Proceedings of 2003 IEEE Wireless Communications
and Networking Conference (WCNC)

Efficient Location Tracking Using Sensor Networks

H. T. Kung and D. Vlah
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138, U.S.A.

***Abstract* -- We apply sensor networks to the problem of tracking moving objects. We describe a publish-and-subscribe tracking method, called Scalable Tracking Using Networked Sensors (STUN), that scales well to large numbers of sensors and moving objects by using hierarchy. We also describe a method, called drain-and-balance (DAB), for building efficient tracking hierarchies, computed from expected characteristics of the objects' movement patterns. DAB is shown to perform well by running it on 1D and 2D sensor network topologies, and comparing it to schemes which do not utilize movement information.**

I. Introduction

Progress in miniaturization has allowed researchers to build *networked sensors*, increasingly compact devices that combine the functionality of sensors, radios, and processors [1, 5, 8]. Their low cost and wireless communication capability make it feasible to deploy them in large numbers, and without pre-existing infrastructure. With more sensors available in the environment, it is more likely that phenomena of interest are near some sensors, thereby leading to the main appeal of wireless sensors compared to the tethered ones. On the other hand, such sensors rely on limited sources of power, and so energy efficiency becomes an important feature of the systems that use them.

Tracking consists of detecting and monitoring locations of real-world objects, possibly using several types of sensing such as acoustic, seismic, electromagnetic, etc. Numerous applications of tracking are currently in use; for example, air traffic control, fleet tracking, habitat monitoring, mobile telephony, etc. Networked sensors have recently been used for this purpose [2].

We are interested in designing an efficient tracking method, based on networked sensors, that has the ability to cover large regions of interest by using many sensors with small detection range. The method will need to handle a large number of moving objects at once. To address the scalability goals, we use hierarchical organization, similar to schemes previously used in cellular telephone systems [6]. We focus on improving the energy efficiency of such schemes.

A. Movement Locality

In order to aid the task of designing methods for tracking objects across large areas, we discuss the expected characteristics of large-scale traffic patterns. A popular mobility model used for several ad hoc network protocols [4, 7] consists of nodes that move uniformly across a region under study, in order to approximate small-scale scenarios where the movements are expected to be random and close to uniform. Examples of such scenarios are visitors at a convention, rescue teams at a disaster site, etc.

We postulate that large-scale movement patterns are not likely to be uniform, because large-scale real-world environments usually have inherent structure that makes this infeasible. For example, a downtown area of a city may consists of a street grid and buildings that prevent pedestrians from moving around arbitrarily. Further, it is uncommon for a city-dweller to roam around uniformly at random; a more likely trajectory consists of spending time within some local region, and occasionally making long-range transitions.

As a second example, consider a large-scale disaster recovery or battlefield scenario. It may be unusual for every participant to move uniformly across the entire field; rather, due to organizational structure, the movement may be constrained so that there is little traffic between different units, while movement within units is quite high.

Finally, let us examine a large natural habitat, such as a national park, that scientists may wish to monitor [2]. Once more, it is unlikely to observe animals moving around uniformly. The animals may be living in herds; different types of animals may avoid trespassing upon each others' territory; or there may be various natural obstacles like hills or undergrowth that restrict movement in certain regions, etc.

The common characteristic of the above three examples is movement locality. This observation provides support for our design in later sections.

B. Summary of Contributions

This paper makes two contributions. The first is STUN—Scalable Tracking Using Networked Sensors—a scalable tracking architecture that employs hierarchical structure to allow the system to handle a large number of tracked objects. The design of this architecture is the topic of Section II.

Our second contribution is DAB (drain-and-balance)—a method to construct STUN's hierarchical structure based on expected properties of the object movement patterns such as the frequency of object movements over a region. Utilizing these properties permits it to build hierarchies which are more efficient than those constructed without. The DAB method is described in Section III. We demonstrate the benefits of DAB in Section IV by applying it to 1D and 2D sensor network topologies.

II. STUN: Scalable Tracking Using Networked Sensors

Consider a set of objects moving through a certain region of interest, where a number of short-range wireless sensors has been deployed for detection purposes. The *distributed tracking* problem we are interested in solving is about communicating the locations of the detected objects from the sensors to a *querying point*, where the information is further utilized by the user.

A. The STUN Design

The approach we use, called STUN, rests on the observation that the objects we wish to track move predictably, in a sense that there exists a natural limit on their speed, and thus the maximum distance they can travel in a given amount of time is bounded. The approach uses *hierarchy* to record information about *presence* of the objects; since their movements are limited, keeping this information up to date usually requires updating just the nodes near the bottom of the hierarchy. As a result, the system can track more objects than one where all information travels indiscriminately to the querying point.

Our method uses a hierarchy to connect the sensors, using the querying point as the root. Let us call this the *hierarchy tree*. The leaves of the tree are sensors, and the other nodes are communication nodes, which are called intermediate nodes in the rest of the paper. The information about presence of the detected objects is stored at intermediate nodes; in particular, each intermediate node stores the set of objects that were detected jointly by its descendants. Let us call this set the *detected set*. For example, the detected set of a sensor at a leaf node consists of just the objects within the sensor's detection range, while the root node's detected set contains all objects present in the region.

In order to keep the detected sets up to date, leaf sensors send detection messages toward the root. However, the messages do not always need to reach the root; an intermediate node will pass a detection message upward only if it has modified its detected set. Otherwise, the message is terminated, because it would not have modified any ancestor nodes. By eliminating redundant message passing, this message-pruning hierarchy is the key to lower communication cost.

