Energy Efficient In-Network Data Indexing for Mobile Wireless Sensor Networks

Mohamed M. Ali Mohamed[1], Ashfaq Khokhar[1], and Goce Trajcevski[2]

[1] University of Illinois at Chicago, ECE Department, USA
{mali25,ashfaq}@uic.edu
[2] Northwestern University, EECS Department, USA
goce@eecs.northwestern.edu

Abstract. In-network indexing is a challenging problem in wireless sensor networks (WSNs), particularly when sensor nodes are mobile. In the past, several indexing structures have been proposed for WSNs for answering in-network queries, however, their maintenance efficiency in the presence of mobile nodes is relatively less understood. Assuming that mobility of the nodes is driven by an underlying mobility control algorithm or application, we present a novel distributed protocol for efficient maintenance of distributed hierarchical indexing structures. The proposed protocol is generic, in the sense that it is applicable to any hierarchical indexing structure that uses binary space partitioning (BSP), such as k-d trees, Quadtrees and Octrees. It is based on locally expanding and shrinking convex regions such that update costs are minimized. Based on SIDnet-SWANS simulator, our experimental results demonstrate the effectiveness of the proposed protocol under different mobility models, mobility speeds, and query streams.

Keywords: Distributed Algorithms, Mobility, Wireless Sensor Networks, Data Indexing, Query Processing.

1 Introduction

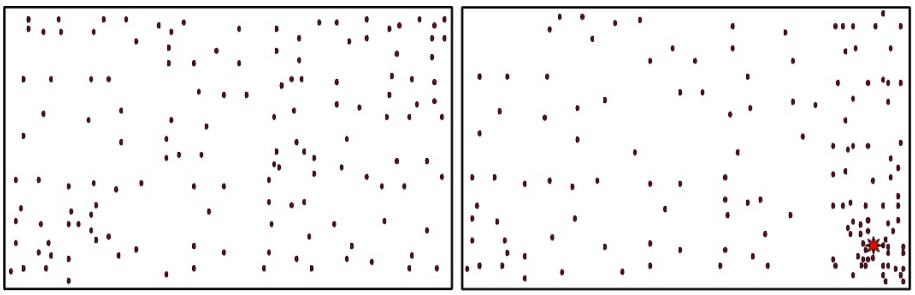
Wireless Sensor Networks (WSNs) have been proposed as effective and efficient distributed systems for monitoring varieties of phenomena in different application domains [1]. In particular, the ability of sensor nodes in WSNs to self organize and provide coverage for monitoring a given region or activity makes them highly useful for scenarios involving harsh conditions or remote surveillance. Typically, individual sensor nodes cooperate in real-time monitoring of phenomena over a given geographic region in two end-of-spectrum modalities: (1) either periodically reporting the sensed values to a given sink (possibly coupled with in-network aggregation); or (2) reporting detections of pre-defined events, i.e., exceeding of a certain temperature-threshold – possibly over spatial extents. Broadly speaking, the purpose of indexing structures in WSN is to facilitate the process of collaboration for monitoring the sensed field, the detection/reporting events of interest, as well as providing in-network storage for answering queries about the sensed phenomena.

M.A. Nascimento et al. (Eds.): SSTD 2013, LNCS 8098, pp. 165–182, 2013.

Mobile sensor nodes [2, 21] greatly increase the adaptability of the WSNs from different perspectives: (1) ensuring a level of Quality of Service (QoS) in response to phenomena fluctuation, in the sense of providing better spatial resolution of sampling in desired/targeted areas; (2) enabling a control over (balancing) the levels of connectivity and coverage. We note that the motion of the nodes may vary in different applications but, from a general perspective, it can be predictable [3], random [4], or controlled [5]. For example, in the data coverage problem in WSN [22], controlled mobility of the sensor nodes is utilized in different applications to achieve more efficacious coverage.

An illustrating example of the motivation for this work is shown in Fig. 1. In Fig. 1(a), a sensed field with randomly deployed sensor nodes is shown. Part (b) of the same figure shows the nodes location distribution, after the occurrence of an event of interest in the southeast corner of the field, where the application or mobility control algorithm (as [29]) has steered more sensor nodes towards that corner, in order to collect more precise information, while still maintaining coverage and network connectivity across the region. Due to this mobility of the nodes required by the application, the underlying distributed indexing structure may become highly skewed, unless it is adjusted to reflect the new distribution of the nodes in a balanced way. The main question addressed in this work is how to efficiently adapt the indexing structures that manage in-network query processing and aggregation in such mobility scenarios, in response to the change of nodes' distribution, such that the overall maintenance cost is minimized. We emphasize that the actual mobility information as to which nodes should move in what direction is given by the application. Also, it is the application responsibility to guarantee minimum number of nodes needed to provide connectivity and coverage. In order to show our work, we use [29] as the dictating application for mobility.

Existing data indexing approaches in WSNs, centralized [6] (i.e., all the data are gathered to one centralized sink node), or distributed [7, 8] – presume that the sensor nodes are *static*, i.e., their locations do not change. Being centralized, they have two-fold disadvantage: (1) increasing traffic towards the sink node, which creates a communication bottleneck; and (2) decreasing the network lifetime, especially in the



(a) **(b)**

Fig. 1. Part (a) - a set of sensor nodes randomly deployed. Part (b) – nodes distribution after occurrence of an event of interest in the southeast corner.

vicinity of the sink node. Some data gathering algorithms employ mobile sink node(s) that traverses the network to gather the data [9-11]. However, a potential drawback of such approaches is the latency/delay.

Organizing the network information across a distributed indexing structure, while knowing that the reporting nodes are not necessarily in the location they reported information from is highly challenging, particularly when the nodes' resources, such as storage, computing power, communication range/bandwidth, and battery capacity are limited. In order to accommodate mobility of the nodes in a general indexing framework, the indexing structure should adapt to the vicissitude of nodes distribution across the field. Once the mobility information for each node is available, the indexing structure should adapt to the change in a distributed fashion to avoid skewed, unbalanced structures. Furthermore, such adaptation should induce minimal overhead.

