consumer are fairly close, yet the search message has to travel to the hash location and back; which means slower response to the query, and additional load on the home nodes and those on the path. If the search found a nearby node on the producer's storage path, these unnecessary penalties would be avoided.

Our next brokerage technique, called Double Rulings[43, 41] expands this concept, and stores data along a path instead of at a node. The challenge is to design the trajectories such that the search paths find the storage paths easily.

2.1.1 Double Rulings

Double Rulings is the general idea of storing data on a path such that the consumer's search finds it easily. Simply taking the GPSR path to the hash location does not suffice – the goal is to have the intersection of search and storage paths close to the consumer. The close intersection means less communication and faster response to the query. It will be best to have the intersections at different points on the storage path and have the load of responding to queries more evenly distributed. The specific scheme we will discuss is from [43, 41].

The intuition is to design the paths as abstract curves on a sphere. A type of curves we can use are called *great circles*. These are circles that lie on the sphere and have the largest possible diameter. On the earth for example, the equator and the longitude circles are all great circles. The great circles on a sphere are analogous to straight lines on a plane – shortest distances are measured along them. Just as with the plane, given two points on a sphere, there is a unique great circle through them[3].

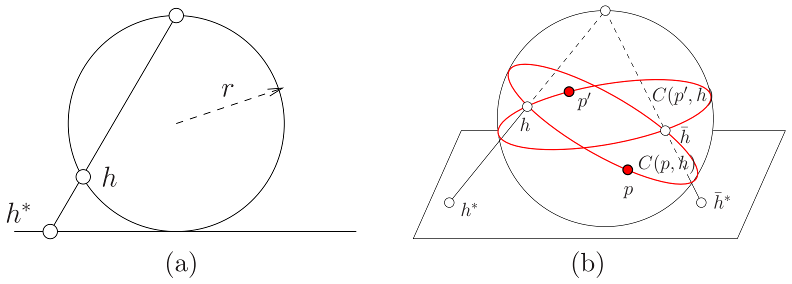


Figure 3: (a) Stereographic projection in side-view. (b) Producer storage curves (red) for p and p'. They are great circles passing through hash location h.

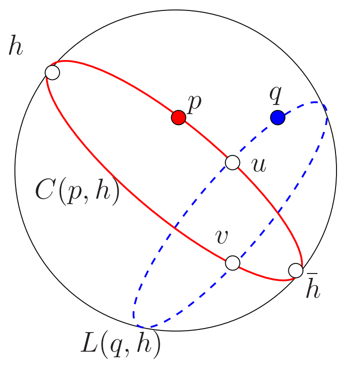
The sensor network itself lies on a plane. To make use of curves on a sphere, we first need a correspondence between the sphere and the plane. This is done through a mapping called *stereographic projection*. Imagine the sphere is placed on the plane. For any point h^* on the plane, we imagine the straight line from h^* to the north pole of the sphere – its top most point. This line intersects the sphere at another unique point h, which is the image of h^* under the stereographic projection. This idea is shown in Figure 3(a).

With this map between the plane and the sphere, we can now define the storage and search curves in terms of curves on the sphere. An example of

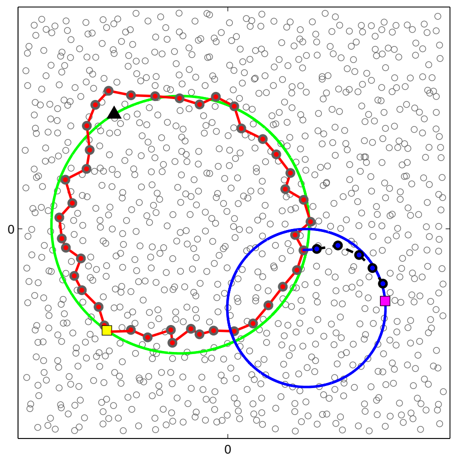
[3]The exception to this is the special case when the two points in question are antipodes to each-other on the sphere, and there are an infinite number of great circles passing through them.

storage curves is shown in Figure 3(b) where two producers are mapped to points p and p' on the sphere. In double ruling, the hash function gives a location h on the sphere. The data is stored by a producer along the great circle curve on the sphere passing through itself and h. To be precise, data is stored along nodes on the curve in the plane that is the stereographic map of the storage curve on the sphere. By properties of stereographic projection, the circular curves on the sphere map to circles in the plane. Thus the producer's cost for storing data along any circle in the network field is at most $O(\sqrt{n})$.[4]

Using curves on a sphere makes data search and retrieval easy: a great circle intersects any other great circle. Thus, the consumer doesn't even have to consider the hash h – it can simply send a message on a great circle and find any data in the network.



(a)



(b)

Figure 4: (a) Double Rulings p and q are stereographic maps of producer and consumer; h is the hash, $\bar{h}$ is its antipode. $C(p,h)$ is storage curve (a great circle) and $L(q,h)$ is the retrieval curve, intersecting at u and v. (b) Actual Network storage path (red) and search and retrieval path (blue) in the plane. The hash location is shown as black triangle.

The knowledge of the hash function can be used to create smarter paths that have provably small search cost. We make use of *latitude curves* for this efficient search and retrieval in place of great circles. A latitude curve is defined to be one that maintains constant distance to the hash location on the sphere – see Figure 4(a)[5]. The advantage of such curves is that they intersect any producer

[4]A function $f(n)$ is said to be $O(g(n))$, if there are constants b, N such that for all $n > N$, $f(n) \leq b \cdot g(n)$. That is, for large enough n, $f(n)$ is less than $g(n)$ to within constant factors. See [7] for details. The largest possible circular arc has length $O(\sqrt{n})$ in a square of length $O(\sqrt{n})$.

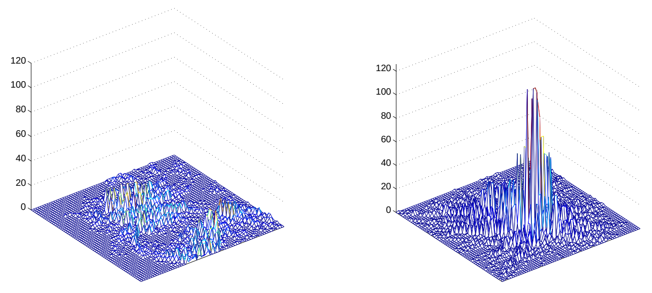
[5]The name derives from latitudes on the earth, that maintain a constant distance to the poles.

curve within a distance $O(d)$ from the consumer, where d is the distance between the consumer and the producer on the sphere. In Figure 4(a), this means that either u or v – must be close to the consumer q. The location and radius of the sphere can be chosen such that distance between any two points on the plane is at most a constant times the distance between corresponding points on the sphere, and the length of a path on the sphere is within a constant factor of the length of the corresponding path in the plane produced by the stereographic projection. Thus, the cost of search and retrieval for the consumer is $O(d^*)$, where d^* is its distance to the producer in the plane. This property is called *distance sensitive retrieval.*

The storage and search paths in the plane are shown Figure 4(b). In a discrete network the paths cannot be the smooth curves we construct by projection. The network path following the curves as shown in the figure are obtained using a general strategy of "Routing along a curve" described in [33].