We provide an example in Figure 1 where a row of sensors covering a region is shown to be descendent from the same intermediate node X. When an object, which is a car, moves through the region, each sensor will detect its arrival and departure, and generate the corresponding detection messages. To simplify the illustration, we consider here only the messages that detect the arrival of the car. Note that the message from sensor 1 triggers Y and its ancestors to add the car to their detected sets. Subsequent messages from all the other sensors do not result in additions to X's detected set, and thus X does not forward them to its ancestors. Similarly, the message from sensor 3 adds the car to the detected set of Z, is forwarded to X, but then stops at X as it leaves X's detected set unchanged. Messages from sensors 2 and 4 do not change the detected sets of their parents, and thus do not propagate past them.

The main purpose of maintaining the detected sets is to allow efficient querying. A query is routed from the top of the hierarchy, across a single path toward the sensor that reported the sought object. Without the information contained in the detected sets, the queries would need to be flooded to all leaves, and would thus be much more costly.

This method can be viewed as a sort of publish-and-subscribe mechanism. The sensors publish the existence of detected objects in the hierarchy, while the querying points are subscribers who want to track current locations of the desired objects.

B. Discussion of STUN

We now discuss several properties of the STUN architecture. First we define some terms.

1) Notation and Definitions

We use a weighted graph $G = (V_G, E_G, l_G, w)$, called the *sensor graph*, to represent the sensors and certain expected characteristics of the moving objects being detected by the sensors. The vertices V_G represent the sensors. The locations of the sensors are represented by a function l_G, mapping senors to geographical locations. The edges E_G indicate adjacencies between pairs of sensors; we say two sensors are *adjacent* if it is possible for objects to move from the detection range of one of the two sensors to that of the other without passing through that of any other sensor. We assume that G is connected, in the sense that any two nodes can be connected through a path.

The weights w represent the movement pattern of the objects in various regions in units of *detection rates*, as follows. Each time an object arrives at or departs from a sensor's detection range, the sensor generates a detection event. We assume that objects move from sensor to sensor so that for each pair of adjacent sensors that an object passes through, the two sensors will experience the same detection rates with respect to that object. This assumption basically says that an object will not terminate its movement in the middle between two sensors,

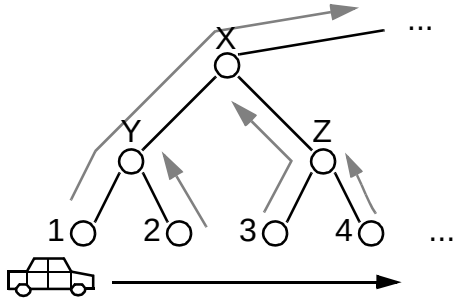

Figure 1: Example of a message-pruning hierarchy. Consider those detection messages from sensors that detect the arrival of the car. Sensor 1's message will update the detected sets of all its ancestors. The messages from sensors 2 and 4 do not update the detected sets of their parents and thus will be pruned there. The message from sensor 3 updates only its parent Z and thus will be pruned at X

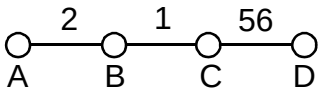

Figure 2: An example 1D sensor graph. Each weight represents the frequency of object movement between a pair of adjacent sensors

or if it does such occurrences must be statistically insignificant. With this assumption, we can assign a weight w to each pair of adjacent sensors, representing the detection rate that each of these two sensors will experience with respect to objects moving between them. For example, the sensor graph in Figure 2 depicts a 1D sensor array with each edge connecting a pair of adjacent sensors. The larger weight on the rightmost edge indicates heavier movement between the two rightmost sensors.

The hierarchical network that connects the sensors is represented by a tree denoted by $T = (V_T, E_T, l_T)$, where V_T and E_T represent the nodes and edges of the tree, respectively, and the function l_T specifies locations of tree nodes. We assume that tree nodes are located in the same space as that of sensors. For example, for the case of a 1D or 2D sensor graph, the tree nodes will be in the same 1D or 2D array as the sensors, respectively. A tree link can be an overlay link over multiple sensor nodes.

2) Performance Metrics: Communication Cost and Delay

We will evaluate the performance of distributed tracking using two metrics. The first is the *communication cost* defined as the total number of detection messages occurring in the network in a unit time. The second is the *delay* defined as the time required for the information about a detection to reach the querying point from the detecting sensor. A good tracking method is characterized both by a low communication cost, and a low delay.

The performance of STUN is determined by the structure of its message-pruning hierarchy and the tracked objects' mobility pattern. Given information about these two items, we can compute STUN's communication cost and delay.

Communication Cost. Given a sensor graph and a message-pruning hierarchy tree T, we can compute the cost, in detection messages per unit time, that the network will incur in order to keep the detected sets of intermediate nodes up to date. In particular, consider the cost contributed by a pair of adjacent sensors, say x and y, with weight $w(x, y)$. Their detection messages, triggered by object movements between x and y, travel upward, but get absorbed at the first common ancestor, say, z, of the two sensors. Thus, the events between x and y induce a cost of $w(x, y)$ messages per unit time on the path from x and y to z.

We illustrate this with the example in Figure 3, where all of the detection events occur between the middle two sensors at a unit rate. These events cause detection messages to travel across the six links indicated by the arrows until they reach a common ancestor, leading to a cost of 6 messages per unit time. Note that these links comprise the only path in T between the two indicated sensors.

