Multi-query optimization for sensor networks

N. Trigoni
Birkbeck, Un. of London

Y. Yao A. Demers J. Gehrke
Cornell University

R. Rajaraman
Northeastern University

Abstract

The widespread dissemination of small-scale sensor nodes has sparked interest in a powerful new database abstraction for sensor networks: Clients "program" the sensors through queries in a high-level declarative language (such as a variant of SQL) permitting the system to perform the low-level optimizations necessary for energy-efficient query processing. In this paper we consider multi-query optimization for aggregate queries on sensor networks. We develop a set of distributed algorithms for processing multiple queries that incur minimum communication while observing the computational limitations of the sensor nodes. Our algorithms support incremental changes to the set of active queries and allow for local repairs to routes in response to node failures. A thorough experimental analysis shows that our approach results in significant energy savings, compared to previous work.

I. Introduction

Wireless sensor networks consisting of small nodes with sensing, computation and communication capabilities will soon be ubiquitous. Such networks have resource constraints on communication, computation, and energy consumption. First, the bandwidth of wireless links connecting sensor nodes is usually limited, on the order of a few hundred Kbps, and the wireless network that connects the sensors provides only limited quality of service, with variable latency and dropped packets. Second, sensor nodes have limited computing power and memory sizes that restrict the types of data processing algorithms that can be deployed. Third, wireless sensors have limited supply of energy, and thus energy conservation is a major system design consideration. Recently, a database approach to programming sensor networks has gained interest [1], [2], [3], [4], [5], [6], [7], where the sensors are programmed through declarative queries in a variant of SQL. Since energy is a highly valuable resource and communication consumes most of the available power of a sensor network, recent research has focused on devising query processing strategies, like in-network aggregation, that reduce the amount of data propagated in the network.

Our Model and Assumptions. We assume that nodes are stationary and battery-powered, and thus severely energy constrained. We distinguish a special type of node, referred to as a *gateway* node, where users inject query requests. The raw data generated at a sensor node is produced by one or more attached physical sensors. Sensor nodes generate new readings either at regular intervals (such as polling a temperature sensor every minute) or irregularly (such as a motion sensor detecting movement). The sensor network is programmed through declarative queries posed in a variant of SQL or an event-based language [1], [2], [3], [4], [5]. We concentrate on aggregation queries, and the sensor network performs in-network aggregation while routing data from source sensors through intermediate nodes to the gateway.

Existing work has focussed on the execution of a single long-running aggregation query. In our new usage model, we allow *multiple* users to pose both long-running and snapshot queries (i.e. queries executed once). As new queries occur, they are not sent immediately to the network for evaluation, but are gathered at the gateway node into batches and are dispatched for evaluation once every *epoch*. The query optimizer groups together queries with the same aggregate operator and optimizes each group separately. Hence, in our presentation of our optimization techniques, we assume that queries use the same aggregate operator. Each epoch consists of a *query preparation* (QP) and a *result propagation* (RP) phase. In the QP phase, all queries gathered during the previous epoch are sent to the network together for evaluation. In the RP phase, query answers are forwarded back to the gateway.

An example query workload is illustrated in Figure 1. Request r_{ij} represents the jth result of query q_i. Queries q_1 and q_2 are presented in the first QP, while q_3 and q_4 are presented in the second. Query q_1 is active during the second epoch, while q_2 terminates after the first epoch. Notice that our model

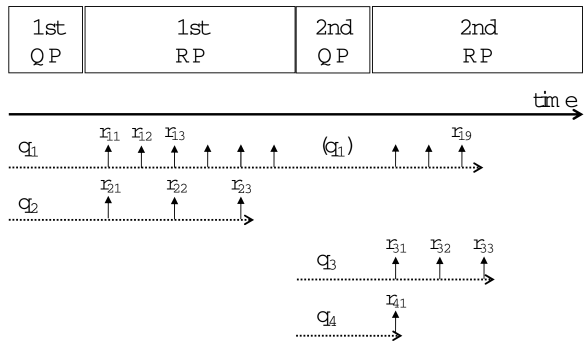

Fig. 1. An example query workload.

