Dynamic Multi-Root, Multi-Query Processing Based on Data Sharing In Sensor Networks[1]

Zhiguo Zhang, Ajay D. Kshemkalyani and Sol M. Shatz
Department of Computer Science
University of Illinois at Chicago
Chicago, Illinois 60607

Applications that exploit the capabilities of sensor networks have triggered significant research on query processing in sensor systems. Energy constraints make optimizing query processing particularly important. This paper addresses multi-root, multi-query optimization for region queries. The work focuses on application-layer issues exploiting query semantics. The paper formulates three algorithms: a naïve algorithm, without data sharing, and a static and heuristic data-sharing algorithm. The heuristic algorithm allows sharing of partially aggregated results of preconfigured geographic regions and exploits the location attribute of sensor nodes as a grouping criterion. Simulation studies indicate the potential for significant energy savings with the proposed algorithms.

Keywords: Sensor networks, distributed query processing, geographic coverage, multi-query optimization

---

1. INTRODUCTION

Wireless sensor networks consist of relatively inexpensive sensor nodes capable of sensing, computing, and wireless communication. This makes sensor networks a promising platform for many applications, such as military reconnaissance, disaster rescue, traffic surveillance, manufacture automation, and construction and environment monitoring. However, compared to general wireless networks, sensor networks have significant resource constraints, including limited bandwidth, computation power, and energy supply (usually the result of a non-replaceable battery). All these constraints make energy conservation in system design an important research area.

There are two main sources of energy consumption in sensor nodes. One is consumption in the active waiting state [2][8], and the other is consumption due to communication [26]. Zheng [28] proposed an optimal wakeup-sleep scheme to maximize sensor nodes in a sleep state, to reduce energy consumption in the waiting state. To reduce energy use associated with communication, a number of research efforts use in-network aggregation [1][3][13][14][17][25]. Others use data caching at locations that minimize packet transmissions [16] or special network routing to minimize messages needed for query processing [4][12]. For example, we previously proposed a grouping technique based on query-informed routing to make in-network aggregation more energy efficient [27].

One way to extract sensor data from a distributed sensor network is by using mobile agents that selectively visit the sensors and incrementally fuse appropriate measurement data [22]. Another technique, which is the subject of this paper, is to inject queries into the network, treating the sensors as a distributed database [7]. The sensors are programmed through declarative queries in a variant of SQL [14]. The following is an example of such a query for monitoring harmful radiation in some building:

SELECT room, AVG(radiation) FROM sensordb WHERE building = ERF GROUP BY room HAVING AVG(radiation) > 100 DURATION 30 days EVERY 1 minute

[1] This material is based upon work supported by the U.S. Army Research Office under grant number W911NF-05-1-0573.
An earlier version of this paper appeared in the Int. Conference on Distributed Computing in Sensor Systems (DCOSS), 2008.

Most previous research on query processing in sensor networks has focused on the processing of a single long-running aggregation query (see, for example [13][14][25][27]). As an extension to this line of research, Trigoni et al. [20] and Emekci et al. [6] considered the case of reducing message transmission by sharing sensor readings for multiple queries, where queries are represented by particular query regions. For a given query, the *query region* $[X_{min}, X_{max}] \times [Y_{min}, Y_{max}]$ is the geographical region that serves as the source of sensor data to be retrieved by that query. We consider such region queries. For example, for the query "SELECT AVG(temperature) FROM sensorDB WHERE position.X>=25 and position.X<=75 and position.Y>=40 and position.Y<=60 DURATION 1 day EVERY 1 minute", the query region is [25, 75] × [40, 60]. Considering other predicates, such as "WHERE temperature>80", in conjunction with region queries would be an interesting generalization to the work presented here. To simplify the presentation and focus on the key idea of data sharing, we do not consider further such a generalization.

Existing multi-query processing techniques work only in a centralized environment, and require that queries arrive at a common root node. In this case, since all query regions are known by this common root node, intersection regions (the intersection areas among query regions) can be computed at the root node, making it possible to share partially aggregated results of intersection regions. Since these methods use centralized computation of intersection regions at the root node, they are not directly suitable for multiple queries injected from different root nodes. In addition, although existing methods attempt to share the sensor-value readings of intersection regions, they do not account for the effect of the routing structure on the efficiency of aggregation, and do not address the problem of how to group sensor nodes according to query regions. As a result, many intermediate nodes need to unnecessarily wake up and transfer messages for sensor value readings of nodes that reside in the same query region, but which belong to different queries. The routing tree structure of sensor networks can have significant impact on the aggregation efficiency of data retrieval in sensor networks. Therefore, using query information in the construction of a routing tree can provide improvement in terms of reducing message transmission. Of course in practice, network properties such as link quality and MAC protocols also significantly impact messaging performance. Since this paper focuses on application-layer issues by proposing key ideas exploiting query semantics, "lower-level" networking properties are left for future investigation.

This paper formulates and addresses for the first time the problem of *multi-root, multi-query processing* for long-duration aggregation queries over regions. This problem arises in many applications where loosely-coupled, or independent, stakeholders want to gather information from a common (shared) sensor network. As a specific example, consider a case of environmental monitoring, where scientists studying wildlife migration and climatologists studying pollution patterns are operating from different locations, but both need to monitor average rainfall volumes associated with different regions in a forest-based sensor field. Another example arises in a battlefield situation, where two remotely located battalions want to monitor enemy troop movements in different but partially overlapping battlefield sectors.

We consider the most general case, where multiple queries are injected asynchronously into the network at different nodes, which we designate as root nodes. Note that such a root node can be any sensor node of the deployed sensor network; root nodes are not some special distinguished processor nodes. In addition there is no global knowledge of the different queries, and hence completely distributed solutions are required. We consider

three algorithms – a naïve algorithm (NMQ), a static algorithm (SMQ), and a dynamic, heuristic algorithm (ZMQ). The heuristic algorithm is based on dynamically sharing partially aggregated results of preconfigured geographic regions, and it exploits the novel idea of applying a grouping technique for optimization of multi-root, multi-query processing, by using the location attribute of sensor nodes as the grouping criterion. This optimization aims to maximally share the reading and transmission of sensor node values belonging to multiple queries.

Since the goal of our heuristic approach is to share sensor readings and data transmission among different queries that have intersecting query regions, we group sensor nodes so that nodes in an intersection region only need to send their sensed values once, independent of the number of queries involved. To facilitate this grouping, the sensor field is divided into zones, and a *logical data aggregation tree* is established to hierarchically represent the zones. In a distributed and asynchronous manner, a query taps into the data aggregation tree at the lowest possible tree node such that the zone represented by that node's subtree contains the geographic area of its query coverage. The idea of using a recursive tree for query dissemination and data retrieval has been used in various contexts, such as for spatial aggregation queries in sensor networks [12]. Our approach becomes increasingly effective as the query regions of the multiple queries increasingly overlap.

We performed extensive simulations on the discussed algorithms. The NMQ algorithm treats queries independently and does not exploit any sharing of the aggregated data by sharing of message transmission. Our simulation studies indicate that the SMQ and ZMQ algorithms provide a significant reduction in message transmission, which increases energy savings, under a wide range of network conditions and query region options. The three algorithms are shown to provide interesting tradeoffs. One approach to further validate the findings of this work is to perform measurement experiments using some sensor network testbed.

Summary of Contributions: This paper addresses the problem of processing multiple aggregation queries introduced at different nodes in large-scale sensor networks, and makes the following contributions:

1) To the best of our knowledge, this is the first work to formulate and address the optimization of *multi-root, multi-query processing* for region queries. Previous approaches to multi-query processing optimization assumed a common root node for all queries, and used a centralized approach to compute the overlap regions. Our approach regards single-root multi-query processing as a special case of multi-root, multi-query processing.

2) This research provides a distributed algorithm, ZMQ, for determining intersection query regions of multiple queries, and also proposes a static algorithm, SMQ, that serves as a benchmark.

3) By deriving the ZMQ algorithm, this research also extends query-informed routing from single query processing to multi-query processing.

4) The research provides extensive simulations that show that the ZMQ algorithm performs reasonably well under a wide range of sensor network deployments and query region options. ZMQ is compared with the static algorithm, SMQ, as well as the naïve algorithm, NMQ, which does not exploit any data sharing or transmission sharing.

The remainder of the paper is organized as follows. Section 2 provides background information, including system assumptions and challenges, and an overview of two key areas of related research – using grouping to support query-informed routing and defining sensor localization based on sensor field zones. Section 3 proposes three algorithms for multi-root, multi-query processing: the distributed, heuristic (zone-based) algorithm, ZMQ, the

static algorithm, SMQ, and the naïve algorithm, NMQ. Section 4 provides details and examples for the ZMQ algorithm. Section 5 discusses simulation experiments to evaluate the ZMQ algorithm against the SMQ and NMQ algorithms. Finally, Section 6 provides a conclusion, including some discussion of issues related to sensor node failures, and outlines directions for future work.

2. BACKGROUND

2.1 System Model

The sensor network system model and the solution framework for multi-root, multi-query optimization make the following assumptions:

1) The queries to be optimized refer to the same type of sensor data, such as the temperature of the environment. Multiple queries that probe for different types of sensor data can be considered independently, and such query processing can be simplified by using techniques like attribute correlation [10].

2) Each node knows its geographical position and the scale of the sensor field. Since GPS-enabled sensors incur a high cost and increased power consumption, GPS is not a practical option for all sensor nodes. However, much research has explored other techniques for localization in sensor networks [18][21][23], making this assumption reasonable.

3) A query region is a rectangle aligned along the X and Y axes. For a region that is not aligned with the axes, or not naturally rectangular, one can approximate the region by identifying one or more rectangular query regions that cover the region of interest. An algebra of grid-fields can also be used for dealing with arbitrarily gridded datasets [11].

4) For simplicity of presentation, we start by assuming that all queries use the same sampling rate. Of course, in practice, multiple queries can have different sampling rates, and our method does generalize for such cases. We will discuss the implication of different sampling-rate queries in Section 4.4.

Our approach to multi-root, multi-query processing in sensor networks is motivated by the goal of sharing sensor readings and data transmission among different queries with intersecting query regions. We identify two key challenges in designing a distributed solution for optimization of multi-root, multi-query processing:

Challenge One (C1): How to determine the intersection regions of multiple queries when those queries are injected into the sensor network at different sensor nodes.

Challenge Two (C2): How to group nodes in different query regions for the sake of aggregation efficiency.

To address challenge **C1** we use the notion of "zones" to represent query regions. A zone is a subdivision of the geographical extent of a sensor field, and each sensor node can independently compute the zones according to the scale of the sensor field [12]. More details on zones are given in Section 2.2.2. Since zones are predefined when the network is deployed, intersection zones are easy to identify even for queries injected at different root nodes.

To address challenge **C2**, we apply a grouping technique [19][27] to group sensor nodes in the same zone, forming an aggregation-efficient tree topology for multiple queries. Grouping sensor nodes in the same zone into a

subtree not only increases aggregation efficiency, but also makes possible the sharing of partially aggregated results of zones. The grouping technique we use is reviewed in Section 2.2.1.

In Section 3, we will show how to use the notion of zones and the grouping technique to compose a fully distributed solution to the multi-root, multi-query optimization problem.

2.2 Related Work

2.2.1 Using Grouping for Query-Informed Routing

Query processing in sensor networks typically proceeds in three phases: (i) disseminating queries into the network, (ii) sensing data, and (iii) retrieving data from the network. For phases (i) and (iii), a tree topology is formed using some variant of the broadcast and convergecast techniques, e.g., [5] presents an optimized broadcast protocol using an adaptive-geometric approach. Here, each node performs two actions: 1) according to the messages it receives, each node decides its own tree level and selects a parent node with respect to the tree topology being created, and 2) the node broadcasts its own id[2] and tree level. Once all nodes in the network have established their tree levels and parent nodes, the tree topology is defined.

In previous methods [13][14][25], sensor nodes select parent nodes only according to tree levels. The grouping techniques used in [19] and [27] are explicitly motivated by the fact that it is common for queries in a sensor network to be aggregation queries (such as COUNT, MAX, MIN, AVERAGE) using "GROUP BY" or "WHERE" clauses. Such queries can form aggregation groups according to a specific attribute of the sensor nodes, and there are situations where these queries must remain active over long durations. The basic idea of the grouping techniques is to try to force those sensor nodes with the same specific attribute – which is used to form groups according to the "GROUP BY" clause or to form query regions according to the "WHERE" clause – to be logically close to each other when forming the tree topology. This will result in partial aggregates being formed as low as possible in the tree topology. The work here is based on the specific grouping technique described in [27].