We thus define the communication cost for a sensor graph G and hierarchy T as the sum of the individual contributions of all pairs of sensors adjacent in G:

$$CommunicationCost(G, T) = \sum_{(x,y) \in E_G} w(x, y) \cdot PathCost_T(x, y)$$

Since adjacent tree nodes may be physically distant, we define the costs of tree links used in the $PathCost_T$ computation to be Euclidean distances. Thus, the communication cost reflects the required radio power consumption.

In order to obtain hierarchies that yield low communication costs, the intermediate nodes near the root should be connecting points for adjacent sensors sharing low-rate edges. For example, the hierarchy from Figure 3 could have had a lower cost if the adjacent sensors sharing the high-rate edge had been connected by a low level intermediate node, instead of the root. An example of a better hierarchy appears in Figure 4.

Delay. The second metric is the maximum or average delay incurred by messages traveling from leaves to the root. Expressed in units of intermediate nodes, this is equivalent to the height of the hierarchy tree.

The maximum or average delay is smallest in trees which are fully balanced, since their height is minimal. The worst case delay occurs in a degenerate tree where the intermediate nodes form a line graph.

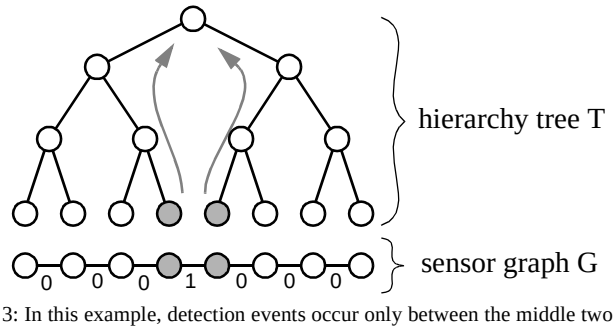

Figure 3: In this example, detection events occur only between the middle two sensors, at a unit rate. Since the detection messages travel to the root, the communication cost of six messages per unit time is incurred at the links indicated by the arrows

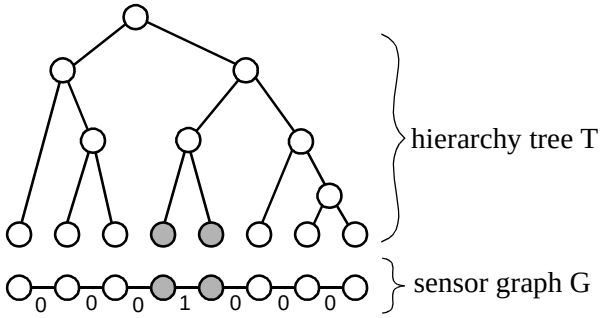

Figure 4: This example shows a hierarchy with a minimal communication cost. The detection messages from the shaded sensors are pruned by their immediate ancestor

III. DAB: Drain-and-Balance Method for Constructing Message-Pruning Hierarchies

In this section we describe a method for constructing desirable message-pruning hierarchy trees, that is, trees for which both the communication cost and the query delay are low.

A. Method Approach and Description

Our method, called DAB (drain-and-balance), constructs the tree in a bottom-up fashion, i.e., from leaves to the root, through a series of *DAB steps*. Within each DAB step, a subset of the sensors is merged into balanced subtrees, without regard for event rates. The effectiveness of the method comes from properly choosing the nodes to merge in each of these steps—this time by utilizing the event rate information. In particular, sensors are partitioned using one or more event rate thresholds, that we will call "draining thresholds," and the high-rate subsets are merged first. We now describe this process precisely.

The input to the method is a sensor graph $G = (V_G, E_G, l_G, w)$, and its output is a hierarchy tree $T = (V_T, E_T, l_T)$. The method is parameterized by a sequence of decreasing "draining" thresholds, $H = \{h_1, \ldots, h_k\}$, where $h_k = 0$. The *k*-step DAB tree construction for a 1D sensor graph proceeds as follows:

1. Initialize T to be an empty graph.
2. For each draining threshold $h_i \in H$, in the increasing order of *i,* perform a DAB step, consisting of the following two phases:
   - 2.1 **Draining**. Add those nodes in V_G into V_T which have at least one incident edge whose weight is greater than or equal to h_i. This inserts a number of singleton trees into T. We say that two trees in T are adjacent if some of their leaves are adjacent in G.
   - 2.2 **Balancing**. Repeatedly merge pairs of adjacent trees in T to form clusters of sensors, in a non-decreasing manner. That is, at each merging step connect the roots of two adjacent trees with a new intermediate node so that the merged tree will have the smallest number of sensors among all possible merges of adjacent tree pairs. The merging process terminates when there remain only non-adjacent trees in T.

Note that the last DAB step corresponds to the threshold $h_k = 0$, guaranteeing that all nodes from G are inserted into T and that the output of the last merging step is a single tree. This follows from the assumption that G is connected.