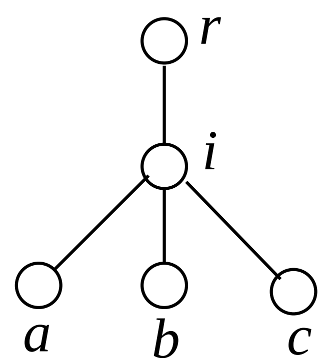

Fig. 2. An example communication tree.

is general enough to include queries with different result frequencies and different lifespans. Although the algorithms proposed in this paper apply to this general usage model, for ease of presentation we will restrict our discussion to a simpler scenario: all queries asked in a QP phase have the same result frequency, and their computation spans a single (and entire) RP phase. The duration of an RP phase is a tunable application-specific parameter. Typically, an RP phase is long enough to include multiple *rounds* of query results. To summarize, an *epoch* has a QP and an RP phase, and an RP phase has many *rounds* in which query results are returned to the gateway.

The Intelligent National Park. To give an example of a scenario with multiple queries, consider a sensor network deployed in a national park. Visitors of the park are provided with mobile devices that allow them to access a variety of information about the surrounding habitat by issuing queries to the network through a special purpose *gateway*. For instance, visitors may wish to know counts of certain animal species in different regions of the park. The region boundaries will vary depending on the location of the visitors. The queries also change with time, as visitors move to different sections of the park, and certain queries are more popular than others. In addition, the sensor readings change probabilistically as animals move around the park, and there might be different update rates during the day than at night. In order to extend the lifetime of the sensor network, the park's gateway does not answer queries immediately, but batches queries into *epochs*, and sends all the queries posed during an epoch together to the sensor network for evaluation. Query results are routed to the gateway at the end of every round until the end of the epoch.

Our Contributions. Query processing in large-scale sensor networks poses a number of challenges. First, we need to minimize the communication cost of data dissemination, while ensuring that all the queries posed are correctly computed. This is especially challenging if the rates at which sensors are updated vary across nodes. Furthermore, our solution needs to be deployable in a completely distributed manner and must observe the computational and memory constraints of the sensor nodes. Since communication links may fail and the data dissemination tree reorganized, our solution must also be adaptive to topological changes. This paper addresses these challenges and makes the following key contributions:

- **Multi-Query Optimization In Sensor Networks: Concepts and Complexity.** We formally introduce the concept of *result sharing* for efficient processing of multiple aggregate queries. The main idea is to share intermediate results whenever possible, taking the distribution of results in the network into account. We also address the problem of irregular sensor updates by developing *result encoding* techniques that send only a minimum amount of data that is sufficient to evaluate the updated queries. Our result sharing and encoding techniques achieve optimal communication cost for *sum* and related queries (such as *count* and *avg*). While some of our techniques also extend to *max* and *min* queries, we show that the problem of minimizing communication cost is NP-hard for these queries.
- **Distributed Deployment of Multi-Query Optimization.** We refine our multi-query optimization algorithms to account for computational and memory limitations of sensor nodes, and present fully distributed implementations of our algorithms. Besides a communication-optimal algorithm, we propose a near-optimal algorithm that significantly decreases the computational effort. We show how to tune our

algorithms to take into account the node computational capabilities, and the relative energy expended for communication and for computation. We also show how to adapt our algorithms to link failures that change the structure of the dissemination tree.

- **Implementation Results Validating our Techniques.** We present results from an empirical study of our multi-query optimization techniques with several synthetic data sets and realistic multi-query workloads. Our results clearly demonstrate the benefits of effective result sharing and result encoding. We also present a prototype implementation on real sensor nodes and demonstrate the time and memory requirements of running our code with different query workloads.

Relationship to Traditional Approaches for Multi-Query Optimization (MQO). The problem considered in this paper is significantly different from the traditional MQO problems. The difficulty in devising efficient MQO algorithms for sensor networks is not only in finding common subexpressions, but in dealing with the challenges of distribution and resource constraints at the nodes. This paper is, to the best of our knowledge, the first piece of work to i) formulate this important problem, and ii) give efficient algorithms with provable performance guarantees that are shown to work well in practice.

II. An Illustrative Example

In this section we present a simple example to illustrate our optimization techniques, which are then presented in detail in Section IV. Recall from the introduction that in the QP (query preparation) phase queries are propagated from the gateway node to the network. As a result of query propagation, a tree is created that connects all nodes to the gateway [3]; such a tree will be hereafter referred to as dissemination tree. Each sensor node is a potential data source and the single gateway node is at the root. Internal nodes of the tree are used to process and route information from the source nodes to the gateway node. Figure 2 shows a simple example tree. Here the root r is the gateway node, the leaves a, b and c are the data sources, and values must be routed from the data sources through internal node i to the root as needed to compute query results. Data sources are located at the leaf nodes only for the purposes of this example; in general, we are able to deal with any intermediate node generating sensor readings. In the remainder of this section we present two scenarios of query evaluation on the tree of Figure 2.