In this paper we present a novel protocol that enables several existing in-network data indexing structures to incorporate mobile nodes with high transparency. Our approach is applicable to data structures that use Binary Space Partitioning (BSP) [12, 13], where the field is divided into contiguous, non-overlapping, convex regions (e.g., k-d trees, Quadtrees, Octrees [23]). The proposed protocol runs in a distributed fashion, resolving the consequences of the nodes mobility (i.e., relocation) within their regions by locally shrinking or expanding the convex regions, reducing the need to transfer information about this motion across the network or the indexing structure. After a small "transient regime", in the worst case scenario the message cost to re-stabilize the index over the geographic field of interest is linear in the number of the indexing structure nodes. Our simulation results on SIDnet-SWANS simulator [24] show that the cost of maintaining the indexing structure under different mobility scenarios remains sub-linear. In our experiments, over 83% of the mobility in the field is resolved locally, without the need of informing the rest of the network with this mobility. The results also show an overall improvement in the latency of data queries. Note that we do not compare our work to the solutions that use mobile sink nodes, because they use different energy optimization functions which include the consumed energy of mobility. Besides that these solutions focus on data gathering rather than indexing.

The rest of the paper is organized as follows. In Section 2, we start with a preliminary discussion on BSP based hierarchical structure, and outline one such structure that we have used for in network indexing of static WSNs [27, 28]. We use this indexing structure to explain our proposed mobility management protocol. The details of the proposed protocol and its performance analysis are presented in Section 3, followed by a discussion of the applied experiments and simulation results in Section 4, followed by a discussion of related work in section 5. Section 6 concludes the paper and discusses future work.

2 Preliminaries

To better understand the proposed mobility management protocol, we now briefly overview the features of BSP based hierarchical indexing structures. In such structures, recursive splitting is applied to a given space into convex sets via

hyper-planes. As a hierarchical data structure, each node in the BSP tree represents a space that is subdivided among its child nodes. At each level, the number of children of each node represents the fan out, denoted as *k*. The root node of the tree represents first split of the whole space, and the deeper a node in the tree is, the more local (smaller) space split it represents. Each level in the tree contains nodes that embody the whole space partitioned at a certain level of detail. A higher level in the tree represents coarser partitioning (i.e, smaller number of larger subspaces), whereas a lower level in the tree represents more detailed finer scale partitioning (i.e, larger number of smaller subspaces).

In our previous work [27, 28], we have developed efficient abstractions of data and spatial fields in a hierarchical BSP framework. These abstractions are performed for representing the sensed values and positions of the sensor nodes, which we call physical-space abstraction and data-space abstraction, respectively, both rooted at the corresponding sink. Figures 2 and 3 depict numerical examples of physical-space indexing at a leaf node of the indexing tree, as well as assembling/compressing the data in intermediate non-leaf nodes at a given level in a fixed-size array (see [28] for details). We used Wavelet Transform (WT) [30] to combine and compress the data from the children-nodes.

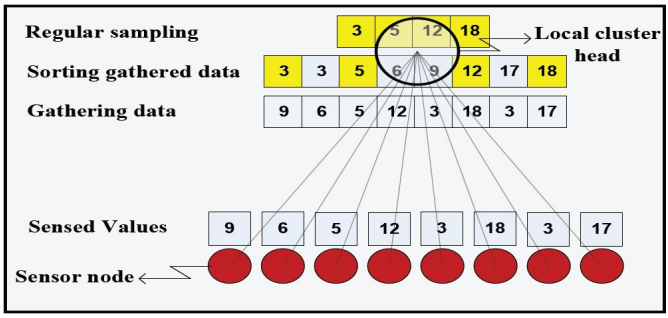


Fig. 2. Processing of sensed values at indexing tree leaf node (Local Cluster Head)

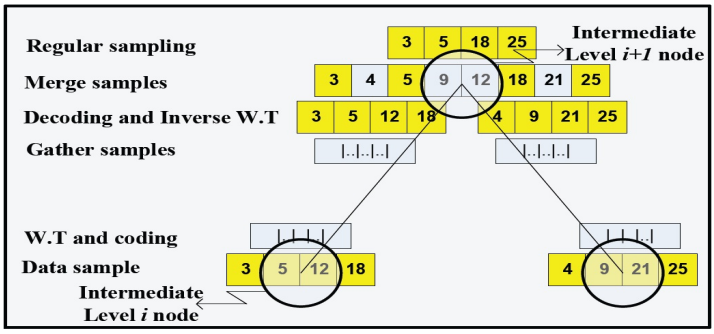


Fig. 3. Processing of sensed values at indexing tree intermediate node(s)

Similarly, the data-space in each region is distributed among a group of nodes managing that region. At the lowest level, the sibling leaf nodes in each locality categorize the reported sensor nodes locations according to data range. Each inner (non-leaf) node receives maps from the nodes in the lower level covering the same data range, having multiple maps received for the same data range across a group of contiguous regions. Fig. 4 shows an example of the process of creating hierarchical maps for the data range (26 – 50) in a given region, and zooming it out to upper level of the hierarchy with a 1:4 factor.

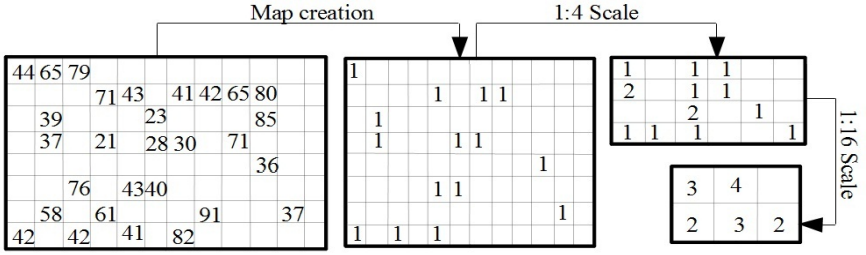


Fig. 4. Data space abstraction process. A set of sensed values in a region (to the left) have a map created for the data range [26-50], then zoomed out twice with 1:4 factor.