B. Examples

1) An Example Run

Figure 5 illustrates a 3-step DAB tree construction for a 1D sensor graph. Figure 5a describes the 1D sensor graph, with the weights depicted using vertical bars, and the draining thresholds $H = \{6, 3, 0\}$ indicated on the side. Figure 5b-d shows the forest T at the end of the first, second and third DAB step,

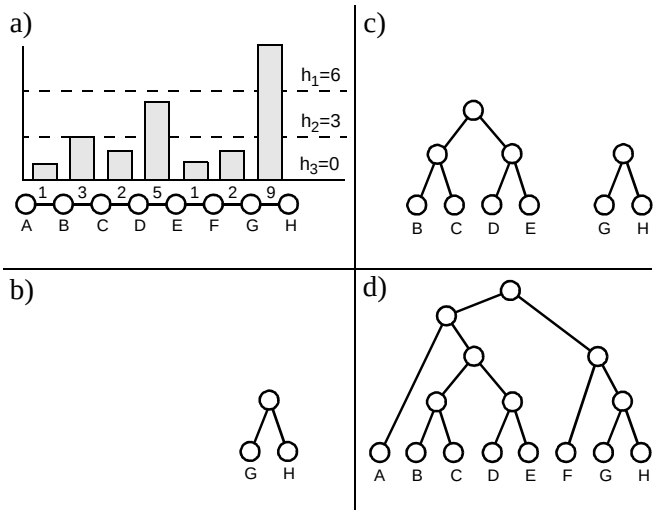

Figure 5: Example of the DAB tree construction for building a message-pruning hierarchy. Part a) shows the 1D input sensor graph G, with the weights depicted using vertical bars, and three indicated thresholds. Parts b, c, d) show the tree after the first, second and last DAB step, respectively. The final tree constructed appears in part d)

respectively. The final tree in Figure 5d is the resulting message-pruning hierarchy tree.

As we can see, processing the sensors with high cost edges earlier lets us place their connecting nodes near the leaves of the hierarchy, as desired. Similarly, the least expensive sensors are processed last, and thus their connecting nodes end up scattered between large regions derived in previous steps, again as desired.

Each step in the DAB tree construction can be viewed as a process of draining water in a region to reveal high peaks and then grouping these revealed high peaks which are adjacent into clusters in a balanced manner. Consider the vertical bars in Figure 5a as peaks. Assume that initially all the peaks are under the water. In step 1, the draining lowers the water down to height $h_1 = 6$, revealing the peak corresponding to (G, H). In step 2, the draining lowers the water further down to height $h_2 = 3$, revealing additional peaks corresponding to (B, C) and (D, E). Finally, in step 3, when the draining lowers the water level to height $h_3 = 0$, all the remaining peaks emerge. This is the reason why we have called the method "drain-and-balance (DAB)."

2) Comparison to Huffman Trees

It is instructive to compare the DAB tree construction with the well-known Huffman tree construction [3, pg. 337], which is also a greedy method. We will do the comparison by way of the example in Figure 6.

First we note that the Huffman tree construction assumes a scenario which is equivalent to the case of distributed tracking where all event updates would travel from leaf nodes to the root without message pruning at intermediate nodes. Under this scenario, the Huffman tree achieves the minimal cost for a given set of event rates associated with sensors. In contrast, the DAB tree construction assumes message pruning at intermediate tree nodes.

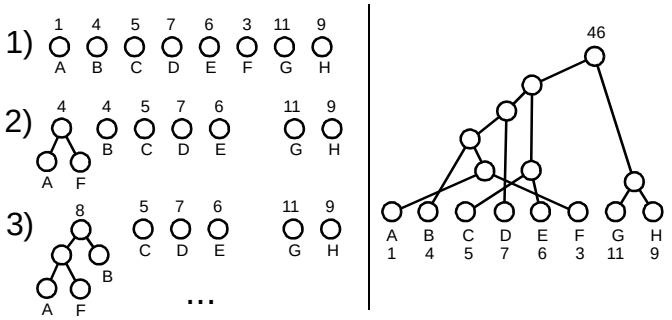

Figure 6: Example of the Huffman tree construction. A node's weight is the sum of the weights of incident edges in Figure 5. In each step, two smallest weight nodes are merged into one, and the new node takes on the sum of the merged nodes' weights. The resulting tree on the right is undesirable for the message pruning purpose, since more expensive nodes are closer to the root.

Second, as we can see in the figure, the Huffman tree construction, by definition, merges the least expensive nodes first. The more expensive nodes, e.g., D and E, end up being connected at intermediate nodes close to the root. This is undesirable for the message-pruning purpose. In contrast, the DAB construction merges the most expensive nodes first, so that heavy traffic can be pruned at intermediate nodes near leaves.

Third, the Huffman tree construction does not concern with tree balancing, unlike the DAB construction. As shown in Figure 6, the tree resulting from the Huffman construction can be relatively imbalanced with relatively long links.

C. Discussion

The choice of the number of steps and the draining threshold for each step in the DAB tree construction may affect the quality of the constructed tree. On one extreme, if the entire tree is constructed in a single step, there is no distinction being made between high and low rate regions, and so the resulting tree does not take advantage of rate information. Such a tree is more likely to be balanced, and thus have low height, since there are no "gaps" in the sensors to force it to a less balanced shape. However, the communication cost for such a tree will likely be high.

On the other extreme, if the tree is constructed using an excessive number of steps, there tends to be a very limited choice of sensors to merge during each step, and consequently very little opportunity to enhance tree balance. In this case, the resulting tree's shape is dictated by the positions of sensors uncovered in each step, possibly leading to a tree with long links and a large height. To achieve a reasonably balanced tree, the number of steps for the tree construction will generally need to be small.