Deterministic Updates: For the simplest scenario, suppose that each sensor produces a new value in each round. We call this the D-scenario: it assumes that data updates occur deterministically, with probability 1. In each round, each sensor node sends its current sensor value to its parent in the dissemination tree. Interior nodes of the tree compute sub-aggregates of the values they receive from their children, and forward them up the tree towards the root. Under these conditions, multi-query optimization involves recognizing when the values of sub-aggregates can be shared effectively among queries, so that redundant data messages can be eliminated.

Consider evaluating the three queries $a + b$, $a + b + c$, and c on the tree of Figure 2. In each round all leaves send their values to the interior node i, which then has enough information to compute the values of all the queries. The three queries $a + b$, $a + b + c$, and c, however, are not linearly independent – the values of any two of them can be used to calculate the value of the third. Thus, node i need forward only *two* of the values (say $a + b$ and c). The root can then recompute the third value ($a + b + c$) locally, achieving a net saving of energy. This technique of "reducing" the set of data values forwarded toward the root can be repeated at every subtree.

Irregular Updates: Next we consider the I-scenario, in which sensor updates are irregular, with the update rates varying across different sensors. The goal of the query optimizer is to choose an efficient "result encoding," sending the minimum amount of data up the tree that will enable the root both to identify the queries affected by the updated sensors and to compute the values of those queries. We call data sent for these purposes ResultCode and ResultData, respectively. Returning to the example in Figure 2, consider the same three queries discussed for the D scenario: $a + b$, $a + b + c$, and c. Suppose only sensor a is updated in a round. Clearly the information forwarded up edge (i, r) must inform the root that queries $a + b$ and $a + b + c$ are affected, and must include the current value of a. However, this does not imply that the root must "know" the exact set of sensors that were updated. It is easy to verify that sensors a and b and the aggregate $a+b$ are all indistinguishable by any of the queries in the workload. Consequently, rounds in which a changes, or b changes, or both a and b change, can all result in identical messages being sent along edge (i, r). Thus, in the I-scenario the goal of the multi-query optimizer is to find an optimally compressed result encoding that eliminates unnecessary distinctions in the RESULTCODE component in addition to representing the RESULTDATA component efficiently. This is a more difficult problem than in the D-scenario, since the expected performance of a result encoding clearly depends on the distribution of sensor updates.

III. OPTIMIZATION PROBLEMS AND COMPLEXITY

We now formally present the multi-query optimization, and give some complexity results, showing that certain aspects of the problem are intractable while for others there are centralized optimal algorithms that run in polynomial time. In this section, we focus on algorithms that aim to minimize the communication cost of query evaluation ignoring any computation limitations or issues of distributed implementation. In Section IV we will develop fully distributed algorithms that take into consideration the computation and memory constraints in sensor networks.

We consider a set of aggregate queries $Q = \{q_1, \dots, q_m\}$ over a set of k distinct sensor data sources. A set of sensor readings is a vector $x = \langle x_1, \dots, x_k \rangle \in \Re^k$. Each query q_i requests an aggregate value of some subset of the data sources at some desired frequency. This allows each query q_i to be expressed as a k-bit vector: element j of the vector is 1 if x_j contributes to the value of q_i, and 0 otherwise. The *value* of query q_i on sensor readings x is expressed as the dot product $q_i \cdot x$.

In our multi-query optimization problem, we are given a dissemination tree connecting the k sensor nodes and the gateway, over which the aggregations are executed. Note that our solutions apply to any given tree. The goal is to devise an execution plan for evaluating queries, that minimizes total communication cost. The communication cost includes the cost of query propagation in the QP phase and the cost of result propagation in each round of the RP phase. While we discuss the implementation of the QP phase in detail in Section IV-B, we ignore the query propagation cost in the following analysis, since it is negligible compared to the total result propagation costs, whenever the RP phase of an epoch consists of a sufficiently large number of rounds. We focus on two classes of aggregation: (i) *min* queries and (ii) *sum* queries. Clearly our results for *min* queries also apply to *max*, and our results for *sum* queries can be extended to *count*, *average*, moments and linear combinations in the usual way.

Complexity of *sum* Queries: For *sum* queries the underlying mathematical structure is a field. We can exploit this fact, using techniques from linear algebra to optimize the number of data values that must be communicated. Let N be an arbitrary node in the tree. Let $P(N)$ denote the parent of N and let T_N denote the subtree rooted at N. We denote as $x(N)$ the vector of sensor values in the subtree T_N and $Q_{x(N)}$ the set of query vectors projected only onto sensors in T_N.