Double Rulings includes GHT as a subcase – every storage curve passes through the hash location, thus the consumer can always retrieve data from there. This is a useful feature when a consumer wants all the data of a particular type instead of just one – this can be done by visiting the hash location. On a sphere, its antipode $\bar{h}$ serves as an additional proxy hash location – any great circle passing through h also passes through $\bar{h}$, and the GHT style retrieval can be performed by visiting its image in the plane.

In data searching, the great circle curves are useful for certain types of searches. For example, when the consumer asks for multiple types of data at once, a great circle retrieval can find all of them at once, since the a great circle will intersect all other great circle producer curves.



(a) (b)

Figure 5: Communication load at different nodes. Information generated by one producer, and queries issued by 500 consumers. (a) Load for double rulings is distributed. (b) Load when using GHT is clustered around the hash location, and therefore poorly balanced.

One of our goals in using in-network storage is to balance the data handling load among the nodes. Double ruling has the advantage that different consumers retrieve data from different parts of the producer curve, so that the tasks of responding to query are better distributed. This effect can be seen in Figure 5. GHT creates high load near the hash location. The load from double rulings is more balanced. In fact, the overall load from double rulings is less, since it requires less communication per query.

Location free double ruling schemes. Intersection of paths to achieve information brokerage has been used based on other structures than the stereographic projection described in [43, 41]. We describe these methods very briefly, making use of Figure 6. The first method, shown in Figure 6(a) is called Rumor Routing, described in [3]. Here, the storage path is a random walk from the producer, and of some maximum length determined beforehand. The retrieval path is also a random walk, starting at the consumer. When the retrieval path meets the storage path, it finds the data, which is returned to the consumer.

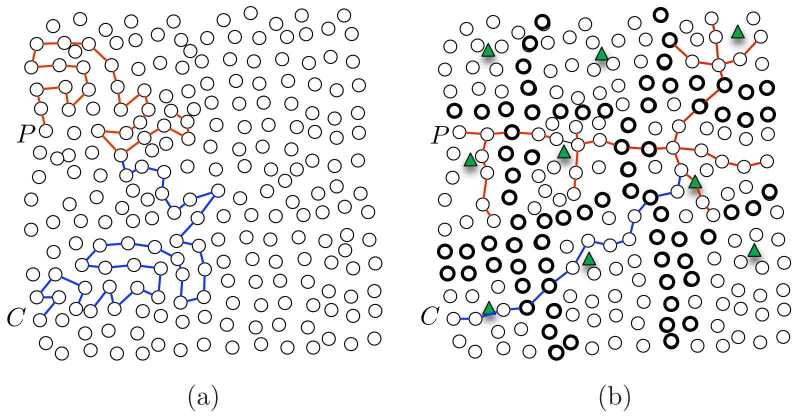


Figure 6: (a) Rumor routing: Producer follows a random walk. Consumer also follows a random walk until it hits the producer. (b) Landmark based double ruling. Landmark nodes are shown as green triangle. Hash to a tile instead of location, then follow GLIDER routing to reach tile. The nodes in bold are on boundary of one or more tiles.

The second method (Figure 6(b)) relies on landmarks to decompose the network, and is described in [12]. A few nodes in the network are selected as *landmarks*. All other nodes determine the landmark nearest to them, and are grouped into "tiles" accordingly. This is easy to do by flooding messages from the landmarks, and using that to measure the distance of each node from the different landmarks. The hash in this case is a complete tile. The producer sends the data using a related landmark based routing scheme described in [11]. In each tile the path passes through, it shoots off additional branches so that a consumer path finds an easy intersection. The consumer uses the same hash and routing scheme, and intersects a storage node in the hash tile or an earlier one.

These methods have the advantage that they do not need locations to operate. Thus, they can be used when GPS or similar infrastructure are not present. However, when locations are available, they are substantially more expensive than the methods we discussed making use of locations. In the next subsection we return to our main topic of location based schemes and discuss how they can be divided recursively for better query response.

2.2 Hierarchical Partitions

Recursively partitioning a space is a common technique in data storage and search mechanisms. The reader may be familiar with the binary partitioning used in binary search on an array – where at each step the array is divided into 2 parts. By creating an abstract node for each such part, we get a corresponding abstract structure called a binary search tree, that represents the recursive partitioning of the array. To apply similar techniques to sensor nodes in a plane we need a two dimensional version of this idea.

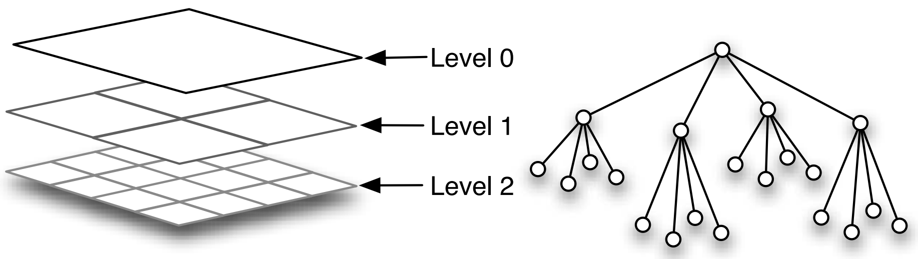


Figure 7: A 3 level quadtree. On the left is the recursive partitioning – each level consists of one or more square areas. At the next level each square is split into 4 congruent squares. On the right is the tree – each square becomes a node, with edges to its children, and the parent.

Figure 7 shows a *quadtree partitioning*. We start with a square space at the top level, and recursively partition it into smaller congruent squares at each level. Recursive partitioning gives rise to an abstract tree structure shown on the right. Each square at each level corresponds to a node in the tree, thus nodes other than the leaves have 4 children each. We can interchangeably refer to a square or corresponding quadtree node as convenient. The partitioning in Figure 7 has two levels in addition to the root, in general we can have any number of levels. It is reasonable to assume that the final level is one where the squares are unit sized. In our constant density square network model, unit sized leaves imply a quadtree with $\Theta(\log n)$ levels.

This general partitioning scheme has been used in different ways for sensor data handling. Here we briefly discuss a few of these.

2.2.1 Structured Replication in GHT

GHT [37] has a variant called structured replication (Figure 8) designed to handle the hotspot problem of many producers trying to transmit updates to a single hash location. In this method, each node considers a quadtree partitioning of the square. On each level of the partitioning, it is possible to perform the hash on each square at that level, giving us 4^i hash location at each level i.

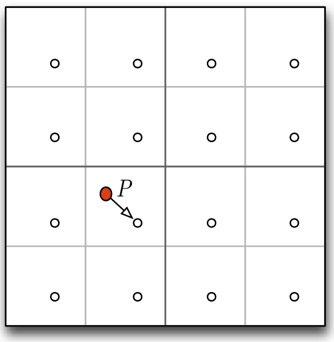


Figure 8: Structured replication in GHT has a hash location in each square of a quadtree partitioning. A producer (shown shaded) stores the data at the nearest hash location among all these.