The setting of the numbers of DAB steps and their draining thresholds should generally depend on the movement frequency as a function of locations in the region where objects are tracked. For example, the DAB method with uniformly spaced draining thresholds is expected to work well for traffic patterns where similar variations in movement frequency appear throughout the region, such as the traffic pattern depicted in Figure 7d.

Note that the DAB method may not help for certain traffic patterns such as those exhibiting little locality. For example, very regular traffic patterns, such as a constant or a strictly increasing movement frequency over a 1D region do not offer opportunities for DAB to improve performance over straightforward traffic-oblivious methods.

D. DAB for Multi-dimensional Sensor Graphs

The basic concept of DAB as described above applies to multi-dimensional sensor graphs. Consider, for example, the 2D case. The input sensor graph would now be a weighted mesh with sensors scattered around a 2D region. The draining process is exactly the same as that in the 1D case. However, for the balancing step, the "best" pair of adjacent trees to merge would now be those whose sensors which are contained by the smallest bounding square.

IV. Evaluation

In this section we evaluate the performance of the DAB tree construction based on simulation results; in particular, we apply DAB to 1D and 2D sensor graphs to show it achieves better performance than a method which ignores the traffic pattern information. We first describe a mobility model, called the *city mobility model*, that we have developed for the purpose of generating traffic to support simulation.

A. The City Mobility Model

This model is to simulate large scale object movements which exhibit locality. As we have argued in the beginning of the paper, such behavior is likely to occur in some real-world scenarios.

The model works as follows. The main region in which the object movements are being tracked is divided into several smaller subregions, which represent points of interest such as shopping areas of a city, main squares, etc. These subregions, which we will refer to as *level-1* regions are in turn subdivided further, representing increasingly local attraction points such as popular areas of a shopping center, or the sidewalks in a main square. We will refer to these increasingly nested subregions as *level-2, level-3,* etc. The subdivisions continue until some fixed depth is reached.

Each object travels with a constant speed along a straight path to its destination. Once it reaches the destination, a new destination is selected for the object as follows. First, the object decides with some probability p_1 whether to leave its level-1 region. If so, it selects a destination in a neighboring level-1 region uniformly at random. Otherwise, it remains in the same level-1 region, and repeats the same process at the next lower level, level-2, with probability p_2, and so on until it either reaches a bottom region or decides to leave a current subregion.

The probabilities p_i to leave the current level-i region decrease with the region's size. This forces the object movements to exhibit locality; they are less likely to embark on long