We present a simple method to minimize the amount of data that N sends to $P(N)$ in each round. Let $B(Q_{x(N)}) = \{b_1, \dots, b_n\}$ be a basis of the subspace of $\Re^k$ spanned by $Q_{x(N)}$. Then any query $q \in Q_{x(N)}$ can be expressed as a linear combination of the basis vectors $q = \sum_j \alpha_j \cdot b_j$, where $\alpha_j \in \Re$, $j = 1, \dots, n$. By linearity of inner product we get, for sensors $x(N)$ (in the subtree T_N)

$$q \cdot x(N) = (\textstyle\sum_j \alpha_j \cdot b_j) \cdot x(N) = \textstyle\sum_j \alpha_j \cdot (b_j \cdot x(N))$$

That is, to evaluate the answers of queries in $Q_{x(N)}$ it suffices to know the answers for any basis of the query space spanned by $Q_{x(N)}$. Any maximal linearly independent subset of $Q_{x(N)}$ is a (not necessarily orthogonal or normal) basis of the space and every such basis has the same cardinality. So we can use any maximal linearly independent subset of $Q_{x(N)}$ as our basis, and N can forward the answers of the queries in this basis to $P(N)$. The parent $P(N)$, using the same set of basis vectors, can easily interpret the reduced results that it receives from N. We assume that N and $P(N)$ use the same algorithm in order to identify the basis vectors of $Q_{x(N)}$, and the factors α_j. We refer to this procedure as *linear reduction*.

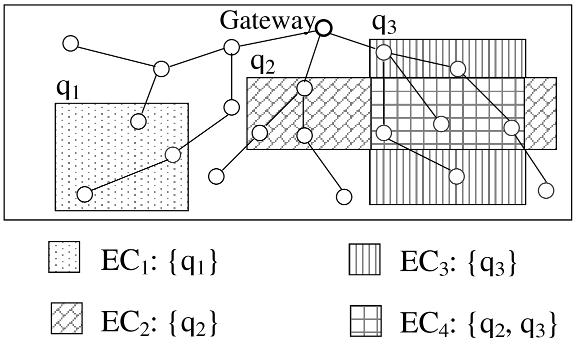

Fig. 3. Equivalence classes formed by three queries.

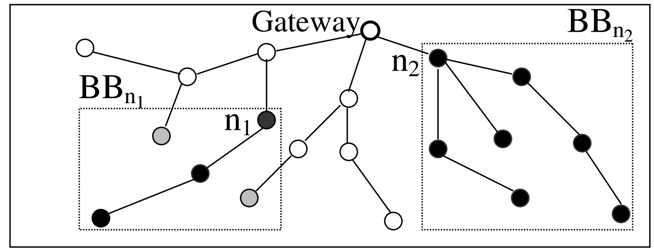

Fig. 4. The bounding box of a subtree is the minimum rectangle that covers all sensors in the subtree. Grey nodes represent sensors that belong to a bounding box of a subtree without belonging to the subtree.

Theorem 1: The size of the query result message sent by the above algorithm in each round is optimal.

Complexity of *min* Queries: For *min* queries, the underlying structure is not a field (it lacks inverses) and the above algebraic techniques do not apply. In fact, the multi-query optimization problem for *min* queries turns out to be intractable even for very simple dissemination trees. We show it is NP-hard by a reduction from the Set Basis problem. An instance of the Set Basis problem is collection C of finite subsets of a set S, together with an integer k. The problem is to find a collection B of at most k subsets of S such that each element of C can be achieved as the union of some elements of B. Mapping an instance of this problem to an instance of multi-query optimization is straightforward. The dissemination tree consists of a root node, a single interior node, and $|S|$ leaf nodes. We map S to the set of sensors at the leaf nodes, and C becomes the set of *min* queries to be computed. There is an execution plan sending at most k messages to the root in each round if and only if the given instance of Set Basis has a solution.

Theorem 2: The multi-query optimization problem for *min* queries is NP-hard.

IV. Multi-Query Optimization

The linear reduction technique outlined in Section III provides an elegant solution for minimizing the cost of processing multiple *sum* queries. However, a number of system considerations and details have to be taken into account to apply to a real sensor network. In this section, we develop fully distributed multi-query optimization algorithms for *sum* queries. We start our discussion by introducing the notion of *equivalence class,* which is central to the algorithms proposed in the remainder of the section.