Queries to the WSN originate at the sink node, which has a coarse representation for the physical-space and data-space of the whole field, with a specific level of accuracy. If the accuracy requirement for the given query cannot be satisfied by the sink node, it forwards it to its child node(s) according to the query constraints of the physical-space and/or data-space, and the process is recursively repeated until a node(s) is reached capable of providing answer with a required level of accuracy. At this stage the query response is backtracked across the same route in the indexing tree [27, 28].

3 Managing Index Structures with Mobile Nodes

We now proceed with the details of the protocol for adapting the hierarchical indexing structure to capture the mobility of the nodes. The protocol has three distinct stages for which we present the corresponding algorithms and discuss the respective complexities.

3.1 Initial Configuration

Assume that logically there are two types of nodes, senor nodes that sense the field and indexing structure nodes that contain the keys to help maintain the indexing structure. Physically, a node can be a sensor node as well as a node in the indexing structure. Further assume that the number of nodes in the indexing structure is n, and the fan-out of each inner node is k, such that the height of the indexing structure is $O(\log_k n)$. The initial setup of the protocol assigns an integer rank for each

border/hyperplane corresponding to node in the indexing BSP tree , equal to the depth of the node in the tree (i.e., its level-distance from the root). Fig. 5 illustrates a field with randomly deployed sensor with the corresponding borders rank (color-coded with the same colors according to the splitting order). Each leaf node is responsible for (the sensed values of) a group of m sensor nodes within its vicinity. Sensor nodes periodically (with fixed cycle length) report their sensed values and locations to their respective cluster head.

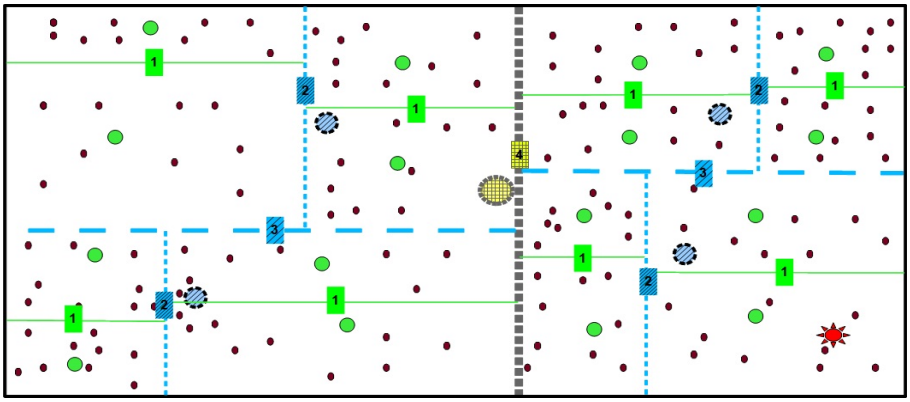


Fig. 5. The field contains (n = 103) randomly deployed (small size/red color) sensor nodes. Example indexing structure in this figure is based on orthogonal bisections, performed recursively, such that 16 (thin solid line/green color) local cluster heads are at the first level. Second level of the indexing structure consists of four (thicker dashed line/blue color) intermediate level cluster heads. Last is the (thickest dotted line/yellow color) sink node. Border line shapes follow same nodes drawing/color. In this (initial) configuration, an event of interest is observed in the South-East corner.

We reiterate that the motion/displacement of the nodes occurs due to a specific objective (e.g., better coverage due to an observed event in a given geographic region) and, as a result, some leaf node(s) in the indexing structure finds more sensor nodes entering to its vicinity and requesting to join. For example, an event of interest may require more sensor nodes to be moved towards, in order to monitor and report more precise data, as depicted in Fig. 5. Also, note that sibling or child/parent node may not be within single hop of each other. In such case, multihop routing of message will be assumed.

3.2 Processing a Request to Incorporate New Mobile Node

Each leaf node has a specified capacity $m' > m$. A leaf node will accept the joining of new sensor nodes coming into its vicinity until reaching the threshold m'. Congestion happens when a new join request is received at leaf node that has reached its maximum capacity m'. The leaf node then initiates a request to reduce the size of its

space of responsibility by changing the position of one of its surrounding borders/hyperplanes.

The process of border change starts with a communication aiming at changing the spatial splitting locally. The leaf node in the indexing structure experiencing congestion starts by locating the border of its surrounding sides corresponding to the lowest rank convex region. It sends to its sibling node(s) on the other side of the lowest rank border, a *change_border_request*. When sibling leaf node receives the *change_border_request* message, it starts assessing if it can change the specified border in order to accommodate some of the sensor nodes currently managed by the requesting sibling. The calculation in this case is based on the capacity of the leaf node that received the request. A response is sent back to the requesting node after the calculation. If all the involved leaf nodes have large populations, then they cannot accommodate more incoming sensor nodes, causing them to reject the request. In such case, since the change cannot be handled locally, a new request for changing borders is propagated in the hierarchy to the node corresponding to the next higher rank - i.e., the requesting leaf node sends the request message to its parent node. Upon receiving the request, the parent node checks if the total number of sensor nodes covered by its children is at the capacity limits. If not, it initiates a request to its sibling on the other side of the smallest rank border of its region. The same assessment algorithm runs at the sibling node, which consequently sends the response back. In case of rejection, the same process is recursively applied - in the worst case, reaching the root of the hierarchy (the sink). The algorithm executed locally by the participating node is formalized below:

Algorithm 1: Forward Mobility Request

Input: *Rank of the border required to change, The count of sensor nodes associated to the requesting indexing node (or its subtree for non-leaf nodes)*