The producer P stores the data at the single nearest hash location among all the locations. In a k level tree storage cost is reduced to $O(\sqrt{n}/2^k)$, but the search cost increases – now the consumer has to search multiple locations. The protocol dictates that the consumer searches the hash at level 0, which automatically forwards the message to the level 1 hashes, each of which forwards the message to level 2 hashes and so on. Thus the search cost is $O(2^k\sqrt{n})$. This method therefore decreases storage or update cost, but increases the search cost. The authors point out that if the levels are such that at the lowest level squares are constant sized, then this costs $O(n)$ – asymptotically the same as flooding to retrieve data from the source node.

2.2.2 Fractional Cascading and Aggregate Information

Let us consider now a different question – answering aggregate queries using a hierarchical structure. Imagine all our sensors are monitoring a signal that has an attribute value, such as temperature. We can thus ask a question "Which sensors in region R have temperatures above T?" or, "How many sensors in region R have temperatures above T?" To answer these, we need a different type of data storage and retrieval than the information brokerage schemes.

The fractional cascading method [14] suggests storing at each sensor, aggregate data about exponentially growing regions: a constant number of values for each quadtree square that contains it. A sensor has detailed information about the local neighborhood, and progressively coarser information about larger regions.

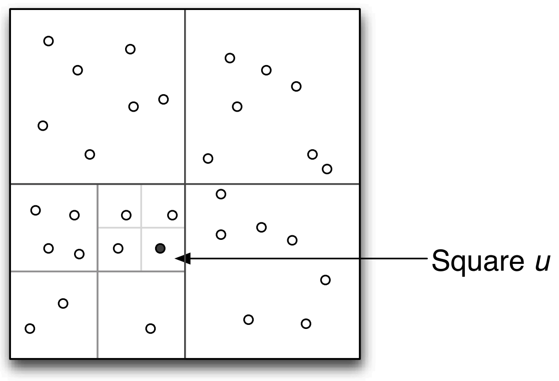


Figure 9: The network from the point of view of the shaded node in quadtree square u. It stores one aggregate (for example maximum) for each square in this figure.

The method starts with partitioning the square with a quadtree. Remember that a node u in the quadtree corresponds to a square in the partitioning. Its parent $p(u)$ is the larger square at the previous level that contains the square u. Fractional cascading then proceeds by storing some values at each sensor in a square. At each sensor in u, it stores the maximum in the square u, and the maxima for each sibling of u in the quadtree. That is, each node saves 4 values for each level in the tree – the quadtree nodes on the path from the leaf to the root, and their siblings (See Figure 9). This means that on a typical tree in our scenario, a node needs to store $O(\log n)$ values, and the communication cost of storing all the data in this format is $O(n \log n)$.

The query response is done in terms of *Canonical Pieces*. Given a region R, a canonical piece is a square in the partitioning that fits completely in R, but its parent does not fit – see Figure 10. To find the true answer, we check each canonical piece. The cost of this traversal can be shown to be $O(D + \sqrt{Ak} + P \log P)$. Here A and P are the area and perimeter of R, while k is the number of sensors with temperature above T. The parameter D is the distance from the query source to the query range – it is the unavoidable cost of communicating with the query region.

Sometimes we may not be interested in a such a detailed report. We may just ask "What is the maximum temperature in R?" to find out if there is a fire in the region. On such a query, it is sufficient to check just one node from each canonical piece, since every node stores the maximum value of each square it belongs to. The traversal can be done nicely by following a spiral path in R visiting the smaller pieces at the edges first, and traversing progressively larger

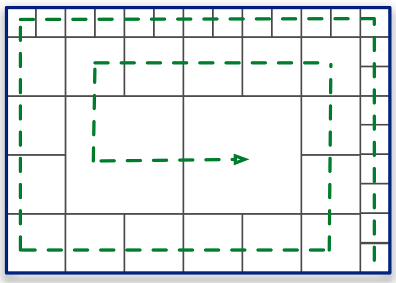


Figure 10: Query response in fractional cascading. The query is to find an aggregate (e.g. sum of values) of all sensors in the outer rectangle R shown in bold. Each square in the figure is canonical piece – its parent square is not in R. The method needs to visit each canonical square once, this is done by following the spiral path shown as dashed segments, that has length $O(P \log P)$.

squares inwards. The cost of visiting all the canonical pieces will be simply $O(D + P \log P)$, taking into account the distance of R from the query point. Similarly, it is possible to compute sums, where each node stores the sum of values in its square and their siblings at all levels. This is useful in answering a question of type "How many animals are there in region R?" and can be answered at the same cost.

Fractional cascading is a fundamental concept in computer algorithms [8], and have been used in different fields in different ways. It will make further appearances in the next section when we discuss tracking mobile devices.

2.2.3 DIM: Locality Preserving Storage of Multi-dimensional Data

Sensors in a network are likely to have many different sensing capabilities. And queries may be with respect to multiple parameters. Whether it is a data center or a wildlife preserve, we need to keep track of many different parameters that will help us understand animal behaviors, hardware failures and other events in the network.

For such data, it is useful to be able to make range queries: which hardware failures in the data center typically happen at high temperature and load conditions? At which locations has the bird been spotted when temperature was between $30°C - 35°C$ and humidity in range $80\% - 100\%$?

To answer the queries efficiently, it helps if similar data are stored close together. For example, if events at similar humidity and similar temperature are stored nearby, the cost of answering the queries above will be low.

Based on this idea, DIM [28] suggests a locality preserving hashing scheme for events. Here "locality preserving" means that it tries to place together events that are similar in some parameters.

Let us suppose for the moment that we have k different binary parameters. We can ask: "Was the temperature low or high?", "Was the humidity above or below 50%?" and similar questions about every parameter. Thus each event is accompanied by a bit vector b of length k. To each possible bit vector, we will assign a zone – a region of the network – where the corresponding events will be stored.

This method is inspired by a different type of space partition called *kd–trees* [8]. We start with our square network region R, and partition it recursively, diving each region into two according to the next bit $b[i]$ of b. If index i is even, we split R with a vertical line, and depending on $b[i]$ being 0 or 1, we choose left or right. Similarly, if i is odd, we split R with a horizontal line and depending on if $b[i]$ being 0 or 1, we choose bottom or top. Figure 11 shows an example how we can map any bit vector to a unique region of the network.

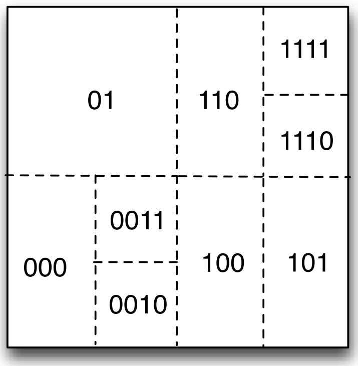


Figure 11: Any bit vector can be assigned to a unique region in the network.

Thus, given an event whose binary properties are represented by b, we have a way to map it to a region, and events with same properties will be mapped to and stored in the same region. Observe that parameters whose values are earlier in the sequence have greater weight in determining the neighborhood of storage, creating an imbalance in the significance of different parameters.

The case where parameter values are not binary can be handled by considering the binary representation of the values. For simplicity, let us say each value is an integer in the range [0 to $2^v - 1$], represented by v bits. The first bit is the most significant – it determines if the value is in the range [0 to $2^{v-1} - 1$] or [2^{v-1} to 2^v]. Given that the first bit determines if the value is in the left or right half of the range, the second bit determines if the value is in the lower or upper half of the reduced range, and so on. To map k such values to the network, we utilize this bit representation concept. The first k bits of b are the most significant bits if the k values, the next k bits store the second most significant bits of the values and so on.