Figure 1 illustrates the main idea of reducing the length of messages transmitted by intermediate nodes, saving energy associated with those nodes and increasing lifetime of the sensor network. Nodes labeled with "B" in Figure 1 represent "blue nodes" (some arbitrary attribute), and we assume that a query is injected at node $S_{1,1}$.

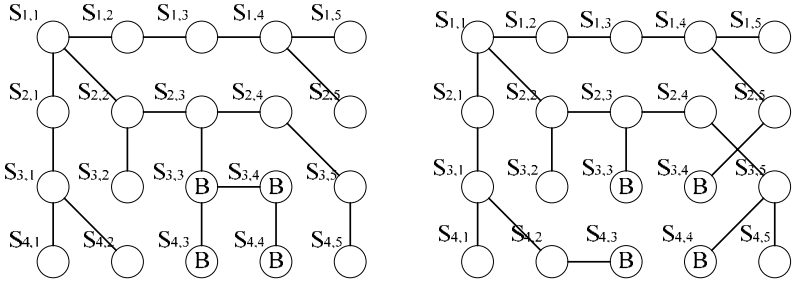

Fig. 1a. Tree topology formed using the grouping technique method

Fig. 1b. Tree topology formed using the conventional method

Figure 1. A processing example for queries having WHERE clause.

[2] We adopt the common assumption that each node in the sensor network has a unique identifier, the node's id.

Consider the query: SELECT SUM(value) FROM sensordb WHERE color=blue.

If the tree topology is formed by using the grouping technique, as shown in Figure 1a, we can see that the aggregation will be completed at blue node $S_{3,3}$. Nodes $S_{2,3}$, $S_{2,2}$, and $S_{1,1}$ only need to transmit a 1-tuple aggregated result to the user. In this example, only seven nodes (including those blue nodes) send messages, and each node only needs to send a 1-tuple message. Other nodes can remain in a low-power idle state. However, if the tree topology is formed by sensor nodes selecting parent nodes non-deterministically, the result may be like that shown in Figure 1b. In this situation, nearly all sensor nodes must receive and transmit a 1-tuple message. For example, blue node $S_{3,4}$ will need to transfer its result via the path $S_{2,5}$-$S_{1,4}$-$S_{1,3}$-$S_{1,2}$-$S_{1,1}$, and node $S_{4,3}$'s result must travel via the path $S_{4,2}$-$S_{3,1}$-$S_{2,1}$-$S_{1,1}$. Note that $S_{3,3}$'s result goes through $S_{2,3}$, where the result can aggregate with the result obtained from $S_{4,4}$ (which travels via the path $S_{3,5}$-$S_{2,4}$-$S_{2,3}$) and then move via the path $S_{2,2}$-$S_{1,1}$ to the user. Because so many nodes are involved in message handling, the energy consumption is high.

The basic strategy of grouping is to influence the routing tree topology construction by leveraging aggregation groups or location groups defined by the queries. In this paper, we exploit this idea in the ZMQ algorithm by grouping nodes based on an existing concept called "zones," which is explained in the next subsection. Our method of grouping by zones is discussed in Section 4.2.

2.2.2 Sensor Field Division Using Zones

Zones are a subdivision of the geographic extent of a sensor field. A zone is defined by the following constructive procedure [12]. Consider a rectangle R in the x-y plane. Intuitively, R is the bounding rectangle that contains all sensors within the network. We call a sub-rectangle Z of R a *zone* if Z is obtained by dividing R k times, $k \geq 0$, using a procedure that satisfies the following property: After the i-th division, $0 \leq i \leq k$, R is partitioned into 2^i equal sized rectangles. If i is odd (even), the i-th division is along the values of the y-axis (x-axis). Thus, the bounding rectangle R is first sub-divided into two zones at level 1 by a vertical line that splits R into two equal pieces. Each of these sub-zones is then split into two zones at level 2 by a horizontal line, and so on. This zone splitting continues until each sensor node belongs to a unique zone. We call the non-negative integer k the *level* of zone Z, *i.e.,* $level(Z) = k$.

A zone can be identified either by a zone code, $code(Z)$, or by an address, $addr(Z)$. The code $code(Z)$ is a bit string of length $level(Z)$, and is defined as follows: If Z lies in the *left* half of R, the first (from the left) bit of $code(Z)$ is 0, else 1. If Z lies in the *bottom* half of R, the second bit of $code(Z)$ is 0, else 1. The remaining bits of $code(Z)$ are recursively defined on each of the four quadrants of R. This definition of the zone code matches the definition of zones given above, encoding divisions of the sensor field geography by bit strings.

Figure 2 shows a deployed sensor network and the zone code for each zone. The zone in the top right corner of the rectangle R has a zone code 1111, and its level is 4. The address of a zone Z, $addr(Z)$, is defined to be the rectangle defined by Z. Each representation of a zone (its code and its address) can be computed from the other.

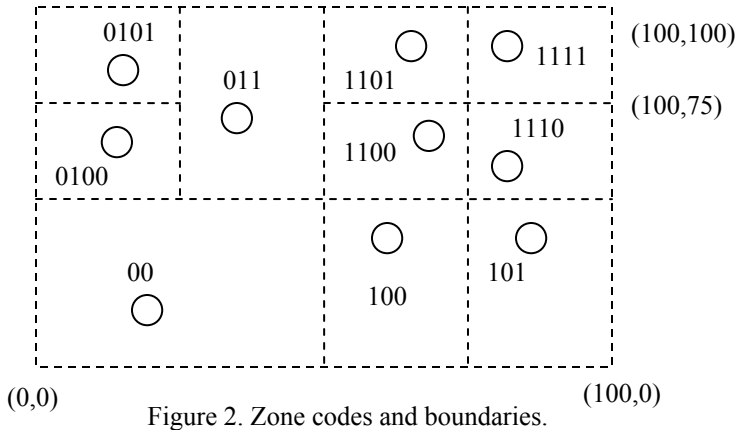

Figure 2. Zone codes and boundaries.

The zone with code 1111 represents the region [75, 100] × [75, 100] in sensor network space [0, 100] × [0, 100], where space $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ represents the geographic rectangular region with left-bottom point (x_{min}, y_{min}) and right-top point (x_{max}, y_{max}). Similarly, given a region [25, 75] × [50, 100], we can know that this region contains zone 011 and zone 110. We use the same prefix of zones to represent a bigger zone that contains those zones. For example, a zone with code 11 would include zones 1100, 1101, 1110, and 1111. We define *Prefix*(*codea, codeb*) as the longest common prefix of *codea* and *codeb*. For example, *Prefix*(1110, 11) equals 11.

Since each node knows its position and the scale of the sensor field, geographic zones can be predefined once the network is deployed. Each sensor node knows its own zone code and the scale of the sensor field, and hence it knows the geographical region that any zone code represents. Given a query represented by a query region, all sensor nodes can independently identify the same set of zones that represent that query region. This facilitates the distributed computing of intersection regions.

3. MULTI-QUERY MODELING

Multi-query optimization in sensor networks seeks to share data readings and data transmissions for nodes in the intersection regions of multiple query regions. In this section, we use an example to explain three algorithms – NMQ (Section 3.1), SMQ (Section 3.2), and ZMQ (Section 3.3) – for multi-root, multi-query processing. NMQ simply treats each query independently, and does not achieve any optimization. It establishes a base level for performance comparison. SMQ is a static algorithm, in the sense that it optimizes based on up-front knowledge of the multiple query regions. ZMQ is our proposed heuristic method, which does not require any advance knowledge of the queries. It is based on dynamically sharing partially aggregated results of preconfigured geographic regions. We also explain how to solve the challenges **C1** and **C2** for SMQ and ZMQ. The details of algorithm ZMQ are then discussed in Section 4.

3.1 The Naïve Method (NMQ) for Multi-Root, Multi-Query Processing

We use the example of Figure 3 to explain the first and simplest algorithm for multi-root, multi-query processing, the naïve algorithm (NMQ). In Figure 3, *Q1* and *Q2* are injected from two different nodes *R1* and *R2*. The different rectangles represent the two different query regions, which we denote as *QR1* and *QR2*. The NMQ algorithm sets up different tree structures for *Q1* and *Q2* separately. This naïve algorithm does not share any sensor readings. The grey

nodes, which are the nodes in the intersection region of query regions for *Q1* and *Q2,* need to send the same readings to different parent nodes twice, once for *Q1* and once for *Q2*. Figure 3 shows the tree structures for *Q1* and *Q2* separately. A better algorithm would be to share the readings and messages of the grey nodes by both *Q1* and *Q2*.

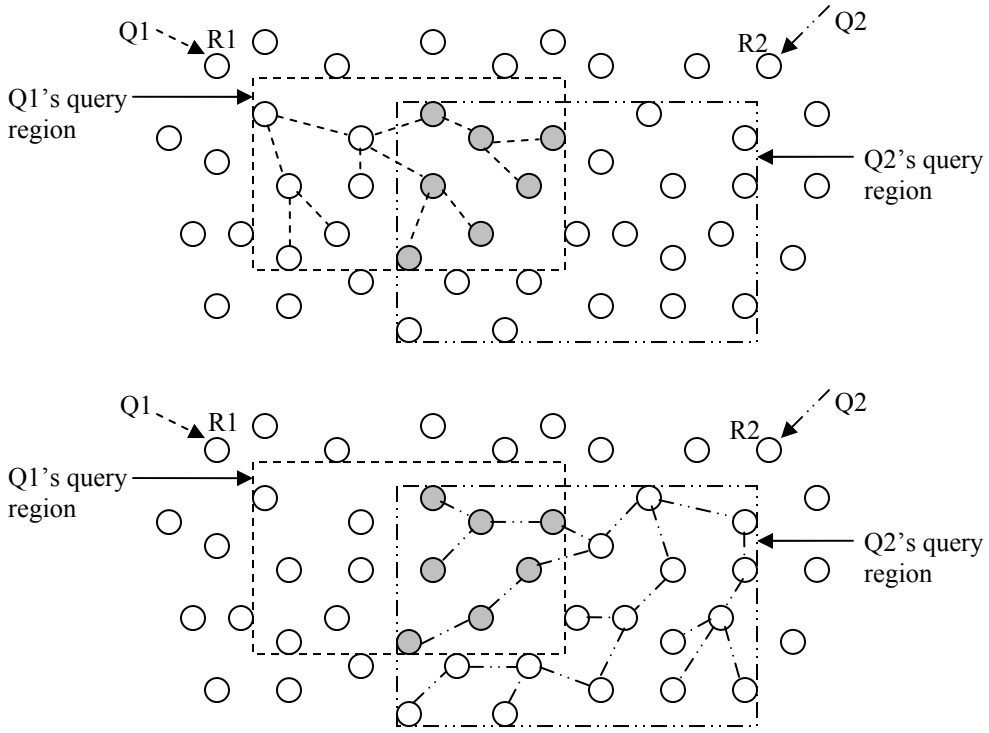

Figure 3. NMQ for multi-root multi-query processing.

3.2 Static (SMQ) Multi-Root, Multi-Query Processing

In order to lower the cost of multi-query processing, we seek to share the sensor readings of sensor nodes in the intersection regions. Here, the two previously mentioned challenges, **C1** and **C2**, need to be solved.

To solve **C1**, reading-sharing methods need to know the identity of the nodes in the intersection regions. Consider Figure 3. Since *Q1* is known only to node *R1*, and *Q2* is known only to node *R2*, one option is to let the network first construct the tree structure for *Q1* and then adjust the tree structure in the intersection region when *Q2* is propagated in the network. Nevertheless, this readjustment in the middle of processing *Q1* would be difficult and energy consuming. Another approach for handling the intersection regions is to simply assume that the multiple queries (and thus query regions) of interest are known ahead of time, i.e., to consider the static case. While this may not be realistic in practice, it does allow us to create a benchmark algorithm that is semi-optimal in terms of taking advantage of pre-computed intersection regions. We defined an algorithm (SMQ) that preconstructs subtrees for the intersection regions and for the non-intersecting regions of queries. Figure 4 illustrates the subtrees for the same deployment example presented in Figure 3. Note that here, one and only one subtree is constructed for each region (including intersection regions). The SMQ algorithm then establishes paths from the root nodes *R1* and *R2* to those subtrees using a shortest path method, as illustrated in Figure 5.

The basic procedure for the SMQ algorithm is as follows:

1. Compute the following regions: The intersection regions for all query regions, and the non-intersecting regions associated with each query region.