Output: *A border_change_response OR, in case the whole region is congested, it issues a new border_change_request (if request is received from a child node).*

```
Receive border_change_request (Receiver, Sender.Rank, Sender.nodesCount)
    If (Sender.depth == Receiver.depth)          // If the request is received from a sibling node
        extraNodesCount = Sender.nodesCount - Receiver.optimalNodesCountForCluster
        If (Receiver.nodesCount + extraNodesCount < Receiver.maximumNodesCountForCluster)
            newBorderLocation  = calculateNewBorderLocation(Sender.Rank, extraNodexCount)
            send border_change_response(Sender, accepted, Rank, newBorderLocation)
            apply border_change_inform(This, Rank, newBorderLocation)
        Else
            send border_change_response(Sender, rejected, Rank,  Receiver.nodeCount)
        EndIf
    Else          // If the request is received from a child node
        Receiver.UpdateNodesCount(Sender, Sender.nodesCount)
        If(Receiver.nodeCount < Receiver.maximumNodesCountForCluster)
            newBorderLocation = calculateNewBorderLocation(Sender.Rank)
            send border_change_response(Sender, accepted, Rank, newBorderLocation)
            Foreach  childNode other than Sender
```

```
                    send border_change_inform(childNode, Rank, newBorderLocation)
            Else
                Rank = Sender.Rank + 1
                send border_change_request (Sibling, Rank, Receiver.nodesCount)
                requestingBorderChange = True
            EndIf
        EndIf
    End_Receive border_change_request
```

e,ii

The local behavior of the nodes participating in the border-adjustment is formalized in Algorithm 2 below.

Algorithm 2: Receive Mobility Response

Input: *Rank of the border to be changed, The response (accept or reject), The new border location (in case of acceptance)*

Output: *Applies the border change for the node, in case of acceptance, Or initiate new request in case of rejection.*

```
Receive border_change_response (Receiver, response, Rank, newBorderLocation)
    if (response == accepted)
        apply border_change_inform(This, Rank, newBorderLocation)
        requestingBorderChange = False
    Else
        Rank = Sender.Rank + 1
        nodeCount = Sender.nodeCount + Receiver.nodesCount
        send border_change_request (Parent, Rank, nodesCount)
    EndIf
EndReceive border_change_response
```

Complexity: In the worst-case scenario, the request needs to be propagated all the way to the sink node. For a BSP indexing tree consisting of n nodes, with a fan-out factor k, at each level, at most $k - 1$ request message(s) will be transmitted to change the lowest rank border, and $k - 1$ rejection message(s) will be received. In the 2D planar case, $k = 2$ for k-d trees and $k = 4$ if quadtrees are used.

Since, by construction, the height of the BSP with n nodes and fan-out factor k is $\log_k n$, the number of messages required $2 * (k - 1) * (\log_k n - 1)$, bounding the message complexity of the forwarding stage to $O(\log_k n)$. We note that the overall network-wide running time complexity is the same, since each participating node is executing constant operations to check its current capacity.

3.3 Response Propagation

When a border change decision is taken in non-leaf nodes, all their affected child-nodes are notified, recursively propagating the changes until the affected leaf nodes. Leaf nodes, in turn, inform the affected sensor nodes to change their reporting destination. While this border change information message is flowing through the

structure, each recipient node recalculates its population according to the new change to ensure that it is within its capacity. If not, the node finding congestion in its region initiates a new *change_border_request* message and sends it to its sibling node. The important observation is that this particular message is guaranteed to affect borders that are in the sub-tree of the originally changed border, which caused this new congestion, because the capacity has already been checked/verified at the parent or ancestor node.

The determining of the new border location is based on the population size of the requesting (congested) and responding nodes. For that, we rely on the structural properties of the tree's boundary between the nodes at the same level. Namely, we move the border of the node that has a capacity to incorporate new sensors in a direction perpendicular to the current border's position towards the requesting node position, resulting in shrinking the requesting node's area, and accordingly getting more sensor nodes out of its region towards the accepting node's region. The new border location in the low level requests (i.e, requests between leaf nodes) is determined by the requesting node, which knows exactly the location of all its sensor nodes. In higher level requests, the border location change is proportional to the desired new population size of the congested region. After the change takes place, the node that asked for the border change recalculates its new population to ensure it is within its capacity limits. If not, the node reissues a new border_change_request, accordingly. Fig. 6 shows the reconfiguration of the borders after sensor nodes have moved towards an event of interest in the southeast corner of the field.

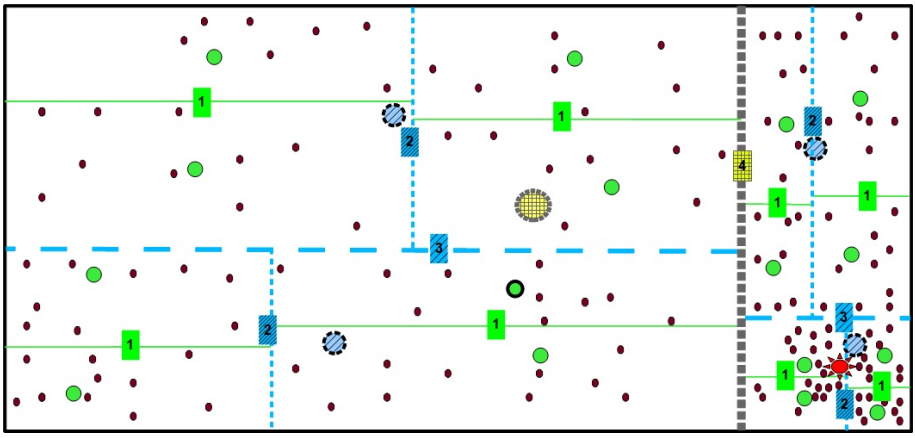


Fig. 6. Borders reconfiguration after sensor nodes are moved towards an event of interest in the southeast corner of the field

The last step of the protocol involves notifying the mobile motes about the new borders of the tree, so that they know which node-ID to use when reporting the sensed values. This if formalized below:

Algorithm 3: Apply and Propagate Mobility Response

Input: *Rank of the border to be changed, The response (accept or reject), The new border location (in case of acceptance)*