3 Mobility Management and Tracking

Let us revisit a question we had considered in the previous section. A tourist asks "Where can I find an elephant?" We discussed some methods of brokerage that helps the tourist to find the animals of interest. These brokerage methods work well as long as the animals stay in their place, or move very rarely. What happens when the animals are active and move continuously? In such cases, methods such as GHT and Double Rulings have to continuously update the storage data – by sending messages on long paths or curves.

This is identical to a common question in mobile networks. "Where is user x?" which is important when placing a call to user x in cellular networks. Cellular networks handle mobility by assigning to each phone a "home" server and having the phone update this server suitably. In a sensor network the corresponding strategy will be updating a hash location in GHT or sending a message along a double ruling path every time x moves, which is impractical for frequently moving targets.

The general problem of tracking and finding mobile objects is therefore a challenging topic in sensor networks. It is particularly important and difficult in the case when the tracked object is a frequently moving device such as phone, or a GPS in a fast moving car. The question of detecting a nearby target and detecting its movements and location are themselves subjects of extensive research. However, our topic of discussion in this chapter is sensor data, we will therefore focus on managing the tracking information obtained by the sensors. For simplicity, we can assume for example, that the mobile devices are GPS enabled and are willing to cooperate by communicating their true locations.

3.1 Hierarchic Tracking Data

Hierarchic data in quadtree format is relevant to tracking mobile objects as it is to tracking isolated data. The methods using hierarchic information fundamentally use the fractional cascading concept of storing more detailed information about the local neighborhood, and lower resolution data about regions farther away.

GLS: Grid location service. This method was originally described for mobile ad-hoc networks in [27], but is based on the same essential ideas that we are using in sensor networks.

GLS assumes a global total ordering of n node ids in a cyclic directed list: $L = 0, 1, 2, \ldots, n-1 (\text{mod } n)$. Suppose the node with id x belongs to a square s_i^x at level i. GLS stores x's location at the node whose id is the first after x in the sequence L among nodes in s_i^x. Then it repeats the procedure for the siblings of s_i^x and similarly stores the id of x at the successor of x in each of these squares.

You may have already noticed the similarity to fractional cascading that we saw in section 2.2.2. The information in a square s_i at level i is replicated in each of its siblings. The difference is that in section 2.2.2 we dealt with only the aggregate value, and every node in a s_i stored the same level i value. In GLS, there is no aggregation. The information about node x is stored at exactly one node in square s_i^x, and at one node in each sibling square.

When node y searches for x, it sends a search message to the node first in L after x for which y has location information. This node performs the same operation again, looking for x. It can be shown that this is guaranteed to reach a location server of x, which will be able to forward the message directly to x. The initial registration of x's location and updates on moving can be executed using the same basic operation. When x wants to select and update a location server in a square s, it sends an update message to s. The message starts as a search for x inside s and will find the node that should be x's location server. Thus, using the same elegant primitive, GLS handles both the fundamental operations of searches and updates.

The difficulty in GLS is that the search cost can be disproportionately large compared to the distance between x and y. When these two nodes are close but lie in different squares of the quadtree partitioning, the search may have to take a long path. The same problem can arise when x moves. A small move of x can produce a costly update. We will discuss next a method that solves this problem.

LLS : The locality aware location service. This hierarchic method [1] uses location servers at different levels of the quadtree. For a mobile node x, there is a hash location $h(x)$ at the root level of the quadtree that stores its data. Similarly, there is a hash location $h_s(x)$ for any square s at any other level of the partitioning. These locations act as location servers for x. Any other node that looks for x can get the information by communicating with a few of these servers.

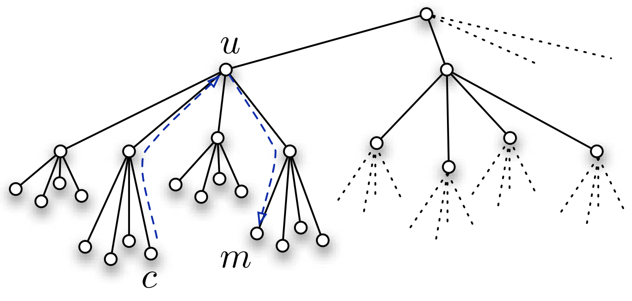
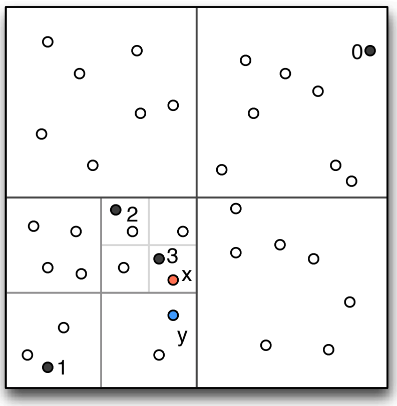


Figure 12: Basic search in hierarchic mobility tracking. Only relevant part of quadtree is drawn. Suppose m is the lowest level square (leaf node in the quadtree) that contains the mobile user. The quadtree nodes on the path from m to the root have information about the mobile user. The consumer at node c searches ancestors of c until it hits p – the common ancestor with m. The search then proceeds down the tree to find m and the precise location of the user.

At any time, if x is in square s_i^x at level i, then the location server at $h_{s_i^x}(x)$ stores location of x. In fact, it only notes a pointer to the square at level $i+1$ that contains x. There are $O(\log n)$ different location servers – one at each level – that carry information about x's location at any time. This is different from structured replication in GHT. In structured replication, there is a similar hierarchy of hash locations, but the data is stored at only one of them, whereas in LLS, information is stored at one server for each level. And in contrast to GLS, the server $h_{s_i^x}(x)$ at level i only records the level $i+1$ square s_{i+1}^x containing x, but not its exact address. This has the advantage that when x moves anywhere inside that square s_i^X, the server $h_{s_i^x}(x)$ does not need to be updated.

When a different node y wants to find the location of x, or to communicate with it, an inductive search is performed. The lowest level square s_j^y containing y is checked first. To be precise, the location server at $h_{s_j^y}(x)$ is asked for x's location. If this fails, the higher level server in s_{j-1}^y is checked and so on, until a server with information about x is found at some level – which may be the top level $h(x)$ in the worst case. Next, the query has to follow pointers down the levels to the leaves. An example is shown in Figure 12.



(a)

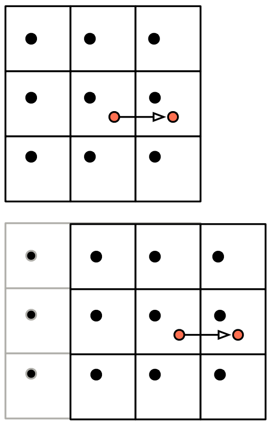

(b)

Figure 13: (a) The black shaded nodes are x's location servers. The numbers next to them are the level of the quadtree that they correspond to. y searches for x, and finds a trace of x at level 1. (b) At any level, x's location is stored at eight adjacent squares in addition to the host, When x moves to one of these squares, no updates take place. However, on the next move an update is triggered and all twelve squares are updated.