   For the example of Figure 4, we obtain the following three regions: $QR2 \cap QR1$ (intersection region), QR1 – QR2 (non-intersecting region associated with QR1) and QR2 – QR1 (non-intersecting region associated with region QR2).

2. For each region R identified in Step 1, find the node $r \in R$ with min($d1 + d2 + \ldots + dk$), where di is the hop-count distance from node r to the root node for query i, and query i contains region R. Let r be the root node of a minimum spanning tree for region R.

   For the example of Figure 4, node a is the root node for region QR1 – QR2, since this node has the shortest hop-count distance to node R1. Likewise, node b is the root node for region $QR2 \cap QR1$, since this node has the shortest combined hop-count distance to both nodes R1 and R2.

3. Setup routing trees using the shortest hop-count paths from the root nodes of each spaning tree to the root nodes of the corresponding queries.

   The resulting routing topology for Figure 4 is shown in Figure 5.

However, since the multiple regions are only known after all queries are injected into the network, this method is applicable to multi-root, multi-query processing only in the *static* case, so that the tree structures for the involved regions can be setup in advance. Even if we could identify such regions that work well for a set of multiple queries, it would not allow for the practical situation when queries arrive dynamically or even dynamically change to specify a different set of query regions. Still, the SMQ method can serve as a benchmark.

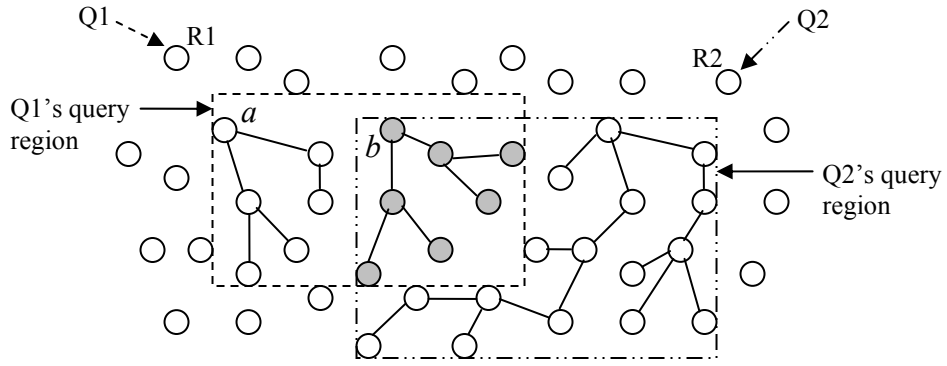

Figure 4. SMQ subtrees for multi-query processing.

If we can know the intersection regions (i.e., if **C1** is solved) we can use the grouping technique to solve **C2** – by grouping nodes in the same region into one subtree. In our simulations in Section 5, we use this method to compute the results for SMQ. Section 5 provides further details on the implementation.

The SMQ algorithm computes intersecting regions after all queries have been decided and injected into the network, but before setting up the routing topology. This guarantees that each overlapping region forms one and only one sub-tree, thus maximizing the sharing of sensor data.

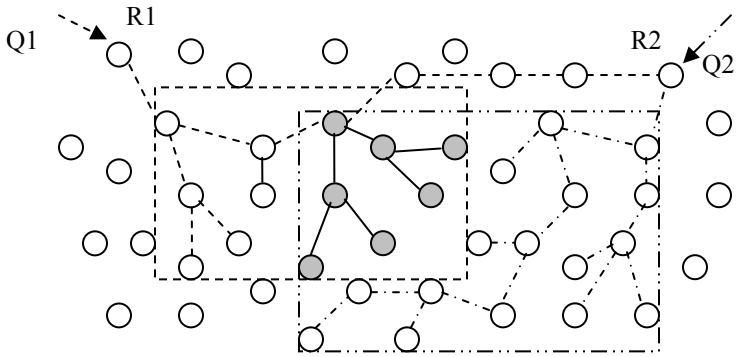

Figure 5. SMQ topology for Figure 4.

3.3 Zone-Based (ZMQ) Multi-Root, Multi-Query Processing

To increase the sharing of data provided by sensor nodes in the presence of dynamically arriving queries, we predefine a set of globally known regions in the network, and then represent query regions in terms of these predefined regions. The reason that the set of regions is defined to be globally known is to provide different nodes the same view of the sensor field and allow them to use these commonly known regions to represent the same query region. Figure 6 shows an example of such predefined globally known regions, represented as a hierarchical tree. Each node in the tree represents one such region. For example, the whole network $[0, X] \times [0, Y]$ is one globally known region, and $[0, X/2] \times [0, Y/2]$ is another. The set of globally known regions forms a tree structure such that the region represented by a parent node consists of the regions represented by its children nodes.

Since a node cannot know ahead of time what the appropriate intersection regions will be, it is not feasible to pre-setup exactly one region for each intersection region. So, we use a set of predefined globally known regions to represent all possible query regions, and an intersection region is represented by one or more such globally known regions. When a query is injected into the network, the root node of the query computes a set of regions that can be used to represent the query region, and then establishes paths from itself to the root nodes of those regions. Queries can share the partially aggregated results of those regions that are a part of different query regions. Although the root nodes of queries cannot know the intersection regions before knowing the queries, all root nodes can use identical views of the globally known regions to represent query regions. Thus, the intersection regions for any set of queries will be a set of globally known regions.

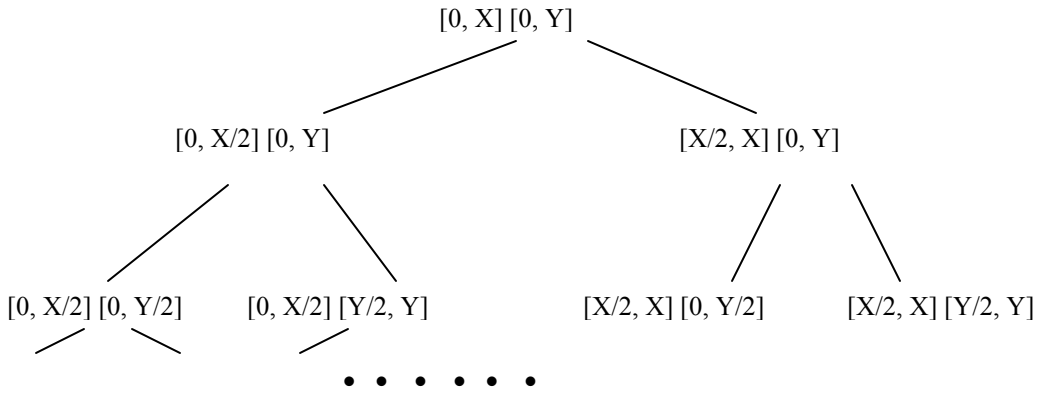

Figure 6. Examples of predefined globally known regions.

The framework of globally known regions solves challenge **(C1)**. This framework is implemented using zones, as reviewed in Section 2.2.2. In other words, zones are a means for representing these specific regions. To solve challenge (**C2)**, the ZMQ algorithm uses the grouping technique, reviewed in Section 2.2.1, to group nodes in each zone into one subtree. All those zone subtrees form a globally pre-setup tree.

Thus in ZMQ, each node knows the globally known regions, and each node can easily determine the globally known region to which it belongs. (Each node computes its own zone code, and from this the node can then easily determine if it is in a given zone.) Furthermore, the representing globally known regions can be easily computed given a query region. Given the size of the sensor field and a zone code, a node can easily compute the region represented by this zone. Therefore, intersection regions are easy to determine, even in a distributed environment. In the next section, we explain details for the zone-based algorithm (ZMQ) for multi-root, multi-query processing.

4. ALGORITHM ZMQ (ZONE-BASED MULTI-ROOT, MULTI-QUERY PROCESSING)

Algorithm ZMQ is based on predefined zones and a statically pre-setup tree structure for those predefined zones. When multiple queries are injected, each root node of a query computes the zones of the query region, and sends the query to those zones. Shared zones among queries can share sensor readings and message transmissions inside the zones.

4.1 Zone Setup

The system model assumes that each sensor node knows its location and the scale of the network region (see Section 2). Thus, each node can learn the locations of neighbors within radio range through direct-broadcast communication. Upon hearing any neighbor node, a node builds its zone code and boundaries accordingly. The first split of the whole sensor field creates two sub-zones, 0 and 1. Then, upon detecting a new neighbor, a sensor node splits its current zone either vertically or horizontally into two equal sub-zones, and then adjusts its zone code accordingly. By this technique, each node knows its own zone code. All those zones form the zone-tree structure, as shown in Figure 7. The parent-child relations in Figure 7 represent containment, where the parent zone is comprised of child zones. Each node in the zone tree is a zone, and the path from the root node to the current node is the zone code of the zone represented by the current node.

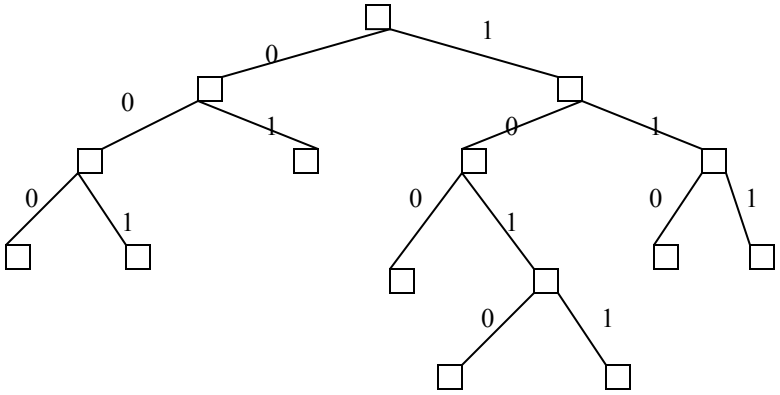

Figure 7. The zone tree of all zones.

The rectangular region represented by each zone is decided once the construction of zone codes is complete. For example, zone 01 represents the region [0, X/2] × [Y/2, Y], while zone 1011 represents the region [3X/4, X] × [Y/4, Y/2]. After computing their zone codes, sensor nodes in the same zone automatically form groups based on the zone codes they computed, as shown in the next section.

4.2 Sensor Node Grouping in Zones

The method to group sensor nodes in each zone uses the grouping technique introduced in Section 2.2.1. In contrast to the grouping technique in [27], which used the "GROUP BY" or "WHERE" information in queries to group sensor nodes, the grouping technique employed here uses the information of "zones" to group sensor nodes.

In the algorithm *GROUPING_ZONE* (and as used later), A is the sensor node executing the algorithm, Z_A is the zone represented by A, and $code(Z_A)$ is the zone code of Z_A. B is any neighbor node of A. Operator "+" is for string concatenation. Zone code *ES* (empty string) represents the whole sensor field zone. Operator ">>n" eliminates the last n characters of a string; for example, the result string for "110011>> 1" is "11001".

Each node executes this algorithm after it detects that all its neighbor nodes have decided their zones. The basic idea is that a node A (that is not the root node) first searches neighbor nodes in its immediate parent zone (the smallest zone contains Z_A), to find a node with minimum zone code. If such a node B exists, and its zone code is smaller than A's zone code, then A selects B as its grouping parent. Otherwise, A searches neighbor nodes in the parent zone of its immediate parent zone, and so on, until it finds a parent.

GROUPING_ZONE

1. *code* = *code_old* = code(Z_A) // Initialize zone code variables
2. While (*code* ≠ *ES*) // Iterate until *code* represents the whole network
3. *code* = *code_old* >> 1 //*code* represents the zone containing the zone represented by *code_old*
4. *T* = ∅ // Initialize the temporary set *T*
5. For each *B* where *B* is neighbor of *A*
6. If (*Prefix*(*code*, code(Z_B)) = *code* and *Prefix*(*code_old*, *code*(Z_B)) ≠ *code_old*)
7. *T* = *T* ∪ {*B*} // Put any neighbor *B* into set *T*, if *B* is not inside
8. // zone of *code_old*, but is in the zone of *code*
9. If (*T* = ∅)
10. *code_old* = *code*
11. Continue //Back to line 2 to try the upper level zone
12. Let *C* be the node in *T* with smallest code
13. If (*code*(Z_C) < *code*(Z_A))
14. Select *C* as *A*'s grouping parent node
15. Return // Parent node of *A* is *C*
16. *code_old* = *code*
17. Return // *A* is the root node

Figure 8 illustrates a simple example of the algorithm *GROUPING_ZONE*. Node 11 has three neighbors, 01, 00, and 10. This node first searches the direct parent zone of zone 11, which is zone 1, and finds node 10, which has a smaller zone code than itself. Therefore, node 11 selects node 10 as its grouping parent node. The root node of zone 11 is node 11. Similarly, node 10 also has three neighbors. This node first searches its direct parent zone 1, and finds node 11, which has a bigger zone code than itself. So node 11 searches zone 1's parent zone which is zone *ES*, the whole region, and it finds that node 00's zone code is the smallest one. Node 10 would select 00 as its grouping parent node. The grouping-tree structure for the four nodes is shown in Figure 8. Similarly, Figure 9 shows the result of the zone grouping for Figure 2.