Output: *Applies the border change for the node, in case of acceptance, Or initiate new request in case of rejection.*

```
Receive border_change_inform (Receiver, Rank, newBorderLocation)
    Receiver.border[Rank] = newBorderLocation
    If (Receiver.depth == MaximumDepth)          // Leaf node
        Foreach sensorNode
        If sensorNode.Location is out of leaf node new region
            send detach_sensor(sensorNode)
        EndIf
    Else          // Non-leaf node
        Foreach childNode other than Sender
            send border_change_inform(childNode, Rank, newBorderLocation)
    EndIf
EndReceive border_change_inform
```

Complexity: Algorithm 3 executes when Algorithms 1 and 2 have terminated, and is applied to all the children of the subtree rooted at the node at which Algorithm 2 has terminated. In the worst-case scenario, the execution of Algorithms 1 and 2, will cause the request to be forwarded all the way to the sink node. This, in turn, means that each of the n nodes in the tree will have to be notified about borders change (and, eventually, decide upon the new border's location). Assuming an average of h hops communication between the nodes participating in the tree, the total message-complexity of Algorithm 3 is $O(hn)$**.** On the other hand, the computation complexity is bounded by $O(\log m)$ – the capacity of each node. Namely, in the worst case, the neighboring nodes (siblings) will have a difference of m – 1 motes (assuming at least one mote for a minimal occupancy). Sorting the nodes according to the common-boundary coordinate will take $O(\log m)$, plus the constant time for placing the new boundary.

We note that the mobility scenario that would make the protocol for adjusting the tree incur its maximum cost, is having sensor nodes oscillating around the highest rank border, in a way such that their majority moves towards one side of the border within one update cycle causes the indexing nodes to discover congestion and issue border_change_request(s). In the next update cycle, the sensor nodes return back to the other side of the border. In such a scenario, starting from a balanced state, the algorithm behavior would start by a first request at the node(s) adjacent to the highest rank border to change their lowest rank border, which gets accepted at the same level. After the accepting node(s) reach their capacity, while sensor nodes are still crossing the highest rank border towards the adjacent cluster(s), the next request will need to be elevated on level in the indexing tree. On the higher level, the same operation will take place until the managed region is congested.

3.4 Data Indexing under Mobility

The aim of a in-network data indexing system is to arrange and store the sensed data in a distributed fashion. Indexing tree manages the sensor nodes where each group of sensors report their sensed values and positions to a node of the indexing tree. The recipient indexing nodes store the received information, process them, and elevate approximate constructs across the indexing hierarchy. Mobility causes some of the sensor nodes to move apart from their reporting node(s) of the indexing structure, and hence, get into other node(s) vicinity. This causes unbalance in number of senor nodes reporting to the nodes of the indexing structure. Such unbalance results in the reported data across the indexing structure.

In physical-space abstraction, two approaches can be followed. The first approach is to increase the size of the update message according to the count of the sensor nodes population attached to each node of the indexing structure, in order to keep same sampling distance between the update message values. This would not increase the overall size of physical-space update messages traversed, because the total number of sensor nodes in the field is the same. However, it will create a skew in the size flowing in each branch of the indexing tree, where the larger population branches will have larger size update messages than the other branches. The second approach is keeping the update messages size unchanged, at the expense of increase in the accuracy loss across the indexing hierarchy. In other words, upon receiving a physical-space query, there might be a bigger chance of not being able to satisfy its accuracy requirements from the higher level nodes of the indexing tree, and having to forward the query to next level(s) for achieving the required accuracy. The advantage for physical-space abstraction because of the mobility handling algorithm is that the change in number of nodes is bounded by the capacity of each leaf node in the indexing tree m'.

In data-space abstraction, the change occurring is not because of the motion of sensor nodes, but rather because of the modification of borders location to balance the indexing tree. Due to this change, the bitmap constructs used to represent each data-space are increased/decreased in size, in order to represent the new cluster space. Contrary to the physical-space abstraction, which has its skew factor bounded by the capacity of the indexing structure leaf nodes m', the area of a single cluster can increase to approach the size of the whole field. This can only be bounded with the logic of the mobility algorithm, physical constraints of the sensors (i.e., robots moving them), and the field physical barriers. In such extreme case, the large regions can be represented with lower granularity, so the cell size would be coarser than the same level other nodes. This would require high accuracy queries for this region to be forwarded all the way to the leaf nodes. The other solution is to forward the update of such lower level large size cluster(s) as an array of positions rather than a bitmap, and insert them into the bitmap in the higher level node(s) of the indexing tree.

4 Experimental Results

The proposed mobility management protocol was implemented on SIDnet-SWANS simulator for WSN [24]. IEEE 802.15.4 protocol is used for the MAC layer, and

Shortest Geographical Path Routing for the routing layer. The power consumption characteristics are based on Mica2 Motes specifications, MPR500CA. Each sensor node is assumed to have a GPS to obtain the location information. In this section, we present the simulation results and discuss the performance.

The simulations were run for a 300-nodes network, where nodes were randomly deployed in a 500x500 square meters geographic area. The nodes' mobility was assumed under two different mobility models: random and controlled. The controlled mobility refers to a scenario where sensor nodes are moved based on an underlying application requirement. For our simulations we used the algorithm presented in [29] to compute the coordinates of mobile nodes at each step. In the case of random mobility the new location of each node is computed using a random direction. In addition, we also tested the mobility management protocol under different speeds, ranging from 0.5 m/s to 2 m/s, which is practically used in several WSN systems [25, 26]. In our future work we plan to simulate higher speed nodes as well.

For our experiments, we have constructed a K-D tree based hierarchical indexing structure over the sensed field. The index nodes are considered to be static, but would rather be moved according to the borders change, to maintain connectivity with the other nodes in their region. The cycle time in our simulations is 5 seconds, i.e., every 5 seconds, nodes inform their value as well positions to their immediate cluster heads (indexing tree leaf nodes).