A. Queries and equivalence classes

Rectangular Queries: We have represented each query as a k-bit vector, where k is the number of sensors. Expressing queries in this form requires that the user have complete knowledge of the sensor topology. It is more natural, and generally more compact, to represent queries spatially. In the remainder of this paper, we focus our attention on queries that aggregate sensor values within a rectangular region, and represent such a query as a pair of points $((x_0, y_0), (x_1, y_1))$ at opposite corners of the rectangle. Since queries do no longer enumerate nodes specifically, we can even evaluate queries in an acquisitional manner [8], e.g. by selecting a sample of sensor values generated within a query rectangle.

Equivalence Classes (ECs): To deal efficiently with rectangular queries and distribution, we introduce the notion of *Equivalence Class* (*EC*). An equivalence class is the union of all regions covered by the same set of queries. For example, Figure 3 shows that queries $\{q_1, q_2, q_3\}$ form four ECs $\{EC_1, EC_2, EC_3, EC_4\}$, each one of which corresponds to a different set of queries. For instance, EC_4 is covered only by queries q_2 and q_3, and can be represented by the column bit-vector $[0, 1, 1]^T$; EC_1 is represented by $[1, 0, 0]^T$, EC_2 by $[0, 1, 0]^T$ and EC_3 by $[0, 0, 1]^T$. Notice that an equivalence class is not necessarily a connected region (see EC_2). An equivalence class may contain no sensors. Equivalence classes are identified based solely on spatial query information; they are independent of the node locations in the network or of the dissemination tree that connects nodes. We can, however, speak of the *value* of an equivalence class – this is the aggregate of the data values of sensors located in the EC region. The value of an EC can be obtained by a subset of sensors located in the EC region, if an acquisitional processing style is adopted.

Query Vectors and Query Values: We can now express queries in terms of ECs as follows. We number equivalence classes (instead of sensors) from 1 to ℓ, where ℓ is the number of equivalence classes. Let x denote the column vector in $\Re^\ell$ representing the values of the equivalence classes; thus, x_i denotes the sum of (all or a sample of) sensor values in EC_i. Each query q is a linear combination of the set of equivalence classes and can be captured by a row vector in $\{0,1\}^\ell$. For example, query q_3 in Figure 3 can be represented as the vector $[0,0,1,1]$, since it only covers EC_3 and EC_4. The value of a query q given the EC values x is simply the product $q \cdot x$. Given the above representation of the queries and EC values, it is natural to represent a set of m queries as an $m \times \ell$ (bit) matrix Q, in which the (i,j) element is 1 if the ith query in Q covers the jth equivalence class. The value of the query set Q given the EC values x is again given by the product $Q \cdot x$, which is a column vector in $\Re^m$. We often refer to the rows of a query-EC matrix as *query vectors*, and to the columns as *EC vectors*.

Bounding Boxes (BBs): Expressing queries in terms of ECs brings out the dependencies among queries. In order to exploit these dependencies fully, each node needs to view queries in the context of its subtree, rather than the entire network. Therefore, in our algorithms, a node expresses queries in terms of ECs intersecting with its subtree; an EC intersects with a subtree if any of the sensor nodes in the subtree lies within the EC region. A node N can accurately determine which ECs intersect with subtree T_N, if it either knows the locations of all nodes in T_N or receives from its children a list of all ECs intersecting with their subtrees. Both approaches are prohibitive in terms of the communication involved. An approximation of the set of equivalence classes intersecting T_N can be obtained if we consider the minimum rectangle that contains all sensors in the subtree. This rectangle is hereafter called the *bounding box* of T_N and is denoted as BB_N. Figure 4 depicts the bounding boxes of the subtrees rooted at nodes n_1 and n_2. Note that a bounding box may contain nodes that do not belong in the subtree (grey nodes in Figure 4).

Queries and ECs projected to the bounding box of a subtree: Let X_N denote the set of equivalence classes that intersect with the bounding box of the subtree T_N. It is easy to see that X_N is a superset of the equivalence classes that actually intersect with the subtree T_N. For given query set Q, we let Q_N denote the projection of Q on to X_N; that is, we obtain Q_N by setting all entries of Q that appear in columns not in X_N to be zero. Duplicate and zero rows are removed. We extend the notation to let x_N denote the vector of projected EC values onto the subtree T_N (*not* onto BB_N since a node N can only receive values generated by its descendants). The ith entry corresponds to the sum of sensor values lying in the intersection of EC_i and the subtree T_N. The entries of ECs that do not intersect with T_N are set to 0. If we denote the values of queries Q that are contributed from sensors in the subtree T_N as $V(Q,N)$, then the vector of values of Q_N contributed from subtree T_N is $V(Q_N,N) = Q_N \cdot x_N$.