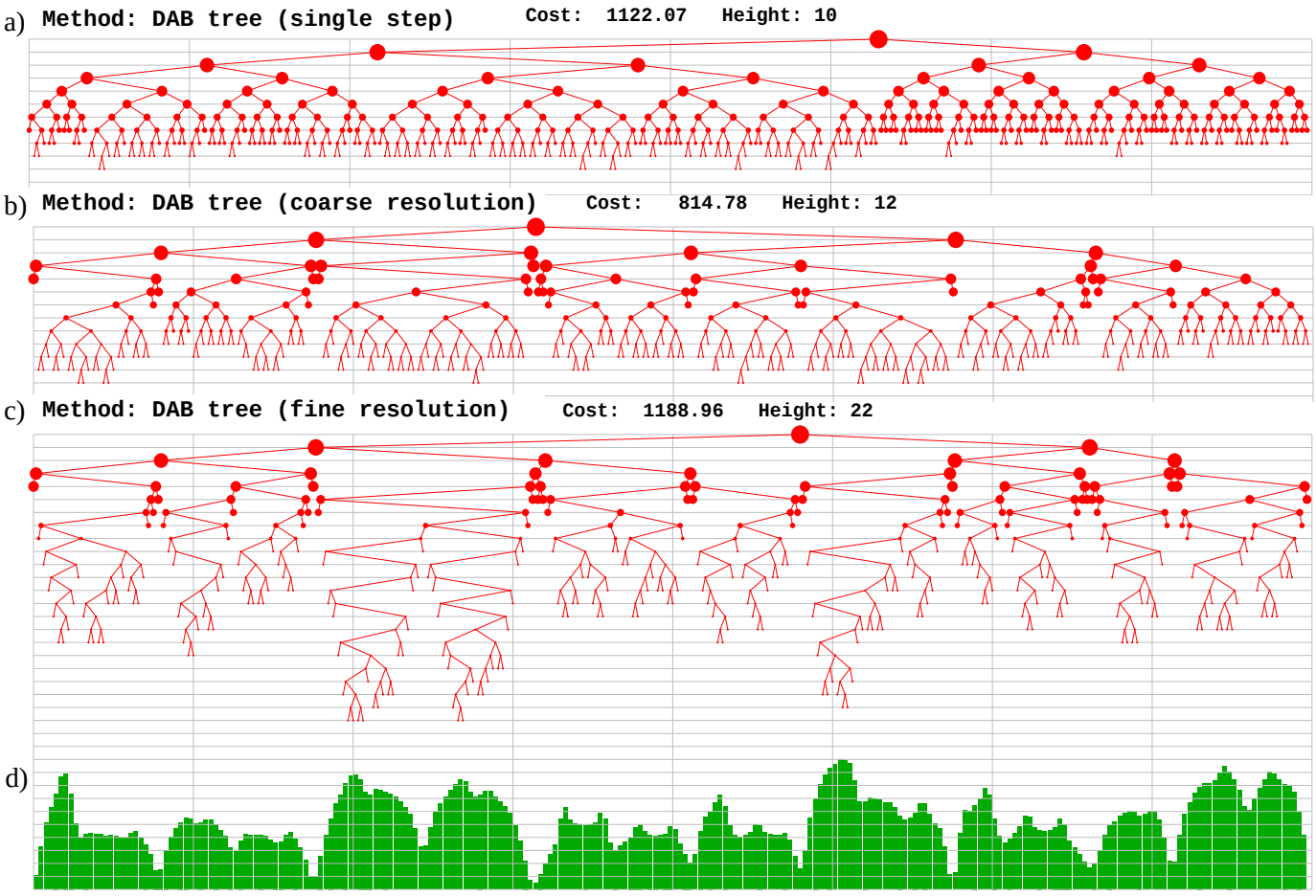

Figure 7: The trees constructed by the DAB method for a 1D sensor graph with 256 nodes. The intermediate nodes are placed between their subtrees, in order to make it easier to inspect the associated weight. d) The input weights are shown at the bottom. a) The top tree, obtained by a single-step run, ignores the weights and places some high level nodes into expensive areas of the region. b) The middle tree, obtained using 4 DAB steps, is best of the three. c) The bottom tree, obtained in 17 DAB steps, suffers large delays due to poor selection of merging choices at each step

trips than they are to embark on short trips. In particular, the traffic patterns we have generated in our experiments use an exponential leaving probability $p(i) = e^{-C2^{d-i}}$, where C is a positive constant, and d is the number of levels of the attraction spots.

B. Description of Results

We show the trees constructed by three types of the DAB method under the city mobility model:

- A *single-step* run, where a single DAB step is used. This type of run disregards the information about movement frequencies, and thus serves as a basis for comparison.
- A *coarse-resolution* run, where DAB uses a small number of steps.
- A *fine-resolution* run, where DAB uses a large number of steps. Even though this type of run utilizes movement frequencies, it suffers from large delays due to limited availability of merging choices in the balancing phase of each DAB step.

In addition, we plot the communication cost and tree height achieved using varying numbers of DAB steps. For both 1D and 2D cases, we use the expected value of the average weight as the first threshold h_1, and exponentially decaying thresholds thereafter. That is, $h_i = h_1 \cdot 2^{-i+1}$. Each data point reported represents the average of 5 runs.

The locations of intermediate nodes were placed to minimize the communication cost as defined in Section 2B, using an optimizing heuristic. However, in Figure 7, intermediate nodes are placed between their subtrees for easy inspection. In experiments where the communication cost of DAB is compared to that of other methods, the same layout optimizing heuristic was applied to all trees under comparison.

1) Results for a 1D Sensor Graph

Figure 7a shows the outcome of the single-step DAB run, where the resulting tree is almost fully balanced. However, as we can see, the high-level nodes in this tree sometimes cut the main region at busy spots, and thus a high volume of detection messages may travel on long paths. This results in a high communication cost of 1122.07.

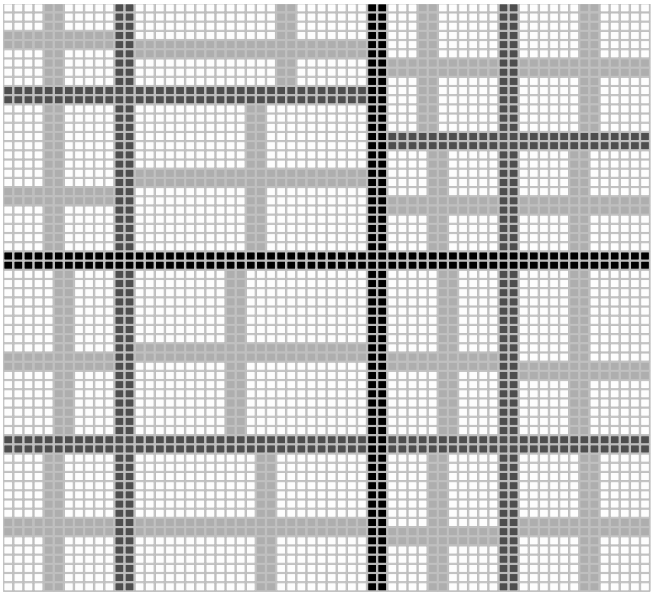

Figure 8: 2D traffic pattern used by 2D DAB, consisting of sensors arranged in a 64x64 grid. Darker and lighter colors indicate smaller and larger weights, respectively

Figure 7b shows the outcome of the coarse resolution DAB run, where the resulting tree is not far from balanced while high level tree nodes cut the main region mainly at spots with small event rates. The communication cost is reduced to 814.78, while the height is 12, just slightly larger than that of the fully balanced tree.

Figure 7c shows the outcome of the fine resolution DAB run, where the resulting tree is both unbalanced and inefficient. The communication cost is 1188.96 and the height is 22.

The plot in Figure 10 compares the communication cost of the DAB method with that of a fully balanced tree for a 1D sensor graph of 4096 sensors, with the number of DAB steps varying between 1 and 10. The DAB tree has lower cost, but the improvement in cost eventually diminishes, because introducing additional small draining thresholds only affects a small fraction of sensors.

Figure 11 shows the average and maximum tree heights obtained for the same scenario. The height of DAB trees is larger than the minimal, as expected; however, the maximum height is not far from the average height, indicating a good worst case delay. In particular, the tree construction avoids building degenerate trees whose heights are proportional to the number of sensors.