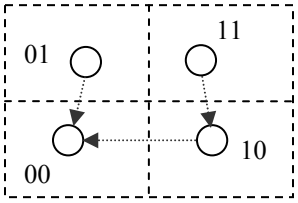

Figure 8. Grouping by zone.

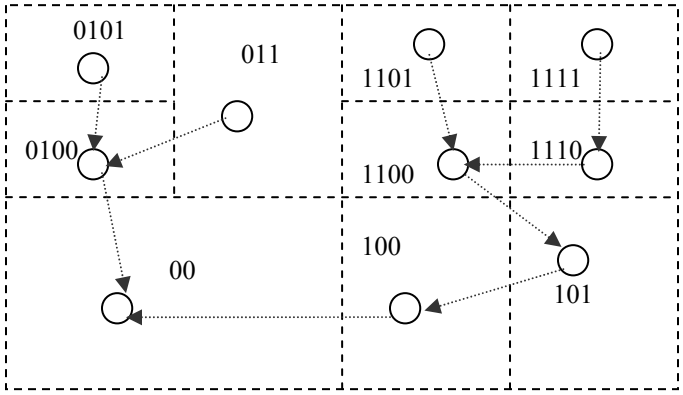

Figure 9. Grouping-tree of Figure 2.

Definition 4.1. *Zone links* for the grouping-tree network are the parent-child links formed using the GROUPING_ZONE algorithm. A node's zone link is the link from the node to its parent. All the links in Figure 9 are zone links.

Zone links are preconstructed once a network is deployed and before any queries are injected into the network. Each node has a unique zone link. The zone-link tree formed by zone links has the property that nodes in the same zone are in one subtree. This achieves the method of grouping by zone. Later, when queries are received, the routing network only needs to be adjusted to include paths from the root nodes of zones to the root nodes of queries. These paths are established by "forward links," as defined in the next section.

Another advantage for grouping by zones is that the grouping-tree structure can serve as a semantic tree structure that contains information about zones in subtrees, which can be used to reduce unnecessary broadcasting to propagate queries into the network; this directly saves sensor node energy. The concept is compatible with other approaches that employ a semantic tree to help reduce energy usage in sensor networks (e.g., [24]).

Note that for unevenly distributed networks, or in the case of node failures, the *GROUPING_ZONE* algorithm may form several tree structures in one network. Consider Figure 9 as an example. If node 100 is absent (may be due to failure), which means node 101 cannot communicate with node 00, then two tree structures will be formed — one rooted at node 00 and the other rooted at node 101. However, even in the case of such multiple tree structures, the routing-tree construction algorithm of the ZMQ method would still form one routing tree for each query. We will discuss this situation further at the end of Section 4.3.2.3.

It is possible for multiple nodes to end up with the same zone code. For example, in Figure 9, if node 100 and node 101 were out of communication range, then neither node would detect any other node in zone 10. Thus, both nodes would claim code 10 and form two subtrees for zone 10. In such a case, ZMQ works the same way as it does for multiple subtrees in the whole sensor network, with some loss of data sharing due to separated subtrees for one zone. Consider also the case that there is some third node that can detect node 101 and node 100, but still node 101 and node 100 can not detect each other. These three nodes will each initiate splits that continue until reaching a node that does not detect any other node in its same zone. One of two situations is reached – either each node will have its own zone code, which is the general case for ZMQ's operation; or two nodes will claim the same zone code, as in the case previously mentioned. Either way, the zones do allow for proper operation of the ZMQ algorithm.

4.3 Query Handling

Definition 4.2. *Forward links* are the parent-child links that connect root nodes of zones to root nodes of queries. While a node only has one zone link, it can have multiple forward links, one for each different query.

Forward links are created in the grouping-tree in response to queries being injected at root nodes – Section 4.3.2 presents these details. These links are used to transfer partially aggregated value of zones to the root nodes of queries during query processing. During query processing, forward links are all active, while zone links may be in an active or inactive state, i.e., some zone links may not be used to process some queries and thus they are not a part of the final routing topology.

Figures 10a and 10b show examples of active zone links, inactive zone links and forward links that are created for two queries injected at different root nodes – query *Q1* has query region [0, 75] × [50, 100], and query *Q2* has query region [25, 100] × [50, 100]. After the queries have been propagated into the network, a routing topology is created. The edges shown as dashed lines in Figure 10 illustrate the forward links, which form the paths from roots

of zones to root nodes of queries. The edges shown as dotted lines illustrate inactive zone links. Solid-line edges illustrate active zone links, which form the subtrees for queried zones.

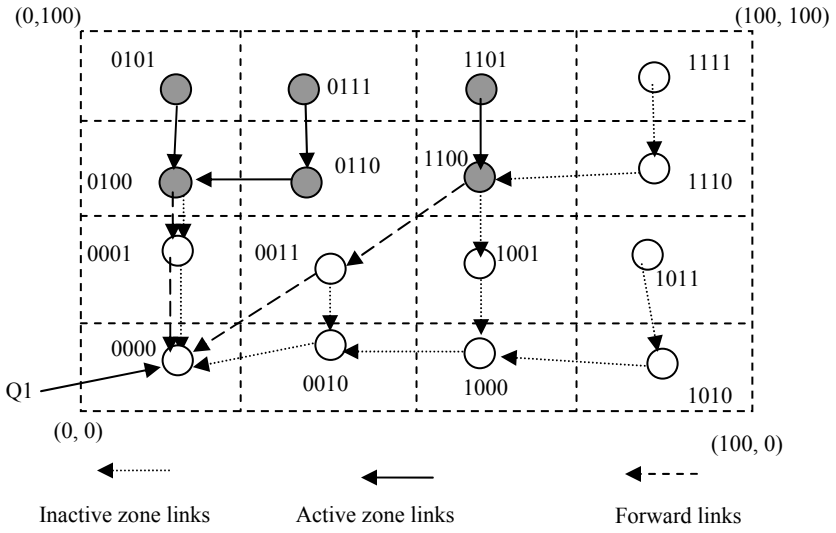

Figure 10a. Example routing structure for Query Q1.

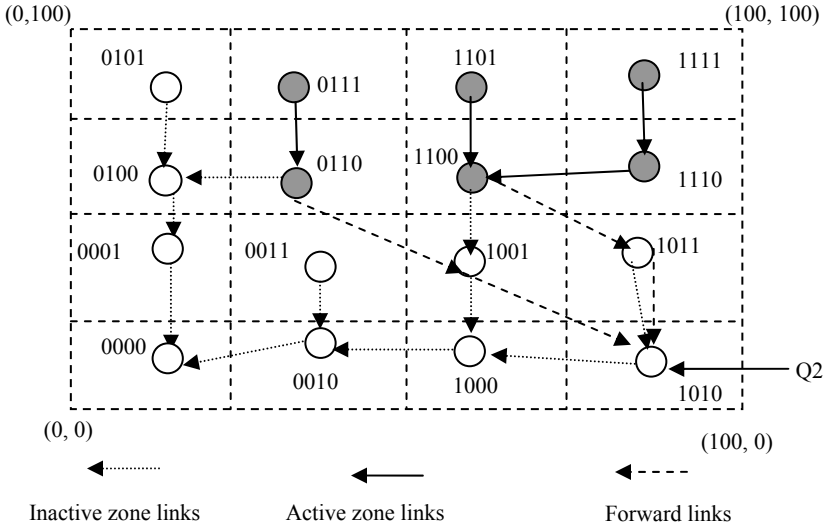

Figure 10b. Example routing structure for Query Q2.

4.3.1 Region Representation

For the sake of simplicity, we assume multiple queries have the same parameters. For example, all queries query the temperature (although for different regions in the sensor field). Let Q be a query over a region R. R can be represented by a set of zones. We can make the following observation:

Observation 1: A region R (of a query Q) can be *uniquely* represented by a set of zones $S = \{Z_1, Z_2, \ldots\}$, where (i) zones in S do not overlap with each other (i.e., no node in $Z_i \in S$ is in $Z_j \in S$, for $Z_i \neq Z_j$), and (ii) no two zones in S can be siblings.

To justify this observation, suppose R can be represented by two different zone sets $S = \{Z_1, Z_2, \ldots, Z_i\}$ and $T = \{Z_1', Z_2', \ldots, Z_i'\}$. As $S \neq T$, there must exist some node in R that is in Z_j ($\in S$) and also in Z_k' ($\in T$), and $Z_j \neq Z_k'$. From the tree structure of zones, we know that either Z_j contains Z_k' or Z_k' contains Z_j. Without loss of generality, suppose that Z_j contains Z_k'. As R includes zone Z_j, R includes all nodes in Z_j. In order to satisfy property (i) for set T, while including nodes in $Z_j - Z_k'$ in some zone in T, no parent zone of Z_k' can be included in T. Thus, a sibling zone of Z_k' must exist in zone set T in order to include all the nodes in R in the representation of T. This contradicts property (ii) that no sibling zones are in the set for a region. So any region R *must* be uniquely represented by a set of zones, and we call such a set of zones as R's "zone representation." For convenience, we also refer to this set of zones as Q's zone representation (i.e., the set of zones can represent both a region and a query that specifies that region).

Observation 1 is the basis for data sharing of intersection regions in our multi-root, multi-query processing. Once sensor nodes are deployed, they first use algorithm *BUILD_ZONE* to compute individual zone codes, and then use algorithm *GROUPING_ZONE* to group nodes in zones into subtrees and thus form a complete zone-link tree. This was the pre-setup process. Now we address the process of handling queries, which requires an algorithm for setting up paths from root nodes of queries to root nodes of zones for queries, and algorithms for data retrieval using the paths.

4.3.2 Routing and Data Retrieval

In the tree built by the algorithm *GROUPING_ZONE*, each zone can compute its partial aggregation result in the zone's root node in every sampling epoch. The processing for a query Q needs to set up paths from the root node of Q to each of the root nodes of the zones that belong to Q's zone representation.

4.3.2.1 Supporting Routines

The main algorithm *BUILD_ROUTING_TOPOLOGY* uses two Boolean functions *IsIn*(A, *Code*) and *IsRoot*(A, *Code*).

- *IsIn*(A, *Code*) determines if node A is in the zone represented by *Code*. The *Prefix* function, as introduced in Section 2.2.2, returns the common prefix string of two codes.

IsIn(*A, Code*)

1. // If *Code* is a prefix of *A*'s code, then A is in zone *Code*
2. If (*Prefix(code(Z_A), Code*) = *Code*) Return true
3.
4. If (*Prefix(code(Z_A), Code*) ≠ *code(Z_A)*) Return false // Zone *Code* is not a sub-zone of zone *code(Z_A)*
5.
6. // Zone *Code* is a sub-zone of *code(Z_A)*
7. // Split *code(Z_A)* to the length of zone *Code* to test if *A* is inside zone *Code*
8.
9. Repeat until *level(Z_A)* = length of *Code*
10. // Split as described in Section 2.2.2
11. Let *tempcode* = *code*(Z_A) after Z_A is split into two new zones
12.
13. // Now tempcode has the same length as *Code*
14. If (*tempcode* = *Code*) Return true
15. Else Return false

Examples: Let *A* be the sensor node such that *code(Z_A)* = 00. Consider three cases: 1) *A* is <u>not</u> in the zone represented by the zone code 01 (line 4 returns false). 2) *A* <u>is in</u> the zone represented by the zone code 0000, since line 11 executes twice – first, vertically splitting the original Z_A to create a new zone Z_A with code 000, and then horizontally splitting that zone Z_A to create yet another new zone Z_A with code 0000. In line 13, *tempcode* equals the *Code* value of 0000, so *IsIn* returns true. 3) *A* is <u>not</u> in the zone represented by the zone code 0001. Again, line 11 executes twice, resulting in the same ending zone Z_A with code 0000. But, in line 14, *tempcode* (0000) is not equal to *Code* (0001).