This basic search idea has a disadvantage that in certain cases, the search cost may be much higher than the distance between x and y – this search is not distance sensitive. See for example the case in Figure 13: x and y are geographically close, but y has to communicate with the distant location server "1" to find x. This happens because the two nodes are separated at a high level of the partitioning, although they are quite close in location. The same issue can cause the update cost of x to be high when x crosses this boundary.

To make the search distance sensitive, LLS replicates x's location data at the eight level i squares neighboring s_i^x. This replication guarantees that the search incurs a communication cost $O(d)$ where the distance between x and y is d. It also makes it possible to keep the update costs distance sensitive by making updates in a lazy manner. When x moves to one of the neighboring square, no updates are performed. When x moves out of this neighborhood, an update is performed to remove the outdated information and to enter the new information in the new neighborhood. Therefore x drags with it a sliding window of servers at each level – see Figure 13(b). The idea 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 means, the average cost over many moves is guaranteed to be low, but the update cost at a particular step can be arbitrarily high compared to the movement at that step.

During a search, 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 increase the asymptotic search cost.

In the next section we will consider a different problem of aggregate queries: answering a question of type: "How many device are in an arbitrary region R of the network?" LLS, based on its information of node locations can answer this question the same way that range queries are answered in the fractional cascading method of section 2.2.2 – using the spiral traversal of canonical pieces shown in figure 10. While this works, the query process is not efficient since the preprocessing carried out by LLS was not designed for such questions. The algorithm in [29] is more efficient in answering such queries. There the preprocessing is based on computing a harmonic function that makes it easy to answer such aggregate questions. However, the computation of a harmonic function and its subsequent updates can be expensive; we will instead look at a different method for answering counting queries in the next subsection.

In summary, the utility of LLS and GLS is in search and communication with individual mobile nodes. The hierarchic partitioning scheme provides the benefit that the search process operates in a small neighborhood of the query origin, and goes to broader regions only when that is necessary. The updates are performed lazily – delayed as much as possible to minimize costs. Thus the location servers provide an essential prerequisite of point to point communication: they find a particular user based on the device identifier.

3.2 Differential tracking forms: Aggregate tracking

Beyond the question of tracking movements of individual users, we can ask questions about aggregates of users. This is analogous to the range queries we had considered in the previous section. Now we wish to answer questions of type: "How many users are there in a region R?" This is useful when we wish to quickly find the traffic density in arbitrary regions of the network, or the number of people in the neighborhood of a sporting venue.

It is possible to answer the question using the location server hierarchy of LLS. We can use the spiral traversal method shown in figure 10, and answer the question the same way. However, there are a few aspects of LLS that we wish to improve upon:

1. **Search Costs:** While the canonical traversal is nice, we do not know where the different location servers in a canonical square may be. We would need to search all nodes in each square, and therefore all nodes in R to find the answer.

2. **Update Costs:** The costs of updating location servers may be high, even when a node moves a small distance. This is particularly significant when updates are very frequent, for example when tracking moving vehicles – with high speeds and large numbers.

3. **Privacy:** LLS requires tracking each device at every moment. The users may wish not to be tracked with such precision. We would like to keep the counting information, without following the precise movement of individual devices.

For this problem, we find it beneficial to leave the hierarchic fractionally cascaded data format and return to a flat data model once more. The model is based on the concept of a *differential form* in mathematics. This fundamental concept can be adapted to sensor networks by interpreting it as weights on edges of a planar graph [38]. We will address a more general problem of counting the total *weight* of targets in an arbitrary region. Counting targets is a special case of this question with each target having weight 1.

We first compute a planar graph in the network the way we discussed for routing(section 1.2). This graph needs to be one such that when a target crosses an edge of this graph, one or more sensor detects this fact – for example, by an explicit update from the target itself, or by a localization carried pout by the sensors. We assign a weight to each edge. This weight function ξ is a special one; it is constructed to have two important local properties:

1. The weight is directed: $\xi(ab) = -\xi(ba)$. That is, if a message moving from a to b sees a weight w, then a message moving from b to a sees a weight $-w$.

2. If the weight of targets in face f is w, then a message traveling along the boundary edges of f in a clockwise direction sees a total edge weight of w. A message traveling along the boundary of a face without targets sees a total weight zero.

It turns out that having these two simple properties allow very sophisticated tracking: we can find the weight of targets inside any region R just from the weights of edges on the boundary of R. For a face f, let us refer to the boundary of f traversed clockwise as ∂f. We can treat the region R to be a collection of faces, and denote its boundary by ∂R.

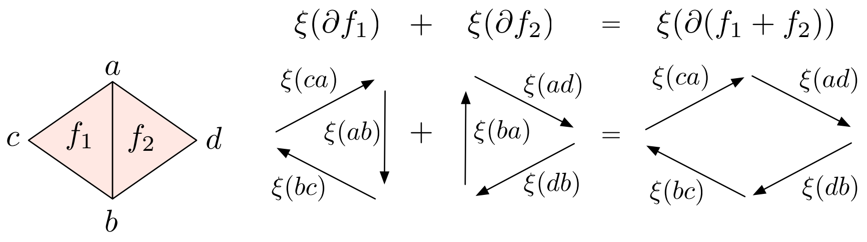


Figure 14: The weight inside a collection of faces $f_1, f_2, \ldots$ can be obtained by simply adding the weights on the boundary of the collection. For every edge ab in the interior, weights $\xi(ab)$ and $\xi(ba)$ cancel, leaving only the outer perimeter edges.

To obtain the total weight on faces inside R, it is sufficient to add the weights of edges on ∂R. Thus, we find the total weight inside R simply by making a clockwise tour of its boundary, without ever entering R at all! Figure 14 shows the intuition behind this idea. For a formal proof, see [38].

When a target moves, we need to update these weights. Suppose a target of weight w moves from face f_1 to face f_2 above, crossing the edge ab, the function ξ is updated by $\xi(ab) \leftarrow \xi(ab) - w$ (or equivalently, $\xi(ba) \leftarrow \xi(ba) + w$). It is easy to check that this update reduces the weight on the boundary of f_1 by w, and increases the weight around f_2 by the same amount. The function ξ is called a *differential tracking form.*

Initializing the function ξ requires setting its value on the edges suitably. To do this, we can assume that the target arrived at its current position through some arbitrary path from the exterior, and update the edges it would have crossed on such a path (see Figure 15). This is done in a more consistent and efficient way by constructing a dual of the planar graph and using a spanning tree of this dual to find paths for all targets. This method works at $O(n)$ communication complexity.

The differential tracking form has several nice properties. It is tolerant to coverage holes: even if a target enters a hole and is currently not in the range of any sensor, its information is stored at the boundary of the hole, and for any region that contains the hole, this method still returns the correct answer. For the same reasons, it is also tolerant to sensor failures. The method can be made locally adaptive to insertion of new sensors and movements of the sensors themselves. The tracking is also anonymous – we update someone crossing an edge, but not the identity of the user or device. In fact, the entire protocol works without any need for identification.