2) DAB Results for a 2D Sensor Graph

We evaluate the performance of the DAB tree construction for 2D sensor graphs using the traffic pattern of Figure 8 generated by the city mobility model. The light areas in the diagram denote higher event rates, while darker areas denote areas of less activity. As we can see, this scenario exhibits locality, since most high activity regions are surrounded by the regions of low activity.

Figure 9a shows an approximately balanced tree constructed by using single-step DAB. Similar to the 1D case, all the nodes

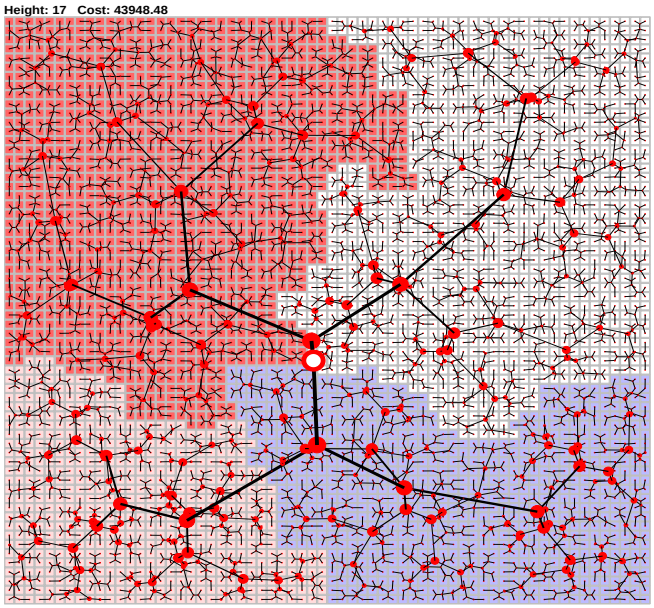

Figure 9: a) A tree resulting from single-step 2D DAB. The different shades denote the areas belonging to one of the four subtrees starting at depth 2. Note that the boundaries do not follow the traffic distribution in Figure 8

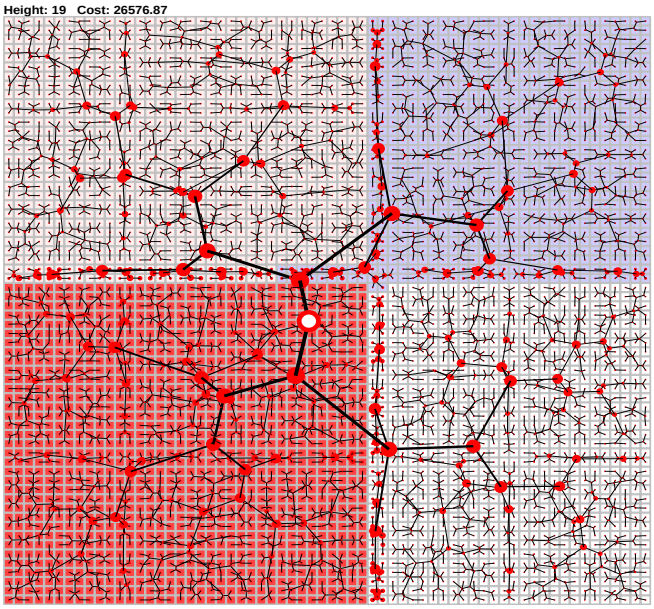

Figure 9: b) A tree resulting from 4-step 2D DAB. The different shades denote the areas covered by one of the four subtrees starting at depth 2. Note that the boundaries here match the areas of low frequency movement found in Figure 8.

are available for combining at once, and the combining process does not take into account the mobility pattern information.

Figure 9b shows a tree constructed by four-step DAB. The earlier steps combine higher cost nodes first, which end up placed near the bottom of the tree. Similarly, the low cost nodes take up the upper levels of the tree. For example, we can see that the borders of the 2 top levels of the tree follow the low-cost regions in the traffic pattern. Thus, only low-rate detection messages end up traveling to the top 2 levels of the tree. As a result, the DAB tree achieves a lower total communi-