- *IsRoot*(*A*, *Code*) determines if node *A* is the root node of the zone represented by *Code*. The function *Left*(*a,b*) returns the portion of string *a* that remains after removal of the substring *Prefix(a,b)*. For example *Left(11011000, 11001001)* returns the string 11000, since *Prefix(11011000, 11001001)* returns string 110.

***IsRoot*(*A, Code*)**

1. If (*Prefix*(*code*(Z_A), *Code*) = *code*(Z_A)) // Zone *code(Z_A)* contains zone *Code*
2. If (*IsIn(code*(Z_A), *Code*)) // If *A* is inside zone *Code*, *A* is the root
3. Return true
4. // Otherwise, not a root node
5. Else Return false
6.
7. // Zone *Code* does not contain zone *code(Z_A)*
8. If (*Prefix*(*code*(Z_A), *Code*) ≠ *Code*) Return False
9.
10. // Zone *Code* contains zone *code(Z_A)*
11. *tmpcode* = *Left*(*code*(Z_A), *Code*)
12. // If *tmpcode* = 0…0, *A* is the root since *A* is the minimum code if all nodes in zone *Code* extend
    // to the same length as *A*'s code
13. // If *A* cannot detect another node in it's parent zone, excluding it's own zone (in the case
14. // of uneven distribution of nodes or the case of node failures), *A* is treated as a root node
15. If *(tmpcode* is a 0-string or *A* does not have zone link)
16. Return true
17. Else Return false

Examples: If *code*(Z_A) is 0010 and *Code* is 000, then *IsRoot*(*A*, *code*) is false. This is because line 1 is false and line 8 is true; the prefix does not match either code. If *code*(Z_A) is 0010 and *Code* is 001, then *IsRoot*(*A*, *code*) is true. This is because line 1 is false, line 8 is false, and *tmpcode* in line 11 is 0. If *code*(Z_A) is 0010 and *Code* is 00, then *IsRoot*(*A*, *code*) is false. This is because line 1 is false, line 8 is false, and *tmpcode* in line 11 is 10 ($\neq$ 00).

4.3.2.2 ZMQ Topology Construction

The objective of the main algorithm *BUILD_ROUTING_TOPOLOGY* is to construct routing tree topologies for data retrieval for queries. The algorithm implements two features: 1) It uses a special forward-link notification message, *FL_Notify*, to build forward links from root nodes of zones to root nodes of queries; 2) It changes inactive zone links to active zone links based on whether a node is in a zone that is a representing zone of a query.

We assume that each node in the network maintains a Neighbor Table recording status information of its neighbors, such as id, tree level, etc. Once a node receives a new query *Q* that has been injected into the network, this node sets its tree level (for the query *Q*) as 1 because it is the root node of this query. Then, this root node broadcasts a message called a *Query Broadcast* (QB) message, containing its own id, tree level, the query information, and the zone representation of the query. The broadcast is across one hop, hence only immediate neighbors of the sender will receive the QB. Upon receiving such a QB message from some node *Z*, a node *A* updates its own Neighbor Table, specifically the data about neighbor *Z* (including *Z*'s tree level for *Q*).

Each node *A* executes the *BUILD_ROUTING_TOPOLOGY* algorithm (specified below) after it receives a broadcast message for query *Q*. **T**he algorithm is executed independently for each query *Q* (this corresponds to line 1 in the algorithm). Consider any query *Q* for which some query broadcasts QB have been received. Node *A* first tests if it has already selected a routing-tree parent node (line 2) for this query. If node *A* has not selected such a parent node, *A* checks its Neighbor Table to find a neighbor node *M* with minimum tree level for query *Q*. *A* then selects *M* as its routing-tree parent node for query *Q***,** and sets its own tree level for *Q* as *M*'s tree level for *Q* plus 1 (lines 3, 4). *A* then broadcasts its id, tree level, and *Q*'s query information in a QB message using a 1-hop broadcast (line 5). Then, if *A* is a root node of a representing zone of query *Q* (lines 6 and 7), *A* sets *M* as its forward link parent for *Q*, and sends a *FL_Notify* message for *Q* to *M* (lines 8 and 9). Otherwise, if *A* is in a representing zone of the query but not a root node of that zone (line 12), *A* sets its own inactive zone link to now be an active zone link.

Consider a scenario in which node *A* has received a *FL_Notify* message for *Q* from its child node**.** This can happen if one of *A*'s child nodes is a root node of a zone representation for query *Q*, or one of *A*'s child nodes received a *FL_Notify* message for *Q* from one the child's children nodes. Node *A* constructs a forward link for *Q* by setting the routing-tree parent node as the forward-link parent node for *Q* and sending a *FL_Notify* message for *Q* to this forward link parent (lines 4-5 in FL_MESSAGE_HANDLING). If node *A* has not received the *FL_Notify* message for *Q* from any child, this implies that *A* might not be in a tree path from root nodes of zones to the root node of query *Q* and *A* might not need to be in the active state in the data retrieval phase.

The construction of forward links is considered backward compared to the construction of the tree structure because the tree structure is constructed from the root node of a query to root nodes of representing zones, but the forward links are constructed from root nodes of zones back to the root node of a query. The QB broadcast messages

diffuse from the root node of a query outwards into the network, while *FL_Notify* messages are first sent out by root nodes of zones.

Since the zone link of each node is unique (pre-setup by the *GROUPING_ZONE* algorithm), if two queries have a common zone in their zone sets, they both actually change the status of the same *zone links* in the zone from inactive to active. Hence, that subtree of the routing tree for both queries is common. Only the root node of that common zone may construct different *forward links* for different queries - the partially aggregated value of the zone will be sent separately by the root node of the zone to the different root nodes of queries along different paths. For the example in Figure 10a and 10b, the routing tree structures of *Q1* and *Q2* share the subtrees of zone 011 (includes nodes 0110 and 0111) and zone 110 (includes nodes 1100 and 1101).

Following is the detailed *BUILD_ROUTING_TOPOLOGY* algorithm and the FL_MESSAGE_HANDLING routine. For any node *A*, its *forward-link parent* is initially null.

```
BUILD_ROUTING_TOPOLOGY(A)   // Node A executes this algorithm upon receiving a query broadcast, QB
1.      For the query Q of the QB Do
2.      If (node A has not selected a routing-tree parent node for this query Q)
3.              From Neighbor Table, select node M with the min. tree level for Q as A's parent
4.              Set A's tree level for Q to 1 + M's tree level for Q
5.              Broadcast id, tree level, query Q and its representing zones
6.              If (there exists a code c in representing zones such that IsIn(A,c) = true)
7.                      If (IsRoot(A, c))          // A is a root of a representing zone for the query
8.                              Assign M as the forward-link parent of A for query Q
9.                              Send a FL_Notify message for Q to M   // This msg is processed by
10.                                                                   // FL_MESSAGE_HANDLING
11.                     Else
12.                             Set the zone link of A to active state
13.     End

FL_MESSAGE_HANDLING         // Node A executes this routine upon receiving a FL_Notify message
1.      For the FL_Notify message received for Q
2.      If forward-link is null
3.              // A identifies a parent node P, as FL_Notify is sent from child to parent node
4.              Assign parent node P to be the forward-link parent of A for query Q
5.              Send a FL_Notify message for Q to P
6.      End
```

Nodes having forward links or active zone links (as established by the above algorithm) would be in the active state, meaning that these nodes can transmit data messages during query processing. Other nodes may go to a sleep state to save energy. Nodes engaged in query processing do so by executing the following *DATA_RETRIEVAL* algorithm.

4.3.2.3 ZMQ Data Retrieval

DATA_RETRIEVAL(A) // *A* is the node executing the algorithm

1. If (*A* is a leaf node)
2. Send zone code and sensor value to parent node
3. Return
4. Aggregate data received from children based on zones // Merge smaller zones
5. While (there is a forward link for a query)
6. Send along the forward link the partially aggregated value for that query
7. If (there is an active zone link)
8. Send all partially aggregated values based on zones along the zone link
9. Return

Using Figure 9 as an example, node 1111 would send the tuple (1111, value) to its parent node 1110, while node 1110 would create an aggregated value for zone 111 and then send the tuple (111, aggregated-value) to its parent node 1100. For node 101, it receives (11, value) from node 1100. It cannot aggregate its zone 101 with zone 11, so it would send a two-tuple message ((101, its own value), (11, value)) to its parent node 100; and so on.

Returning to the case of an unevenly distributed network and the example discussed at the end of Section 4.2, we use Figure 11 to illustrate the formation of the routing tree structures. Recall that node 100 is absent in this example. The link types in Figure 11 are the same as in Figure 10 – dotted lines are inactive zone links, dashed lines are forward links, and solid lines are active zone links. Figure 11a shows the two trees that are pre-setup by the *BUILD_ZONE* algorithm, while Figure 11b shows the routing-tree structure generated by the *BUILD_ROUTING_TOPOLOGY* algorithm for a query Q, assuming that the query region corresponds to zone 10.

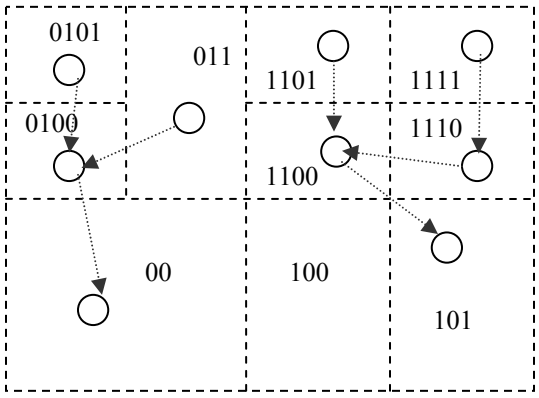

Figure 11a. Zone trees in ZMQ pre-setup.

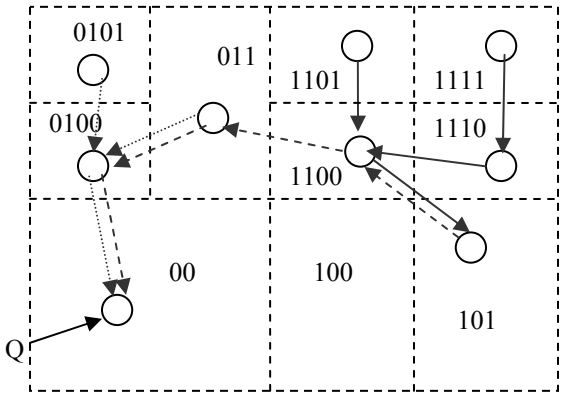

Figure 11b. Routing tree for query Q.

4.3.3 Query Completion Actions

Once a query is complete and the query data has been retreived, some clean-up actions are necessary to allow for subsequent query processing. In particular, special *query_destroy* messages are sent along the forward links back to the root nodes of each zone. Nodes along the path will remove the forward links for the completed query. In addition, the descendents of the root nodes of the zones will deactivate the zone-links if only this completed query (and no others) was active in the zone; otherwise, the zone-links are not deactivated.

4.4 Queries with Different Sampling Rates

In Section 2 we made the assumption, for convenience of presentation, that all queries use the same sampling rate. We now consider the more general case, where queries can have different sampling rates. Clearly, those nodes that must handle multiple queries, (i.e., nodes that are contained within overlapping zones) must sample at a frequency that is sufficient to meet the sampling rate requirement of the query that has the highest sampling rate. Other "non-overlapping" nodes can simply use the sampling frequency associated with the query whose query region contains the nodes. Therefore, only overlapped zone links transmit data at the highest frequency; forward links will use the query frequency that is established when the forward link is setup. Note that even in the case of multiple queries with different sampling rates, the zone-based multi-query processing method will still perform better than the naïve multiple query processing method. This is because the zone-based technique does not increase the sampling and transmission frequency at any node compared to the frequency of that node in the naïve query processing method.

5. EXPERIMENTAL EVALUATION

We performed simulation experiments to compare and analyze the three multi-root, multi-query processing algorithms; the naive multi-query processing algorithm (NMQ) (Section 3.1), the static multi-query processing algorithm (SMQ) (Section 3.2), and the zone based multi-query processing algorithm (ZMQ) (Section 3.3).

- NMQ constructs a different routing tree for each query, thereby using a different tree for each query region.
- ZMQ is implemented based on the details in Section 4.
- Based on the position data for ZMQ, we can compute the intersection regions for SMQ. Then the grouping technique is used to group nodes in the same intersection regions together in the same subtree to compute the results for SMQ. As SMQ assumes that the multiple query regions are known before the query processing, it forms one subtree for each intersection region. For example, if *Q1* is over region *R1*, *Q2* is over region *R2*, SMQ forms one subtree for $R1 - R1 \cap R2$, one for $R2 - R1 \cap R2$ and one for $R1 \cap R2$, and then collects information through the root nodes of these three subtrees.