The most important property, from a performance point of view, is that updates are very inexpensive. The weights on the edges can be maintained

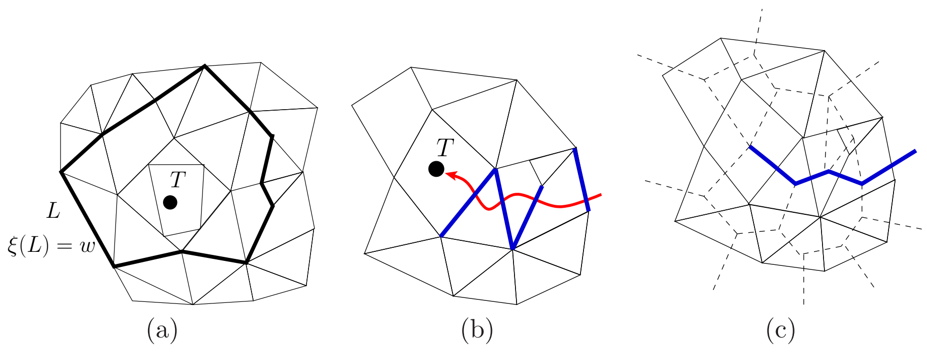


Figure 15: (a) Suppose the single target T has weight w. Then for a loop L that contains T, $\xi(L) = w$. Such a loop is shown in bold. (b) A target T enters the network along the red curve updates a sequence of edges shown in bold blue. If the T is already at the position, we can imagine that T entered through some such path. (c) The trail corresponds to a path in the dual graph. The edges to be updated are precisely the duals of the edges on this path.

locally by the sensors that are the end points of the edge. When a target crosses the edge, this is detected locally, and this weight is modified. That is all that is needed. When a target moves a distance d, the update cost is $O(d)$ – the number of edges crossed by the target. Efficient updates are critical to tracking mobile objects, since movement of devices or vehicles are generally much more frequent that queries. The completely local update has the additional advantage that it does not require participation of sensors far from the region of activity, which can happen, for example, in LLS. It is therefore possible for sensors to stay in sleep mode while there is no activity in the neighborhood.

In general, it is useful to maintain different tracking forms for different categories of objects which we may want to search or count. This is particularly useful for example for the tourist looking for elephant type of query we discussed earlier, or for a traveller searching for a cab. It is shown in [38] that by repeated application of the counting query, such questions can be answered at a distance sensitive cost of $O(d)$.

Other than moving targets, tracking forms are also useful in aggregating general signals monitored by sensors. We simply need to treat the signal value at a sensor as the target weight at that location. For example, suppose we wish to find the average temperature. We need two tracking forms: one for temperature, one for node count. By finding the sum of temperatures and dividing by the number of sensors in any region, we easily find the average.

On the whole, this method is very robust and flexible. The ability to compute sums of values can potentially be extended to compute other types of functions of sensor values. Exactly how to achieve such extensions and exactly what can be achieved with this method remains to be investigated.

4 Networks with Complex Shape: Segmentation and Virtual Coordinates

We had started off assuming that our sensors are in a nice square field, and we applied plane geometry without regard for obstacles. Reality may not be so accommodating. Obstacles like buildings may create *holes* in the sensor network or the perimeter may be irregular instead of a square. What happens to the hierarchic and flat structure algorithms when they encounter the boundary of such a gap in the network?

Empty regions in a sensor network are often governed by the nearest sensor. Recall what DIM [28] does: it stores data at zones determined by its attributes $(attr_1, attr_2, \ldots)$. In irregular networks the data aimed at zones in large empty regions are recorded by sensors at their boundary, and are also retrieved at the boundary. While this works, it is not the most efficient. See for example

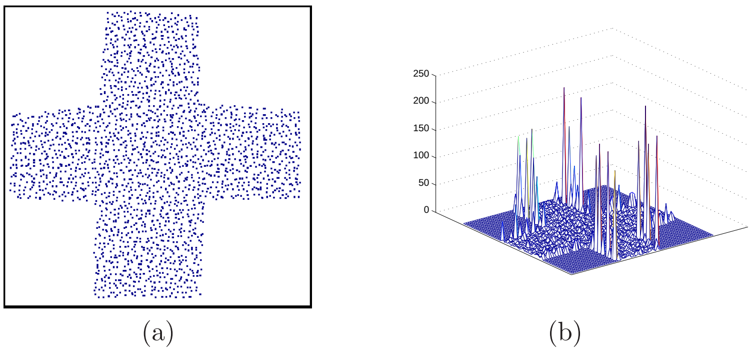


Figure 16: (a) A network whose boundary does not match the square. (b) Load distribution of DIM for a set of random data. The boundaries are overloaded while interior nodes have much lower workload. The variation in load between different boundary points comes from the design of the algorithm.

Figure 16. The network in 16(a) has an irregular shape, and we apply DIM to the bounding box shown as the black square. All data hashed to the empty white regions are eventually stored at nodes nearest to these regions – at the network boundaries. These boundary nodes store a higher fraction of data than others as shown in 16(b).

The problem is not specific to DIM. GHT and all the hierarchic data handling methods rely on "nearby" sensors in some way or the other, and therefore show similar imbalance against boundary nodes. Double ruling and other path based methods are also not immune. A path that encounters a hole will typically move along the boundary to find a way to the destination. Routing methods like GPSR frequently move along hole boundaries in perimeter mode, and create heavy loads there. Without any special care, workload distribution will generally be heavily against boundary nodes, resulting in hotspots, delays, low efficiency, and possibly loss of packets.

The difficulties appear largely from our misplaced assumption of a simple square network. If we take the precise network shape and adjust our algorithms not to stray outside, they may operate in a balanced way. However, there are several practical and theoretical hurdles to this ideal approach. Efficiently describing a convoluted boundary and storing it at each sensor is difficult in general, and impossible for a sufficiently complicated boundary. Even if we can somehow figure out the boundaries and their descriptions, it is not clear how to adjust our algorithms. Hashes and space partitions that are natural in a square, are hard to adapt to arbitrary shapes. Instead we will discuss two other methods of handling complex shapes. The first is to decompose a network into simpler pieces and apply the algorithms independently in each piece. The second is to create virtual coordinates with simple shapes thus eliminating the problems of complicated boundaries.

4.1 Network decomposition

The network of Figure 16(a) has a natural structure made of five approximately square shapes. Figure 17(a) shows a decomposition of this type. Nodes belonging to the same segment after decomposition are shown in the same color. The

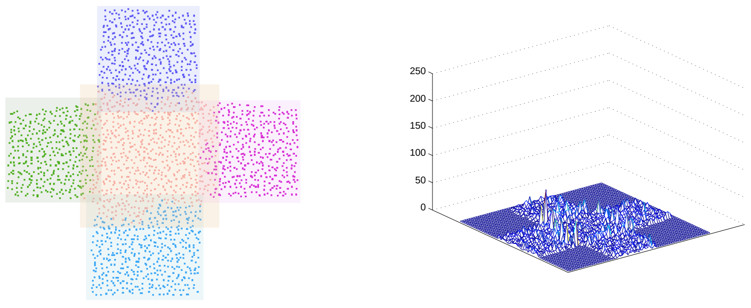


Figure 17: (a) The network decomposed into segments forming simple shapes. Each segment is shown in a different color, along with its own bounding box. (b) Load distribution when DIM is applied to segment-wise bounding boxes instead of the global range. Bounding boxes may have partial overlaps, and those nodes have to operate for both the squares and therefore take double the load as seen above. But load balance is still better than Figure 16.

segmented network can now be covered by five bounding boxes – one for each segment. Instead of dividing event ranges into two parts and allocating to two parts of the network, we can now divide them into 5 parts of suitable sizes, and allocate to different segments proportional to sizes. In each segment, we follow the usual DIM method of binary space partitions to allocate events to zones in the bounding box. As a result, the allocation of zones is tight with respect to the actual sensor distribution and the imbalance at boundaries disappears – see

Figure 17(b). Other data storage schemes will have similar improvements.