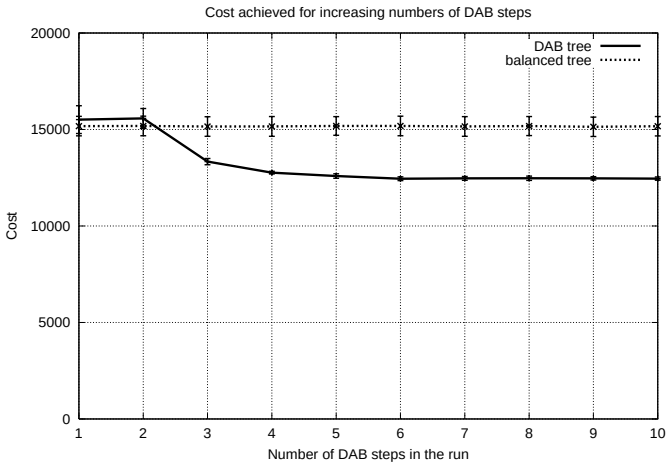

Figure 10: Communication cost for 1D sensor graphs with 4096 sensors

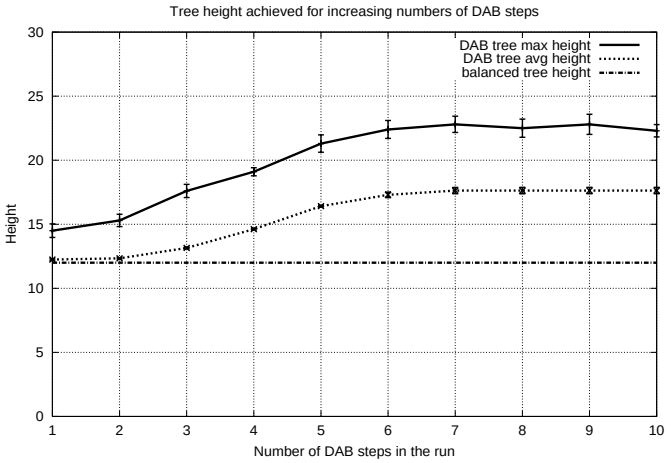

Figure 11: Delay (tree height) for 1D sensor graphs with 4096 sensors

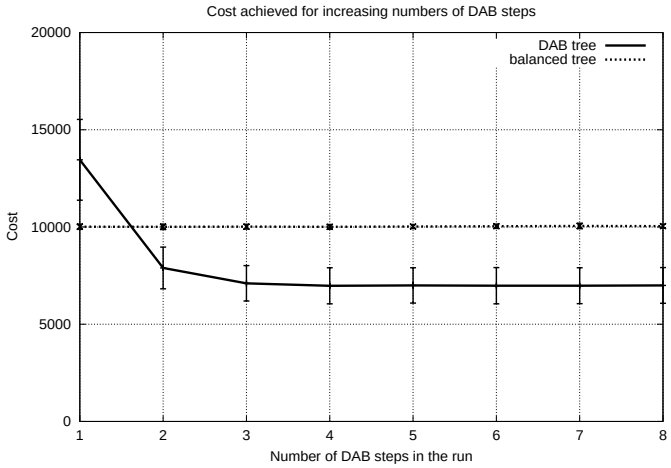

Figure 12: Communication cost for 2D sensor graphs with 64x64 sensors

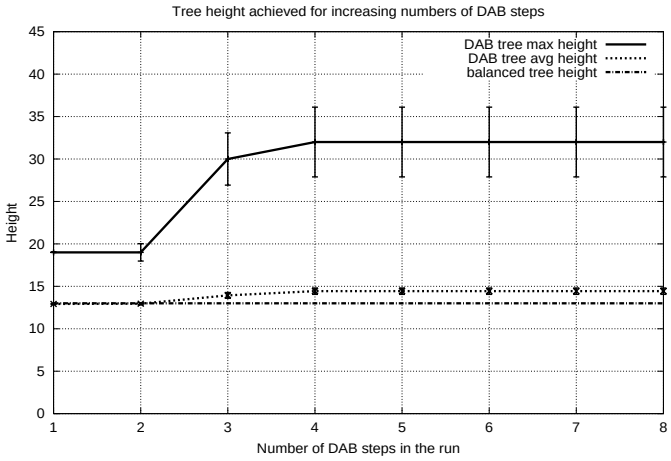

Figure 13: Delay (tree height) for 2D sensor graphs with 64x64 sensors

cation cost, while retaining a relatively small tree height.

The plots in Figures 12 and 13 compare the communication cost and height, respectively, of the DAB constructed trees and those of fully balanced trees for a 2D sensor graph with nodes arranged in a 64x64 grid. The results are similar to those of the 1D case.

V. Conclusion

This paper made two main contributions. First, we described STUN (Scalable Tracking using Networked Sensors), a method for tracking large numbers of moving objects that gains efficiency through hierarchical organization.

Second, we described the DAB (drain-and-balance) method for building STUN hierarchies that take advantage of information about the mobility patterns of the objects being tracked. We have demonstrated that this method gains an advantage over traffic-oblivious schemes when the mobility patterns exhibit locality. As we argued, locality is to be expected in large-scale environments, and so we conclude that the DAB method is useful in large-scale sensor tracking systems.

Acknowledgments. This research was supported in part by the Air Force Office of Scientific Research Multidisciplinary University Research Initiative Grant F49620-97-1-0382, and in parts by research grants from Microsoft Research and Sun Microsystems.

References

[1] Bult, K., Burstein, A., Chang, D., Dong, M., Kaiser, W. J., et. al. “Wireless integrated microsensors,” Proc. of Conference on Sensors and Systems (Sensors Expo), April 1996

[2] Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., and Zhao, J., “Habitat Monitoring: Application Driver for wireless Communications Technology,” to appear in Proc. First ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, San Jose, Costa Rica, April 2001

[3] Cormen, T. H., Leiserson, C. E., and Rivest, R. L., “Introduction to Algorithms,” McGraw-Hill, New York, 1990

[4] Johnson, D. B., and Maltz, D. A., “Dynamic Source Routing in Ad hoc Wireless Networks,” in Imielinski, T., and Korth, H., editors, Mobile Computing, pages 153-181, Kluwer Academic Publishers, 1996

[5] Kahn, J., Katz, R. and Pister K, “Mobile networking for smart dust,” Proc. Mobicom ‘99, August 1999

[6] Pitoura, E. and Samaras, G., “Locating Objects in Mobile Computing,” IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 4, pages 571-592, July/August 2001

[7] Perkins, C. E. and Royer, E. M., “Ad hoc On-Demand Distance Vector Routing,” Proc. 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999

[8] Pottie, G. J., and Kaiser, W. J. “Wireless Integrated Network Sensors,” in Communications of the ACM, Vol. 43, No. 5, May 2000