We use the SUM queries that compute the sum of all readings of the sensors in a rectangular area in our experiments. The following is an example query for the sum of sensed temperatures in the area $[10, 20] \times [50, 70]$ every 10 seconds for one hour:

SELECT SUM(sensor.temp) FROM sensorDB WHERE (sensor.x ≥ 10 AND sensor.x ≤ 20) AND (sensor.y ≥ 50 AND sensor.y ≤ 70) DURATION 1 hour EVERY 10 seconds.

For simplicity, all queries have the same duration and same epoch, but have different query areas. The configuration of our experiments is as follows:

Deployment:

We use a 256×256 cell matrix, where a sensor can be placed at the center of a cell. The length of each side of a cell is 1. Each node, except the nodes in the border cells, can communicate directly with its eight direct neighbors in the matrix. By default, each cell has a sensor placed in it.

Input parameters:

N: The number of queries (each query defines a query region).

D: The network density, defined as the number of sensors per cell of the sensor field matrix. To be consistent with the system model, the highest value of D is 1. This is also the default value.

QR: The query region representing a query.

OP: Overlap percentage, the percentage of the nodes inside intersection regions over all nodes in query regions. This is defined as follows:

$(\sum_{I}(sizeof(I)*(numberof(I)-1))/\sum_{Q}sizeof(Q))$ * (N/(N-1)), where

- ***I*** is an *intersection region*, defined as the largest region in which all the nodes are queried by the same set of queries.
- *sizeof*(*I*) is the number of nodes in the intersection region *I*,
- *numberof*(*I*) is the number of queries that each node in *I* receives,
- *N* is the total number of query regions, and
- (N/(N-1)) is a scale factor for normalization.

The intuition behind the definition of overlap percentage is as follows. In any region, the number of nodes times the number of queries in excess of one that the nodes handle represents the net excess overlapping of queried nodes in that region. The numerator of the OP value is the summation across all regions of this net excess overlapping of queried nodes. This is divided by the total number of queried nodes, considered independently across all queries. Thus, the OP value provides the fraction of queried nodes (across all queries, considered independently) that overlap across the various queries. This fraction is multiplied by (N/(N-1)) as a normalizing factor so that the value of OP ranges from 0 to 1.

Example: An example of the overlap percentage (OP) is shown in Figure 12.

sizeof(*QR1*∩*QR2*∩*QR3*) = 4 and *numberof*(*QR1*∩*QR2*∩*QR3*) = 3.

sizeof(*QR1*∩*QR3*-*QR1*∩*QR2*∩*QR3*) = 6 and *numberof*(*QR1*∩*QR3*-*QR1*∩*QR2*∩*QR3*)= 2.

sizeof(*QR1*∩*QR2*-*QR1*∩*QR2*∩*QR3*) = 5 and *numberof*(*QR1*∩*QR2*-*QR1*∩*QR2*∩*QR3*)= 2.

sizeof(*QR2*∩*QR3*-*QR1*∩*QR2*∩*QR3*) =2, and *numberof*(*QR2*∩*QR3*-*QR1*∩*QR2*∩*QR3*) =2.

sizeof(Q1) is 30, the *sizeof(Q2)* is 24 and the *sizeof(Q3)* is 24.

OP = (4×(3-1) + 6×1 + 5×1 + 2×1)/(5×6 + 4×6 + 4×6)) × (3/(3-1)) = (31/78) ×1.5 =60%.

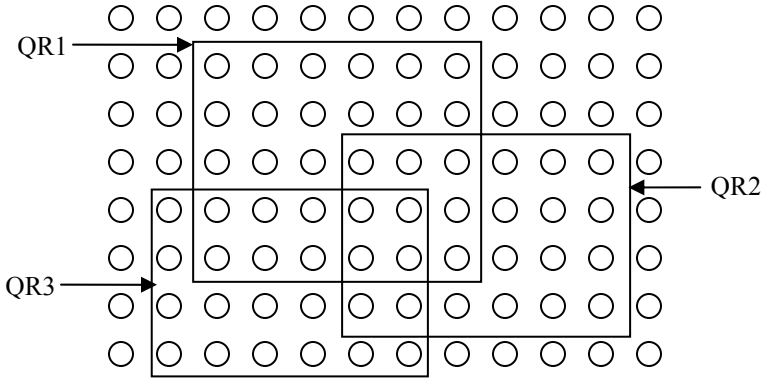

Figure 12. An example of Overlap Percentage.

Metric: Average number of messages (*ANM*) per node per epoch, which is defined as the total number of messages used in each retrieving epoch divided by the total number of nodes included in the query regions. Nodes in intersection regions are counted separately for each query, i.e., nodes in intersection regions are counted multiple times. This is because we want to use *ANM* as a metric to capture the extent of sharing of the readings. Smaller *ANM* means more sharing inside the network. However, the ANM may be more than 1 because we count all messages transmitted inside the network. Thus, while we only count nodes inside query regions, there are overhead messages from root nodes of the query regions to root nodes of queries. For NMQ, *ANM* is always greater than 1. Formally,

ANM = (total number of messages) / ($\sum_{Q} sizeof(Q)$)

In the example of Figure 11, if the total number of messages transmitted in the network is T, then $ANM = T / (5\times6 + 4\times6 + 4\times6) = T/78$.

The query region (QR) of each query is represented by a rectangle, for example 64×98 represents a query region of width 64 and length 98 units. Given all 4 input parameters, we perform an initial simulation setup step which computes random positions for the variously sized query regions and for the root nodes of each query. Those query region locations that fulfill the input conditions on the overlapping percentage (OP) are then used to drive the simulation step. To randomly place one of the fixed-size query regions, we randomly select a coordinate position within the simulated sensor field cell matrix and use this to define the top-left-most corner of the query region.

In practice, sensor nodes and query regions are likely to follow some type of clustering, following the boundaries of buildings, directions of roads, etc. Thus, it is possible to consider tuning of the ZMQ algorithm for specific application environments so as to take advantage of known or predicted boundaries in the topology and of workload when setting up zones. Since our evaluations are based on random assignment simulations, the analysis represents a "worse-case" type scenario. The algorithms would likely perform better if they were tuned for specific conditions.

5.1 Impact of Overlap Percentage (OP)

In this experiment, we show the impact of the overlap percentage on the performance of the three algorithms. We hold constant the input parameters *QR* and *N*, but vary the position of different query regions to change the overlap percentage for all queries. The results for three different sets of *QR* and *N* are shown in Figure 13.

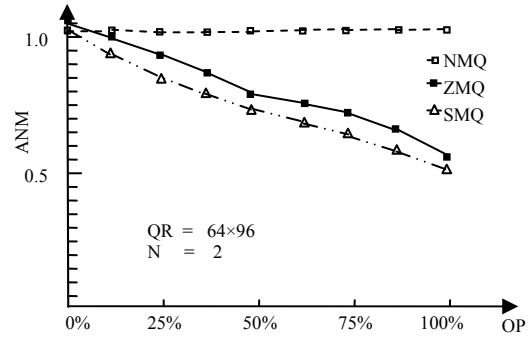

Figure 13a. Impact of OP on the average number of messages.

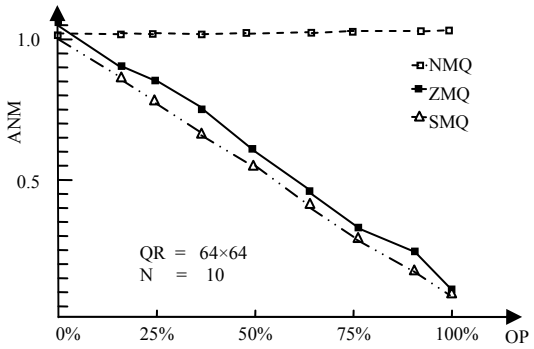

Figure 13b. Impact of OP on the average number of messages.

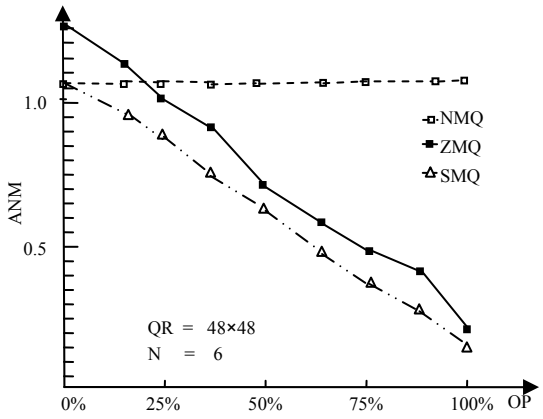

Figure 13c. Impact of OP on the average number of messages.

Observe that ZMQ outperforms NMQ as the percentage of overlapping increases, and ZMQ has nearly the same performance as SMQ.

The reason that the total number of messages used by NMQ does not change significantly is because different tree structures are set up for each of the different queries, no reading and transmissions are shared, and the change of positions of query regions does not significantly affect the total number of messages transferred in the network. The reason that the number of messages used by ZMQ and SMQ decreases is because these algorithms share the readings of sensor nodes in the intersection regions. As the overlapping increases, more sharing is possible, and fewer messages are needed.

Notice that ZMQ uses more messages than NMQ when there is no overlap between the two query regions. This is because ZMQ slightly decreases the amount of data aggregation that can be achieved and increases the number of messages needed if a query region contains more than one pre-setup zones. However, we can also see from Figure 12 that if there is an overlap between the query regions, the number of messages reduced by sharing readings and transmissions exceeds the number of messages increased due to the decrease in data aggregation. We refer to the

ability to exploit potential data aggregation as *aggregation extent.* So, aggregation extent represents how good a routing topology is at exploiting sharing of partially aggregated information. Naturally, high aggregation extent is desired.

The difference between ZMQ and SMQ is because each intersection region in ZMQ may consist of more than one zone, i.e., more than one subtree, while each intersection region in SMQ consists of just one subtree. The aggregation extent for intersection regions for ZMQ is slightly less than the aggregation extent for SMQ. This causes ZMQ to use some more messages than SMQ.

5.2 Impact of Network Density (D)

In this experiment, we show the effect of the density of the sensor network on the performance of the three algorithms. We hold *QR*, *N,* and *OP* fixed, and vary the density of the sensor field. The average number of messages transmitted, as a function of density, for different *QR*, *N*, and *OP* is shown in Figure 14. The density in Figure 14 is computed by the number of sensor nodes divided by the size of the sensor field. For example, given the fixed 256 × 256 sensor field, the density 1/4 in Figure 14 means that there are 128 × 128 sensor nodes evenly spread in the sensor field.

Observe that for all algorithms, as density decreases, the *ANM* increases for all settings of *OP, N,* and *QR*. This is because when density decreases, the nodes become further apart, and more messages are needed to collect the nodes' readings. This can also be seen from the definition of *ANM*, viz., (total number of messages) / ($\sum_{Q} sizeof(Q)$). As *D* decreases, the denominator decreases proportionately to *D*, but the numerator does not change as rapidly.

We can observe in Figure 14(a,b,c) that even if the query regions have significant overlap, ZMQ may perform worse than NMQ when the density is sufficiently low. The reason is that as the density becomes lower, the number of nodes in the intersection area decreases, and the data and transmission sharing also decreases. At some threshold, the number of messages added because of the decrease in aggregation extent may exceed the number of messages decreased by data sharing.

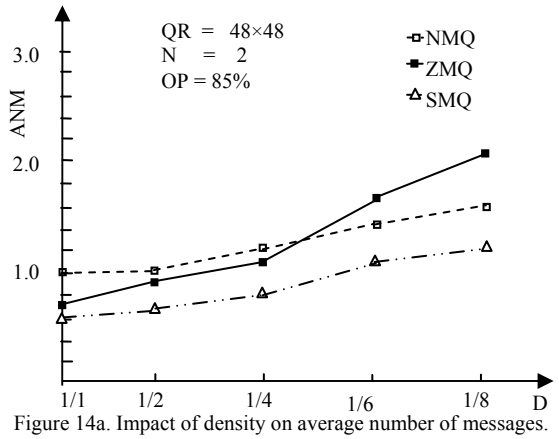

Figure 14a. Impact of density on average number of messages.