We measure the performance of the protocol in terms of following parameters: mobility request latency, mobility resolution factor, and query latency. Mobility request latency refers to the time it takes for the protocol to adjust the structure to reflect the nodes new positions. Mobility resolution factor (MRF) reflects the percentage of requests that required changes beyond the first level of the indexing hierarchy.

Fig. 7 plots the average mobility request latency under different mobility speeds. The performance of both mobility cases is quite stable, where the latency is almost consistent with the change of sensor nodes velocity. The mobility request latency for the controlled mobility scenario (i.e. nodes move towards an events of interest while maintaining coverage [29]) is around 15% higher than the random mobility request latency. This is because the number of mobility request received by the cluster heads in the case of controlled mobility is higher, compared to the random mobility. Note that in the case of random mobility, overall more sensor nodes maybe moving. However, a significant number of consecutive mobility steps may cancel each other, thus keeping the sensor nodes within the same local region. On the other side, in the controlled mobility scenario each sensor node is moving on a specific path towards the target point. Accordingly, with each time step, a node progresses towards moving into or outside of a specific local region, thus requiring mobility adjustment in the indexing structure.

In Fig. 8, MRF is shown for different mobility scenarios. The general trend of the MRF is larger for the controlled mobility algorithm, as the nodes following a specific path are able to cause more disturbance in all the regions they pass by, which creates unbalance in multiple local regions. Because of this unbalance, adjustment to mobility may require adjustment at more than one of the hierarchy. The maximum MRF shown

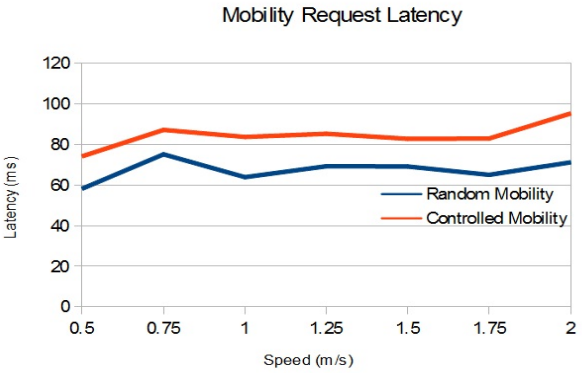


Fig. 7. Average latency of incorporating mobile node in the indexing structure Vs. sensor node speed

for all cases is less than 17%. Which means that the mobility management protocol is able to resolve successfully over 83% of the mobility requests at the lowest level of the indexing tree, without the need of having this mobility information traverse the whole indexing structure.

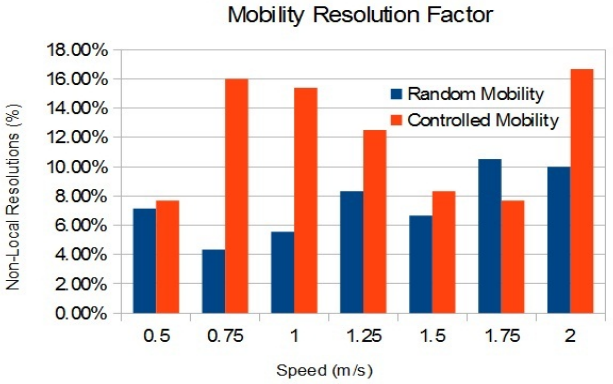


Fig. 8. Mobility Resolution Factor (MRF): The percentage of mobility requests that the mobility protocol is unable to resolve at the lowest level of the indexing structure

Figures 9 and 10 compare the latency of different data queries to the mobility managed structure (under random and controlled mobility) and the static structure where the indexing structure does not change itself to accommodate mobility and thus becomes relatively unbalanced. We present results for three different types of queries.

Physical-space queries inquire values sensed in a specific region. Data-space queries inquire locations of sensor nodes sensing data in a specific data range. A hybrid query inquires either sensed values, or sensor nodes locations, giving constraints of both region and data range. In approximate querying, the user defines a

desired level of accuracy to be met in the response. An example of an approximate hybrid query is:

SELECT TEMPRATURE_VALUES	*inquiring sensed values*
BETWEEN 70º TO 80º	*with data range constraint*
INSIDE RECTANGLE {[0,0],[30,50]}	*and a regional constraint*
WITH ACCURACY = 80%	*at a desired accuracy level*

Fig. 9a shows the difference in data-space query latency for static as well mobility manages structures under different mobility scenarios. The static case shows higher costs for achieving more accurate results. This is because on the lower level of the indexing structure, the static scenario would have a higher memory footprint for the congested regions, which requires more processing and communication time. In Fig. 9b, physical space query latency of the static indexing structure almost matches the mobility managed structure under the random mobility scenario for lower accuracy levels, which is slightly higher than the controlled mobility scenario. However for exact queries (i.e., 100% accuracy), which require the indexing structure to get the data from its leaf nodes, static scenario incurs higher query latency costs.

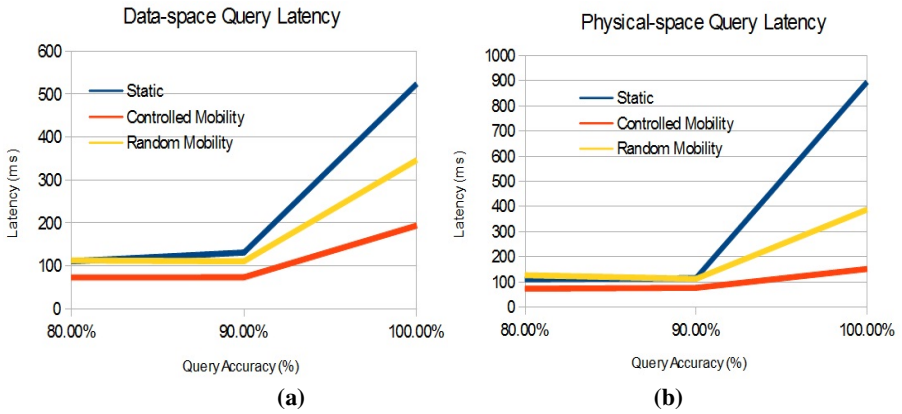


Fig. 9. Query latency for (a) data- and (b) physical-space queries Vs. required query response accuracy

In Fig. 10, the hybrid query latency can be viewed as a combination of latencies of both physical-space and data-space queries, where it is clear that the incurred latency is higher for the static case when requiring higher accuracy level. These results show the efficiency of appropriately handling mobility, and its effect on query latency for most cases of mobility scenario, where the static indexing would not be able to provide same latency for queries inquiring higher accuracy, especially for the queries inquiring exact responses.