The network segmentation idea and the results above are from [46, 47]. The decomposition was obtained by taking the distance of nodes from the boundaries, and constructing a discrete representation of the gradient vector field of this distance. The different segments correspond to partitioning the vector field according to its different *sinks* or end points, and computed distributedly in-network. This method simply tries to divide the network into its constituent large pieces, and not specifically into squares. Smaller segments generally have tighter bounding boxes, and as a result better balanced performance.

Managing the network in suitable partitions has been considered in different scenarios. Decomposing the network in convex regions can help in routing, since simple greedy routing works well in convex shapes. This has led to the effort of decomposing the network into Greedily Routable Regions [21]. Other convex decompositions have helped in localization methods [30].

As we saw above, once the network is segmented, the data storage algorithm takes into account this split at the top level of operation and divides data into the segments. This method needs the storage algorithm to be aware of the segmentation and may be impractical and inefficient when many segments are connected in complicated ways. Our next method tries to avoid these issues by creating virtual coordinates.

4.2 Virtual coordinates

Virtual coordinates are a simple concept. Instead of using the real locations, each sensor is assigned a different logical coordinate to simplify data processing. In our case, we want coordinates that give a simple virtual shape to a network in place of the original complex one – hopefully one that balances the load for our storage algorithms.

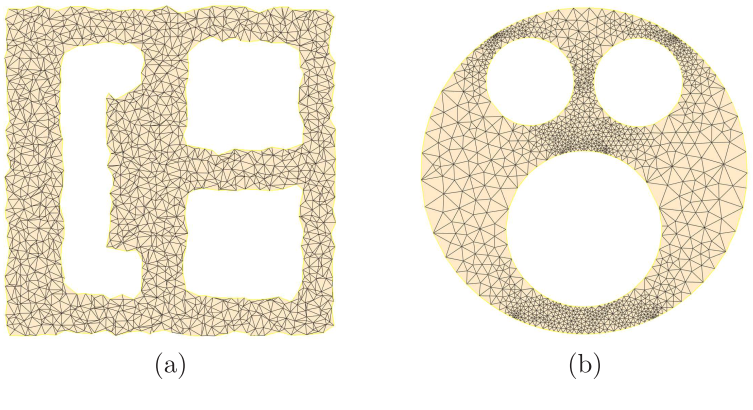

(a) (b)

Figure 18: (a) Triangulated network. (b) Virtual coordinates with circular holes. The shape of the network is transformed by *Ricci Flow* to obtain coordinates with circular holes.

Figure 18(b) shows a type of virtual coordinates where each hole boundary of 18(a) is converted into a circle, and the outer boundary of the network becomes the outer circular boundary. This configuration is easier to describe than the original one. A circle is represented by its center and radius, thus the entire network is now described by just three numbers per boundary.

The transformation of Figure 18 is obtained by *conformal deformations* of the original network. A conformal deformation is equivalent to local scaling – either contraction or expansion – at each point in the network. *Ricci Flow* is a particular technique of repeated applications of such deformations that achieves the transformation of arbitrary boundaries to circle. Application of this method requires a triangulated planar graph. Thus the Ricci Flow algorithm described in [39] starts with a distributed method of triangulating networks. This triangulated state is visible in Figure 18 – each face of the graph, except the holes, is a triangle.

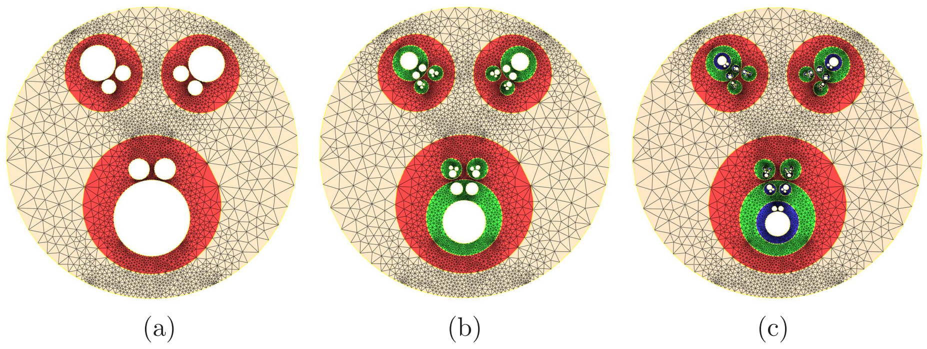


Figure 19: The virtual coordinates can be reflected in a circular boundary. (a) At each reflection, a boundary goes to a circle. (b & c) The reflections can be repeated in the new circular boundaries until holes are negligibly small.

The network still has large holes and we need a method to solve the load imbalance of storage schemes. This is done by a technique called *circular reflections*. Similar to reflecting figures about a line in the plane, it is possible to perform reflections about a circle – points outside the circle are sent inside and vice versa. For each hole boundary, we perform such a reflection that creates a copy of the network inside it, see Figure 19(a). There are two nice effects of this reflection. First, it fills up much of the holes, reducing empty space, and second, the circular boundaries map to circular hole boundaries inside. This second property means that we can repeat the entire process in these new circular boundaries and reduce empty space indefinitely: Figure 19(b & c).

Finally, we can now have a network with only small and insignificant amount of empty spaces inside. For any point that was in a large empty space previously, now it is possible to find a reflected image of a node close to it. A message intended for this point goes to this node instead of the boundary, thus giving us a more balanced storage. Routing to a point inside a "hole" is simple –

reflection preserves continuity at the boundary, and thus it is possible to reach the interior point by simple greedy routing. Either the storage or the routing do not require us to compute the reflected images of the nodes beforehand – they can be decided on the fly as needed. And we also do not need to consider an infinite number of reflections, a few levels of reflections suffice in most cases. See the discussions in [40] for more details of this method.

Several other methods of computing virtual coordinates exist, with different properties and applications. The reader is may find interesting ideas in hyperbolic virtual coordinates [24, 45]. However, these and most other virtual coordinates [11, 13, 32, 36] are designed for routing, and not for data storage.

5 Discussion

We hope that the reader got an impression of the variety of techniques and ideas that can be brought into play with locations as the primary index. Beyond the natural localization of sensor data by coordinates, many interesting queries are location oriented. Whether we look for the nearest cab or ask for the average temperature in a part of the city; we are interested not in the whole of network data, but only in a geometric local part of it, and our methods should be designed accordingly.