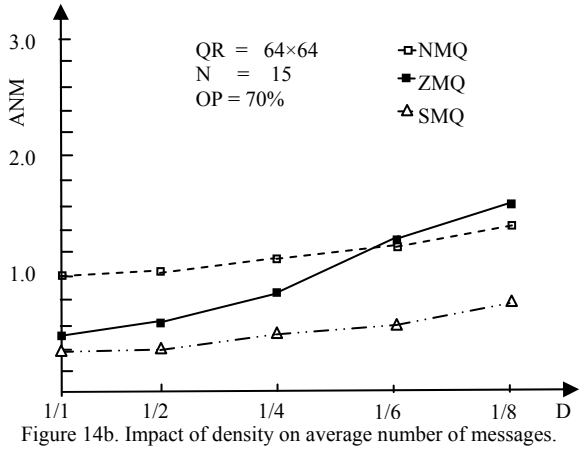

Figure 14b. Impact of density on average number of messages.

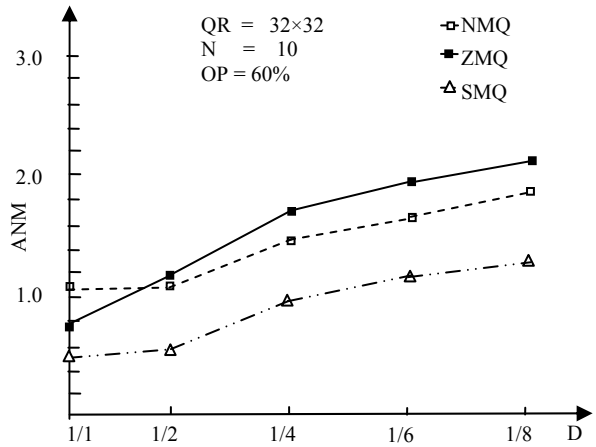

Figure 14c. Impact of density on average number of messages.

5.3 Impact of Query Region Size (QR)

Figure 15 shows the relationship between size of query regions and the average number of messages transmitted in each epoch for the three algorithms. Observe that as *QR* increases, the *ANM* (=total number of messages / ($\sum_{Q} sizeof(Q)$)) for NMQ approaches 1. Also, as *QR* increases, *ANM* of ZMQ approaches the *ANM* of SMQ. This implies that ZMQ performs better when the sizes of query regions are large.

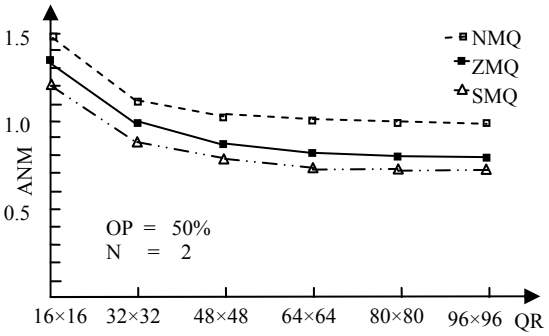

Figure 15a. Effect of QR on three algorithms.

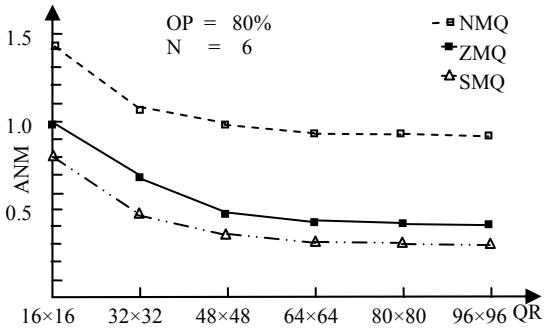

Figure 15b. Effect of QR on three algorithms.

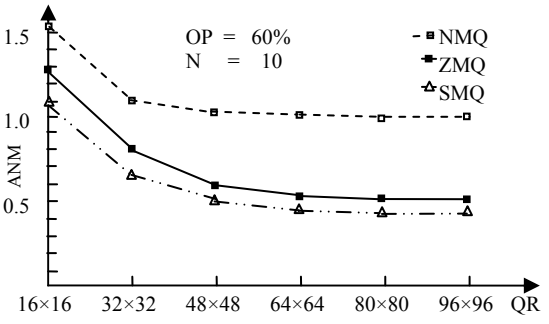

Figure 15c. Effect of QR on three algorithms.

Figure 15 also shows that for all the algorithms, as the size of the query region increases, the average number of messages for each node first decreases and then becomes stable. This is because of two factors.

- Consider the ratio X:Y, where X is the number of messages used from the root nodes of each region to the root nodes of queries, and Y is the total number of sensor nodes in the query regions. This ratio represents an amortized overhead to reach the query root from the region roots. As QR increases, the numerator tends to decrease somewhat, and the denominator increases; thereby decreasing this overhead. As QR continues to increase, the value of this overhead becomes relatively small.
- With increasing QR, the predominant factor becomes the degree of node overlap and the sharing of sensor readings among regions. As QR increases, this also tends to have a saturation effect. In our example, this sets in around regions of size 48x48.

As expected, ZMQ performs better than NMQ, irrespective of the size of the query region assuming that the overlap percentage is not very low.

5.4 Impact of Number of Queries (N) Using Controlled Overlap Percentage

This experiment studies the effect of the number of queries on the performance of the three algorithms. We set *OP* and *QR,* and vary the number of injected queries. Figure 16 shows the results of the experiments. Observe that as the number of queries increases, the *ANM* for SMQ and ZMQ decreases, but remains almost stable for NMQ.

We also can find the answer from the definition of *ANM* as follows. Let:

- *S* be the number of nodes that can share reading and transmission,
- *SN* be the total number of sensor nodes in query regions (i.e., $\sum_{Q} sizeof(Q)$),
- *E* be the extra messages needed to send data from root nodes of subtrees to root nodes of queries.

ANM (=total number of messages / ($\sum_{Q} sizeof(Q)$) = *(SN –S +E)/SN* = 1- *S*/*SN* + *E*/*SN*. For NMQ, *S* is always 0, so *ANM* of NMQ is always more than 1 and decreases as *SN* increases. As the number of queries increases, *SN* and *S* both increase, while *E* stays almost stable. Therefore, the overhead for SMQ and ZMQ approaches the value 1-*S*/*SN* as the number of queries increases.

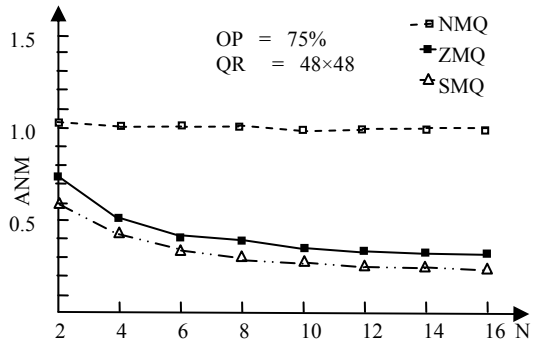

Figure 16a. Effect of N on three methods with fixed OP.

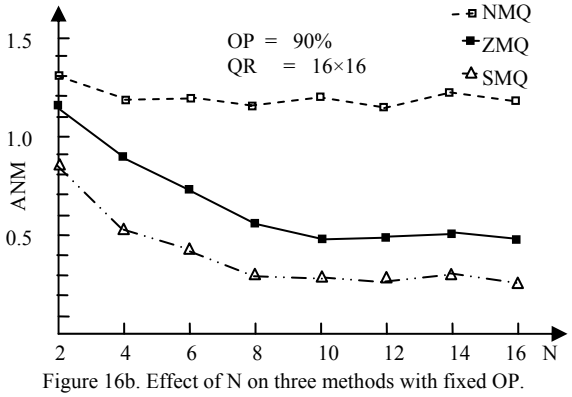

Figure 16b. Effect of N on three methods with fixed OP.

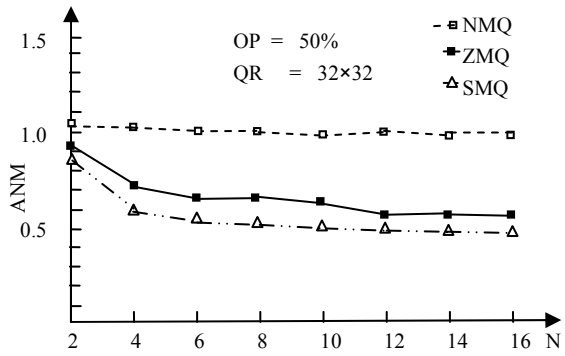

Figure 16c. Effect of the N on three methods with fixed OP.

5.5 Impact of Number of Queries (N) Using Uncontrolled Overlap Percentage

The previous experiment studied the impact of varying the number of queries, while controlling the overlap percentage. That was an "idealistic" experiment in the sense that it does not reflect the reality that overlap percentage is dictated by the queries and cannot be controlled.

Hence, in this experiment, we study the impact of the number of queries without controlling the overlap percentage. We compute the *ANM* for the three algorithms. This experiment assumes that the size of each query region is fixed. The results are shown in Figure 17, assuming a 48x48 query region. Observe that as the number of queries increases, more intersection occurs. This causes the reduction of the average number of messages transmitted by active sensor nodes. As the number of queries increases, ZMQ follows the decreasing curve of SMQ, while the overhead for NMQ approaches 1.

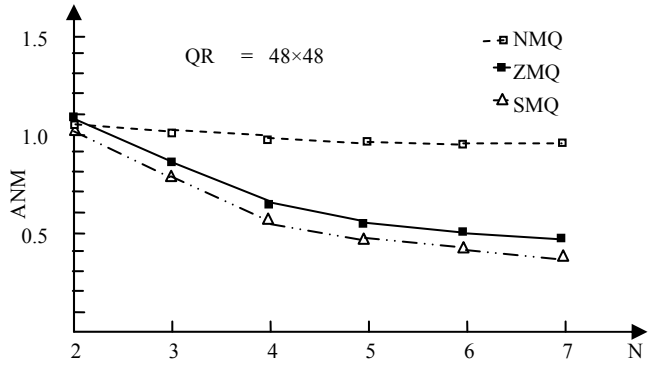

Figure 17. Effect of the N on our method without fixed OP.

5.6 Comparison Summary

From the above simulation experiments, we measured and compared the performance of the zone-based algorithm (ZMQ), the naïve algorithm (NMQ), and the static algorithm (SMQ). From the results, we can see that

- SMQ always performs best, as expected.
- ZMQ performs generally better than NMQ when the sizes of the query regions are big, the density of the sensor field is high, and there are large overlaps among queries. ZMQ performs increasingly better than NMQ as the sizes of query regions become larger, the sensor field density increases, and the overlap among query regions increases.

The following table compares the performance and tradeoffs involved:

	Dynamic queries	Aggregation extent	Energy efficiency	Time complexity	Space complexity	Initiali-zation	Latency
NMQ	Yes	Very good	Good	O(*n*)	O(*n*)	No	Small
ZMQ	Yes	Good	Very good	O(*n*)	O(*n*)	Yes, O(*n*)	Moderate
SMQ	No	Very good	Best	O(*n*)	O(*n*)	No	Small

Dynamic queries: NMQ and ZMQ can perform dynamic query processing, i.e., process queries on the fly. NMQ sets up a separate tree structure for each query. Therefore, NMQ can dynamically set up a tree structure during query processing when a new query is injected into the sensor network. After ZMQ sets up the whole zone-link tree, it only needs to set up the tree paths from the root nodes of a new query to the root nodes of the representing zones of the query. Hence, ZMQ can also dynamically handle new queries in the middle of processing. Because SMQ can only work when it knows all regions before processing, it cannot dynamically handle new queries during its processing.

Aggregation extent: NMQ groups all nodes in a query region in one subtree and SMQ groups nodes in one region (including intersection regions) in one subtree. They both have very good aggregation extent. As ZMQ uses pre-setup subtrees for zones, an intersection region may contain more than one zone. So ZMQ has a good aggregation extent but not as good as that of NMQ and SMQ.

Energy efficiency: SMQ is always the most energy efficient algorithm for multi-root, multi-query processing, and ZMQ performs better than NMQ in most situations.

Data retrieval algorithm overhead (time complexity and space complexity): Since all three algorithms require a node to process results only from its children nodes, which can be at most all of the eight neighbors, all three algorithms have O(*1*) (constant) time complexity and space complexity at each node in one epoch during the data retrieval phase. The whole network would then have time complexity of O(*n*) for all three algorithms since in the worst case the depth of a tree path can be O(*n*), where *n* is the number of nodes in the sensor field. While each node takes a constant processing time, nodes on one tree path must process serially. Similarly, for space complexity, the total is O(*n*), with each node requiring O(*1*) space.

Initialization: NMQ and SMQ do not need to initialize the network, while ZMQ needs to pre-setup the zone-link tree structure.