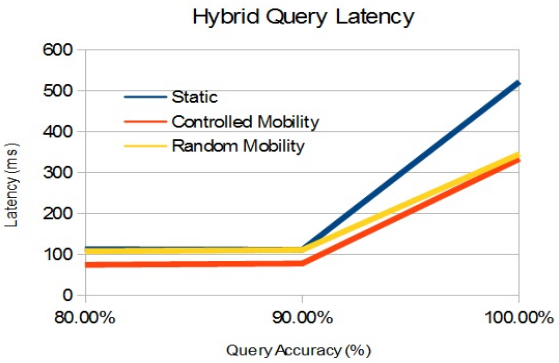


Fig. 10. Query latency for hybrid queries Vs. required accuracy

5 Related Work

Data indexing in WSN has been studied over the past decade, and several algorithms with different perspectives were proposed to solve it. The vast majority of these algorithms did not consider the mobility of sensor nodes. Centralized solutions, as in [6], proposed transmitting data across paths in the network using lifting technique and wavelet based compression. In such methods the network usually suffers from congestion around the sink node, which creates a communication bottleneck, and decreases the lifetime of the nodes in the area around the sink node. Several distributed data indexing algorithms were proposed [7, 8, 14]. In [7], a hierarchical data structure is constructed and data is mapped to the indexing structure using geographic hash tables (GHT). This algorithm creates redundancy in data transmission, where the same raw data is reported to multiple nodes in the indexing structure. Meliou *et al.*[9] proposed an algorithm with a novel idea for data indexing of sensed values in a hierarchical data structure using approximate modeling. Gaussian models were used in this system to abstract large amount of sensed values and elevate them across the hierarchy, leading to more efficient reporting at the cost of accuracy loss across the hierarchy. Such system lacks the representation of sensor nodes positions, and assumes that Gaussian models are suitable for all types of sensed phenomena, which is not generic enough for a wide range of sensed phenomena not of Gaussian distribution nature. Also, Gaussian models are successful in representing the average behavior of a region, but they lose the information about the extreme (maximum and minimum) sensed values, which are of high interest for many WSN applications. Another distributed algorithm proposed by Xiao *et al.*[14] which indexes the WSN data across a spanning tree according to a key for each node of the spanning tree. However this algorithm supports mobility of sensor nodes, it falls short in the maintenance cost of the data updates, as a sensor node may have to update its information at an indexing node that is far from its location. On the other side, if the key is arranged in a way that favors position of sensor node for local region reporting, the system doesn't support data-space indexing efficiently. Monitoring the WSN for

events have been studied in [15], where an algorithm is proposed to use an optimal number of monitoring nodes and minimize false alarms. Such algorithms are useful for event based monitoring applications, which do not consider aggregating the network data as much as answering specific predicates.

Mobile WSN sink node idea in has taken good consideration in recent research. Controlled mobility have been exploited in several works [16-20], in which the – one or multiple – sink node(s) moves in the field and gathers the sensed data. Non-hierarchical solutions, as [16-19], study the optimal path to move across the field, in order to minimize latency. In [20], Xing *et al.* propose at two tier system of mobile sink node(s) which collects data from static rendezvous points that collect sensed data locally within their vicinity. This clustered data gathering approach increases the efficiency of data gathering and scheduling for sink node(s) mobility, however it doesn't provide a full hierarchical solution. It does not present a distributed data indexing solution, but rather an optimized data gathering algorithm based on clustering. Moreover, the energy minimization criteria is significantly different in such solutions, because the amount of energy spent on mobility is orders of magnitude higher than the energy spent on communication and computation.

6 Conclusion and Future Work

In this paper we presented a protocol to manage and maintain in-network indexing structures in WSN under the constraint of mobile nodes. The protocol is applicable to BSP tree structures, where it is based on assigning incrementing values for space splitting borders of the BSP tree. The protocol is based on shrinking and expanding the indexed regions according to the residing number of nodes, in order to keep a balanced load for the indexing structure. The complexity of the proposed solution does not exceed a linear order in the size of the indexing structure. Our results show the capability of handling over 83% of mobility within their local regions of occurrence, without the need of communicating this information across the network. The average latency of balancing the structure in the presence of mobility is in reasonable range. The results also show improvement for query latency results, especially for the higher accuracy queries. In our future work, we plan to incorporate mobility models that involve higher mobility speed and uniform direction. In addition, we also plan to study mobility management under higher dimensional indexing structures that do no involve orthogonal bisections. An extension of our work is to consider the mobility of the nodes participating in the indexing structure itself. Another extension is to incorporate the aspect of optimizing the coverage for multiple-events monitoring.

Acknowledgments. This research has been supported in part by the NSF grants CNS 0910988, 0910952 and III 1213038.