The impact of locations and geometry is almost always useful when looking for simple yet efficient methods to handle sensor data. See for example [9], which uses a gossip algorithm on sensor networks to compute averages of sensor values. Instead of exchanging information with neighbors as in a traditional gossip algorithm, here the nodes perform gossip with random locations in the network. As a result, the algorithm converges to the desired result, such as the average temperature, much faster in this method. Fractionally cascaded hierarchic information can also be computed, using a different gossip technique called spatial gossip [42, 44]. This protocol computes and stores $\log n$ aggregates at each node. For any level i of the hierarchy, a node p receives the aggregate of all nodes within a distance 2^i of p. This is analogous to the quadtree based methods we saw earlier, except that unlike a quadtree, here the average is of a disk centered exactly at each node p. This technique in fact relies on an idea from social networks: a model for creating small world graphs [23, 20].

The concepts in processing information in sensor networks are more general than the applications to sensor networks themselves. With low power nodes and the need to process large quantities of data at minimal communication, sensor networks are a more restrictive and challenging platform than most others. A method that is suitable for sensor networks is also likely to be efficient in other networks with suitable adaptation. The approach of treating all nodes as minimal and equivalent aids this generality. Since we do not assume specific network configurations or special abilities on part of some nodes, these methods are likely to be easily adaptable. Such flexibility is important in the ever changing world of modern wireless and sensor networks.

Here we mentioned only a few works in this domain, and that too only superficially. Our goal was to introduce some elementary yet important ideas in the topic. The reader is encouraged to look into the original articles and the many other works for more details, subtleties and elegant ideas in this fast developing area.

As sensor networks and similar systems become common in the real world, our views on them will surely change. We will learn how they are deployed and used, and face new challenges. We will need to adapt our existing algorithms, and develop new models, applications and algorithms to adjust to the new networks.

With the popularity of location enabled portable devices, the tracking, storing and managing of location data are becoming more important. Since much of our data, including photos, notes and messages are now location tagged, we can develop new types of applications taking advantage of these features. At the same time, we can rethink our existing ideas and protocols for a location-aware world.

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. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[3] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *Proc. of the 1st ACM Int'l Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 22–31, September 2002.

[4] J. Bruck, J. Gao, and A. Jiang. Localization and routing in sensor networks by local angle information. In *Proc. of the Sixth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*, pages 181–192, May 2005.

[5] J. Bruck, J. Gao, and A. Jiang. MAP: Medial axis based geometric routing in sensor networks. *Wireless Networks*, 13(6):835–853, 2007.

[6] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: Network routing inspired by DHTs. In *SIGCOMM'06*, 2006.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Mass., 2001.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[9] A. G. Dimakis, A. D. Sarwate, and M. J. Wainwright. Geographic gossip: efficient aggregation for sensor networks. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 69–76, 2006.

[10] T. Eren, D. Goldenberg, W. Whitley, Y. Yang, S. Morse, B. Anderson, and P. Belhumeur. Rigidity, computation, and randomization of network localization. In *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, volume 4, pages 2673–2684, March 2004.

[11] Q. Fang, J. Gao, L. Guibas, V. de Silva, and L. Zhang. GLIDER: Gradient landmark-based distributed routing for sensor networks. In *Proc. of the 24th Conference of the IEEE Communication Society (INFOCOM)*, volume 1, pages 339–350, March 2005.

[12] Q. Fang, J. Gao, and L. J. Guibas. Landmark-based information storage and retrieval in sensor networks. In *The 25th Conference of the IEEE Communication Society (INFOCOM'06)*, volume 1, pages 339–350, April 2006.

[13] R. Fonesca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 329–342, May 2005.

[14] 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.

[15] J. Gao and L. J. Guibas. Geometric algorithms for sensor networks. *Philosophical Transactions of the Royal Society A*, 2011.

[16] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *ACM Conf. on Mobile Computing and Networking (MobiCom)*, pages 56–67, 2000.

[18] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[19] 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.

[20] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 163–172, New York, NY, USA, 2001. ACM Press.

[21] A.-M. Kermarrec and G. Tan. Greedy geographic routing in large-scale sensor networks: a minimum network decomposition approach. In *Proceedings of the eleventh ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '10, pages 161–170, New York, NY, USA, 2010. ACM.

[22] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker. Geographic routing made practical. In *Proceedings of the Second USENIX/ACM Symposium on Networked System Design and Implementation (NSDI 2005)*, May 2005.

[23] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170, 2000.

[24] R. Kleinberg. Geographic routing using hyperbolic space. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1902–1909, 2007.

[25] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: Of theory and practice. In *Proc. 22^{nd} ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, pages 63–72, 2003.

[26] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric ad-hoc routing: of theory and practice. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 63–72, 2003.

[27] J. Li, J. Jannotti, D. S. J. De Couto, D. R. Karger, and R. Morris. A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, 2000.

[28] 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.

[29] 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.

[30] W. Liu, D. Wang, H. Jiang, W. Liu, and C. Wang. Approximate convex decomposition based localization in wireless sensor networks. In *INFOCOM*, pages 1853–1861, 2012.

[31] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[32] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, DIALM-POMC '04, pages 8–16, New York, NY, USA, 2004. ACM.

[33] B. Nath and D. Niculescu. Routing on a curve. *SIGCOMM Comput. Commun. Rev.*, 33(1):155–160, 2003.

[34] J. Newsome and D. Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 76–88, 2003.

[35] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[36] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, 2003.

[37] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *Proc. 1st ACM Workshop on Wireless Sensor Networks ands Applications*, pages 78–87, 2002.

[38] R. Sarkar and J. Gao. Differential forms for target tracking and aggregate queries in distributed networks. In *proceedings of The 16th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, 2010.

[39] 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)*, April 2009.

[40] R. Sarkar, W. Zeng, J. Gao, and X. D. Gu. Covering space for in-network sensor data storage. In *Proc. of the 9th International Symposium on Information Processing in Sensor Networks (IPSN'10)*, April 2010.

[41] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 286–297, September 2006.

[42] R. Sarkar, X. Zhu, and J. Gao. Hierarchical spatial gossip for multiresolution representations in sensor networks. In *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'07)*, pages 420–429, April 2007.

[43] R. Sarkar, X. Zhu, and J. Gao. Double rulings for information brokerage in sensor networks. *IEEE/ACM Trans. Netw.*, 17(6):1902–1915, Dec. 2009.

[44] R. Sarkar, X. Zhu, and J. Gao. Hierarchical spatial gossip for multiresolution representations in sensor networks. *ACM Trans. Sen. Netw.*, 8(1):4:1–4:24, Aug. 2011.

[45] W. Zeng, R. Sarkar, F. Luo, X. D. Gu, and J. Gao. Resilient routing for sensor networks using hyperbolic embedding of universal covering space. In *Proc. of the 29th Annual IEEE Conference on Computer Communications (INFOCOM'10)*, April 2010.

[46] X. Zhu, R. Sarkar, and J. Gao. Shape segmentation and applications in sensor networks. In *Proceedings of the 26th Conference of the IEEE Communications Society (INFOCOM'07)*, pages 1838–1846, May 2007.

[47] X. Zhu, R. Sarkar, and J. Gao. Segmenting a sensor field: Algorithms and applications in network design. *ACM Trans. Sen. Netw.*, 5(2):12:1–12:32, Apr. 2009.