For the *BUILD_ZONE* algorithm for ZMQ, the time complexity is O(log *n*). To see this, suppose nodes are evenly distributed in a $\sqrt{n} * \sqrt{n}$ space, so the size of each node's zone is one unit, the whole field is *n* units. In each iteration, the space is divided into two equal-sized subspaces. Hence, the algorithm needs log *n* time to terminate when each zone is split into a space of size one unit. So the average time complexity of the *BUILD_ZONE* algorithm is O(log *n*), with the worst case being O(*n*), when all nodes line up.

The time complexity for the *GROUPING_ZONE* algorithm is O(log *n*), because each iteration in the algorithm considers a zone with a size that is two times the size of the previously considered zone. Since the size of the zone of each of *n* nodes in a $\sqrt{n} * \sqrt{n}$ space is $1/n$, the algorithm would terminate when it reaches the zone represented by the empty-string zone code, which occurs in log *n* iterations, worst case.

Since a node only needs to broadcast a message containing its information to its neighbor nodes *after* it changes its zone code, the average additional energy for *BUILD_ZONE* is log *n* for the whole life of the network. This is because a node only needs to split at most log *n* times. In GROUPING_ZONE, nodes do not need to broadcast any messages to select a parent node. Thus, the total communication overhead for ZMQ is log *n* messages per node over the life of the network, for a total overhead of *n* log *n*.

From the above analysis, we can see that ZMQ has some initialization overhead in comparison to NMQ and SMQ, but those overheads are relatively small.

Latency: The depth of the routing tree for ZMQ is O($2\sqrt{n}$), while the depth of the routing tree for NMQ and SMQ is O($\sqrt{n}$). This is because the length of the diagonal of a $\sqrt{n} * \sqrt{n}$ sized sensor field is $\sqrt{n}$, while the step paths from (0, 0) to ($\sqrt{n}$, $\sqrt{n}$) is $2\sqrt{n}$. In the worst case, the depth of all three routing trees is O(*n*), when all nodes line up.

Recommendation: The most applicable situation for ZMQ is when there are big query regions, big overlap among queries, and high network density. Otherwise, if the application requires dynamic query processing, NMQ is a good algorithm. If the queries are known a priori (before any processing), and no new queries come in during processing, SMQ is the best algorithm.

6. CONCLUSION AND FUTURE RESEARCH

This paper identified and addressed, for the first time, multi-root, multi-query optimization for long duration aggregation queries over regions. Three algorithms were presented – a naïve algorithm (NMQ), a static algorithm (SMQ) and a heuristic algorithm (ZMQ). SMQ and ZMQ are based on the sharing of partially aggregated results from overlapping query regions. We performed a detailed comparison and analysis of the three algorithms. Simulation experiments indicate that the data-sharing algorithms provide significant energy savings under a wide range of network conditions and query region options.

The SMQ method, which requires that all query regions be known before processing, is the best algorithm in terms of energy saving, but it is not practical. NMQ is the simplest algorithm, but performs worst in most situations. The ZMQ algorithm performs quite well when there are large-size query regions and high overlap percentage. The performance of ZMQ almost approaches the performance of SMQ. The experiments lead to the conclusion that ZMQ is a practical and energy efficient algorithm for multi-root, multi-query processing.

The following are some interesting problems for further research. It will be useful to examine the case where query regions are not static, such as might be the case for moving regions associated with mobile objects [9]. Another problem is the effect of packet loss. For example, packet loss in the ZMQ technique will affect multiple queries, instead of only one query as in the case of NMQ. To evaluate the impact of network-layer properties on the proposed query processing approach, it will be useful to consider radio-models and MAC layer issues. These might suggest routing trees that are different from those that are optimal for pure query-informed routing, as was studied in this paper. Such multi-dimensional optimization of routing trees will require new heuristics and a careful balance of trade-offs. Another interesting problem is the optimization of heterogeneous multiple queries. This entails exploring sensor-reading and transmission-sharing where queries specify different query fields. One can also study the multi-

query situation when there are "holes" in sensor networks, such that sensor nodes are not evenly distributed. This situation can impact the zone grouping and routing topology construction.

In this paper, we considered geographic information as the sole criteria for zone setup. In practice, information such as network partition in space, and data correlation inside geographical boundaries, can help increase data compression or aggregation. Future work can seek to exploit such information in zone construction.

To simplify the discussion, we did not address the issue of sensor node failures. But, since failure of nodes in a sensor network may be quite common, it is useful to consider the impact of such failures on the ZMQ algorithm and at least suggest how the algorithm can be robust to these dynamic failures. Periodic running of the BUILD_ROUTING_TOPOLOGY is the key for such failure handling of ZMQ. Since ZMQ has two type of links in the routing tree, forward links and zone links, there are two types of node failures to consider: 1) the failure of a node that is in a forward link; and 2) and the failure of a node that is in a zone link.

First, consider node failure in a forward link. Since we use a shortest path algorithm to build forwards links, the periodic running of BUILD_ROUTING_TOPOLOGY in a child node of the failed node will first detect the parent node's failure in its Neighbor Table. That child node can then use a shortest path approach to select a node from among its live neighbors as a new parent node, and send a FL_notify message to that selected node. If the new parent node has not set up a forward link path to the root node of the query, this message will trigger the parent node to do so; if the new parent node is already in the routing topology of the query, then the node does not need to do any thing. Different child nodes may select different parent nodes – this does not affect the running of the algorithm, but may slightly increase the overhead for data retrieval. Maintaining information that indicates if a neighbor is already in a forward link path can help improve performance during the process of deciding on a parent node from several same-tree-level neighbor nodes. It does not appear that this will have a significant impact on the performance of the query processing algorithms discussed in this research.

Second, consider node failure in a zone link. In this situation, a child node of the failed node can first try to select as a new parent node a neighbor node that is in its same zone or the immediate parent zone, and with a smaller zone code. If there is such a new parent node, the child can switch to that node without trouble. If there is not such a node, the child can try to identify a similar potential parent node from within the child's immediate parent zone's parent zone, and so on. If it can select such a node, then the child node can switch to that selected node without affecting its children nodes. However, if it cannot select such a parent node, it will select from its neighbors one node that is also in the query region – to allow for aggregation. If the child still cannot select a parent based on the just identified process, then it will select any one of the nodes from its Neighbor Table and use a shortest path method to build a reverse path from itself to the root node by using BUILD_ROUTING_TOPOLOGY. In the very unlikely worst case, where each child of the failed node needs to build a forward path from itself to query root-nodes, the aggregation extent will be reduced, and more nodes need to be involved in data transmission. Further investigation on the impact of worst case node failure situations on the efficiency of the ZMQ method remains as future research.

In the case of sensor node failures, if a node does not have a neighbor that has a smaller tree level in its Neighbor Table when the failure happens in a forward link, or if a node has to rebuild a shortest path from itself to the root

node of a query when failure happens in a zone link, data will be lost for the subtree rooted at the node until the rebuilding is finished. For this situation, it will be useful to investigate integration of robust aggregation in sensor networks [15] into the ZMQ method.

REFERENCES

[1] A. Boulis, S. Ganeriwal and M. B. Srivastava, "Aggregation in Sensor Networks: an Energy-Accuracy Trade-off," *Ad Hoc Networks*, Vol. 1, Issues 2-3, September 2003, pp. 317-331.

[2] A. Cerpa and D. Estrin, "ASCENT: Adaptive Self-configuring Sensor Networks Topologies," *IEEE INFOCOM*, June 2002.

[3] J. Considine, F. Li, G. Kollios and J. Byers, "Approximate Aggregation Techniques for Sensor Databases," *IEEE Int. Conf. on Data Engineering*, 2004.

[4] K. Dasgupta, K. Kalpakis and P. Namjoshi, "Improving the Lifetime of Sensor Networks via Intelligent Selection of Data Aggregation Trees," *Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2003.

[5] A. Durresi, V. Paruchuri, S. S. Iyengar and R. Kannan, "Optimized Broadcast Protocol for Sensor Networks," *IEEE Transactions on Computers*, Vol. 54, No. 8, Aug. 2005, pp. 1013 – 1024.

[6] F. Emekci, H. Yu, D. Agrawal and A. E. Abbadi, "Energy-Conscious Data Aggregation Over Large-Scale Sensor Networks," *UCSB TechnicalReport*, 2003.

[7] D. Estrin, M. B. Srivastava and A. Sayeed, "Tutorial on Wireless Sensor Networks," *ACM International Conference on Mobile Computing and Networking (MOBICOM)*, 2002.

[8] L. M. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *IEEE INFOCOM*, April 2001.

[9] B. Gedik, K-L. Wu, P. S. Yu and L. Liu, "Processing Moving Queries over Moving Objects Using Motion-Adaptive Indexes, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 5, 2006, pp. 651 – 668.

[10] H. Gupta, V. Navda, S. R. Das and V. Chowdhary, "Efficient Gathering of Correlated Data in Sensor Networks," *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05),* 2005.

[11] B. Howe and D. Maier, "Algebraic Manipulation of Scientific Datasets, " *The VLDB Journal*, 14(4), November 2005, pp. 397-416.

[12] X. Li, Y. J. Kim, R. Govindan and W. Hong, "Multi-dimensional Range Queries in Sensor Networks," *ACM Conference on Embedded Networked Sensor Systems (Sensys'03)*, 2003, pp. 63 – 75.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong, "Tag: A Tiny Aggregation Service for Ad-hoc Sensor Networks,"*5th Symposium on Operating Systems Design and Implementation*, 2002, pp. 131 – 146.

[14] S. Madden and M. Franklin, "Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data," *IEEE International Conference on Data Engineering* (*ICDE*), 2002.

[15] S. Nath, P. B. Gibbons, S. Seshan and Z. R. Anderson, "Synopsis Diffusion for Robust Aggregation in Sensor Networks," *ACM Conference on Embedded Networked Sensor Systems (Sensys'04)*, 2004.

[16] S. Prabh and T. Abdelzaher, "Energy-Conserving Data Cache Placement in Sensor Networks," *ACM Transactions on Sensor Networks*," Vol. 1, No. 2, Nov. 2005.

[17] N. Sadagopan, B. Krishnamachari and A. Helmy, "Active Query Forwarding in Sensor Networks," *Ad Hoc Networks*, Vol. 3, Issue 1, January 2005, pp. 91-113.

[18] A. Savvides, C.-C. Han and M. B. Srivastava, "Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors," *ACM SIGMOBILE*, 2001.

[19] M. Sharaf, J. Beaver, A. Labrinidis and P. Chrysanthis, "Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks, *The VLDB Journal*, 13(4), 2004, pp. 384-403.

[20] N. Trigoni, Y. Yao, A. Demers, J. Gehrke and R. Rajaraman, "Multi-Query Optimization for Sensor Networks," *International Conference on Distributed Computing on Sensor Systems (DCOSS)*, pp. 307 – 321, 2005.

[21] C. Wang and L. Xiao, "Sensor Localization in Concave Environments," *ACM Tran. on Sensor Networks,* Vol. 4, No. 1 (Jan. 2008), pp. 1-31.

[22] Q. Wu, N. S. V. Rao, J. Barhen, S. S. Iyengar, V. K. Vaishnavi, H. Qi and K. Chakrabarty, "On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 6, June 2004, pp. 740 – 753.

[23] L. Xiao and A. M. Ouksel, "Tolerance of Localization Imprecision in Efficiently Managing Mobile Sensor Databases," *ACM International Workshop on Data Engineering for Wireless and Mobile Access (MobiDE),* 2005.

[24] F. Ye, H. Luo, J. Cheng, S. Lu and L. Zhang, "A Two-Tier Data Dissemination Model for Large-Scale Wireless Sensor Networks," *ACM International Conference on Mobile Computing and Networking (MOBICOM)*, 2002.

[25] Y. Yao and J. Gehrke, "The Cougar Approach to In-network Query Processing in Sensor Networks," *ACM Special Interest Group on Management of Data (SIGMOD) Record*, Vol. 31, No. 3, 2002, pp. 9 – 18.

[26] M. Younis, M. Youssef and K. Arisha, "Energy-aware Routing in Cluster-Based Sensor Networks," *10th IEEE MASCOTS*, October 2002.

[27] Z. Zhang and S. M. Shatz, "A Technique for Power-Aware Query-Informed Routing in Support of Long-Duration Queries for Sensor Networks," *International Conference on Sensing, Networking and Control (ICNSC06)*, 2006.
[28] R. Zheng, J. C. Hou and L. Sha, "Asynchronous Wakeup for Ad Hoc Networks," *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, 2003.