References

1. Zhao, F., Guibas, L.: Wireless Sensor Networks: An Information Processing Approach. Morgan Kaufmann (2004)
2. Ekici, E., Gu, Y., Bozdag, D.: Mobility-Based Communication in Wireless Sensor Networks. IEEE Comm. Magazine 44(7), 56–62 (2006)
3. Shah, R.C., Roy, S., Jain, S., Brunette, W.: Data MULEs: Modeling a Three-Tier Architecture for Sparse Sensor Networks. In: 2003 IEEE Workshop Sensor Network Protocols and Applications, SNPA 2003 (May 2003)
4. Chakrabarti, A., Sabharwal, A., Aazhang, B.: Using Predictable Observer Mobility for Power Efficient Design of Sensor Networks. In: Proc. 2nd International Workshop on Information Processing in Sensor Networks (2003)
5. Somasundara, A., Ramamoorthy, A., Srivastava, M.: Mobile Element Scheduling for Efficient Data Collection in Wireless Sensor Networks with Dynamic Deadlines. In: Proc. 25th IEEE International Real-Time System Symposium (2004)
6. Ciancio, A., Pattem, S., Ortega, A., Krishnamachari, B.: Energy-Efficient Data Representation and Routing for Wireless Sensor Networks Based on a Distributed Wavelet Compression Algorithm. In: Proceedings of the 5th International Conference on Information Processing in Sensor Networks, IPSN 2006, pp. 309–316 (2006)
7. Greenstein, B., Estrin, D., Govindan, R., Ratnasamy, S., Shenker, S.: DIFS: A Distributed Index for Features in Sensor Networks. Ad Hoc Networks 1, 333–349 (2003)
8. Meliou, A., Guestrin, C., Hellerstein, J.: Approximating Sensor Network Queries Using In-Network Summaries. In: Proceedings of the International Conference on Information Processing in Sensor Networks, IPSN 2009, pp. 229–240 (2009)
9. Goldenberg, D., Lin, J., Morse, A., Rosen, B., Yang, Y.: Towards Mobility as a Network Control Primitive. In: Proc. ACM MobiHoc (2004)
10. Wang, Z., Basagni, S., Melachrinoudis, E., Petrioli, C.: Exploiting Sink Mobility for Maximizing Sensor Networks Lifetime. In: Proc. 38th Ann. Hawaii Int'l Conf. System Sciences, HICSS 2005 (2005)
11. Gandham, S., Dawande, M., Prakash, R., Venkatesan, S.: Energy Efficient Schemes for Wireless Sensor Networks with Multiple Mobile Base Stations. In: Proc. IEEE Global Telecomm. Conf., GlobeCom 2003 (2003)
12. Fuchs, H., Kedem, Z., Naylor, B.: On Visible Surface Generation by A Priori Tree Structures. SIGGRAPH 1980 Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, pp. 124–133. ACM, New York (1980)
13. Thibault, C., Naylor, F.: Set operations on polyhedra using binary space partitioning trees. In: SIGGRAPH 1987 Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 153–162. ACM, New York (1987)
14. Xiao, L., Ouksel, A.: Scalable Self-Configuring Integration of Localization and Indexing in Wireless Ad-hoc Sensor Networks. In: IEEE International Conference on Mobile Data Management, MDM, vol. 151 (2006)
15. Liu, C., Cao, G.: Distributed Monitoring and Aggregation in Wireless Sensor Networks. In: Proc. of Infocom, pp. 1–9 (2010)
16. Gandham, S., Dawande, M., Prakash, R., Venkatesan, S.: Energy Efficient Schemes for Wireless Sensor Networks with Multiple Mobile Base Stations. In: Proc. IEEE Global Telecomm. Conf., GlobeCom 2003 (2003)
17. Luo, J., Hubaux, J.: Joint Mobility and Routing for Lifetime Elongation in Wireless Sensor Networks. In: Proc. IEEE INFOCOM (2005)

18. Wang, Z., Basagni, S., Melachrinoudis, E., Petrioli, C.: Exploiting Sink Mobility for Maximizing Sensor Networks Lifetime. In: Proc. 38th Ann. Hawaii Int'l Conf. System Sciences, HICSS 2005 (2005)
19. Hanoun, S., Creighton, D., Nahavandi, S.: Decentralized mobility models for data collection in wireless sensor networks. In: IEEE International Conference on Robotics and Automation, ICRA (2008)
20. Xing, G., Li, M., Wang, T., Jia, W., Huang, J.: Efficient rendezvous algorithms for mobility-enabled wireless sensor networks. IEEE Transactions on Mobile Computing (2012)
21. Pileggi, F., Fernandez-Llatas, C., Meneu, T.: Evaluating mobility impact on wireless sensor network. In: UkSim 13th International Conference on Modelling and Simulation, pp. 461–466. IEEE (2011)
22. Mulligan, R., Ammari, H.: Coverage in Wireless Sensor Networks: A Survey. Network Protocols and Algorithms 2(2) (2010)
23. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley (1990)
24. Ghica, O., Trajcevski, G., Scheuermann, P., Bischoff, Z., Valtchanov, N.: Sidnet-swans: A simulator and integrated development platform for sensor networks applications. ACM SenSys (2008)
25. Pon, R., Batalin, M., Gordon, J., Kansal, A., Liu, D., Rahimi, M., Shirachi, L., Yu, Y., Hansen, M., Kaiser, W., Srivastava, M., Sukhatme, G., Estrin, D.: Networked Infomechanical Systems: A Mobile Embedded Networked Sensor Platform. In: Proc. Fourth Int'l Symp. Information Processing in Sensor Networks, IPSN 2005 (2005)
26. Dantu, K., Rahimi, M., Shah, H., Babel, S., Dhariwal, A., Sukhatme, G.: Robomote: Enabling Mobility in Sensor Networks. In: Proc. Fourth Int'l Symp. Information Processing in Sensor Networks, IPSN 2005 (2005)
27. Mohamed, M., Khokhar, A.: Dynamic indexing system for spatio-temporal queries in wireless sensor networks. In: 12th IEEE International Conference on Mobile Data Management MDM, vol. 2, pp. 35–37 (2011)
28. Mohamed, M., Khokhar, A., Trajcevski, G., Ansari, R., Ouksel, A.: Approximate hybrid query processing in wireless sensor networks. In: Proceedings of the 20th International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2012), pp. 542–545. ACM, New York (2012)
29. Caicedo, C., Zefran, M.: A coverage algorithm for a class of non-convex regions. In: IEEE Conference on Decision and Control, pp. 4244–4249 (2008)
30. Chui, C.: An Introduction to wavelets. Academic Press Prof. Inc., San Diego (1992)