B. The Query Preparation (QP) phase

The query preparation phase consists of three steps: a *bounding-box calculation* step, a *query propagation* step, and an *EC evaluation step*. Some of our algorithms for the RP phase do not require the evaluation of ECs, in which case the last step is omitted.

Bounding-box calculation: A dissemination tree is first created using a simple flooding algorithm. Given the dissemination tree, each node N computes the bounding box BB_N of its subtree T_N from the bounding boxes of the subtrees of its children (if any) as follows. If x, (resp., x') and y (resp, y') are the smallest (resp., largest) x- and y-coordinates of the child bounding boxes, then (x,y) and (x',y') form two opposite corner points of the bounding box of N.

Query propagation: The next step is to send query information down the dissemination tree; the query information includes the set of sensors involved in the query as well as its frequency. We distinguish query propagation schemes based on whether bounding boxes are used to reduce the query propagation cost: (i) AllQueries: flood all queries to the entire network; (ii) BBQueries: each node propagates down only queries that have a non-empty intersection with its bounding box. This is performed using semantic routing information, discussed in detail in [8]. Once a node receives information about the relevant queries, it can compute for each round in the epoch the set of queries that are active in the round.

EC computation: Given a set of m query rectangles, we can compute all the ECs formed by the m queries using a two-dimensional sweep algorithm in $O(m^3)$ time using $O(m^3)$ space. Due to space constraints, we defer the algorithm description and its analysis to the full paper. Using this algorithm, each node locally computes the ECs intersecting with its bounding box.

C. The Result Propagation (RP) phase

Each RP phase consists of a number of *rounds*; in each round, aggregation results are forwarded through the tree paths from the leaves to the gateway. Consider a result message sent by a node N to its parent $P(N)$. The data forwarded by N to $P(N)$ should be sufficient to evaluate $V(Q_N, N)$, i.e. to evaluate the *contribution* of sensors in T_N to the values of the projected queries Q_N. A result message that node N sends to its parent $P(N)$ consists of a pair $\langle$RESULTCODE, RESULTDATA$\rangle$; RESULTDATA includes updated values, and RESULTCODE encodes what has been updated, showing how to interpret the values in RESULTDATA.

We now propose a series of result propagation algorithms, all of which use the above message format. These algorithms can be classified according to four dimensions. The first two dimensions are the methods employed for computing the RESULTCODE and RESULTDATA components. The third dimension is whether the linear reduction technique of Section III is applied. The last dimension is whether these choices are identical for all nodes, yielding a *pure* algorithm, or these choices may differ across nodes, yielding a *hybrid* algorithm.

Pure algorithms without reduction: We consider two methods for determining the RESULTCODE component of a message that a node sends to its parent. In *Query-encoding*, a node sends to its parent information about *which queries have been updated* since the last round. Formally, let $UpdRows(Q_N)$ be the matrix derived from Q_N after removing all queries (row vectors) that are not affected by the current sensor updates in T_N. Both N and $P(N)$ agree on unique labels for the queries in Q_N from the integer interval $[1, |Q_N|]$. Then, RESULTCODE consists of a set of $\lg |Q_N|$-bit labels listing the queries in $UpdRows(Q_N)$. We note that Query-encoding does not require computation of equivalence classes. In *EC-encoding*, a node sends to its parent information about *which equivalence classes have been updated* since the last round. Let $UpdCols(Q_N)$ be the matrix derived from Q_N after removing all ECs (column vectors) that do not include any updated sensors in T_N (and after removing duplicate and zero rows). Since both N and $P(N)$ can compute X_N (i.e. the set of equivalence classes that intersect with BB_N) they can agree on a unique label in the range $[1, |X_N|]$ for each equivalence class in X_N. In EC-encoding, RESULTCODE includes the identifiers of ECs (columns) of $UpdCols(Q_N)$.

The size of RESULTCODE depends on the probability distribution of sensor updates. Instead of including a list of identifiers of updated (*present*) queries (resp. ECs) in RESULTCODE, we could only include identifiers of non-updated (*absent*) queries (resp. ECs) or simply represent the updated queries (resp. ECs) as a bit vector (*vector*). Different strategies might be better for different sensor update probabilities. In our experiments, we use an adaptive hybrid scheme that selects at each node the optimal of the three strategies (present, absent and vector).

We also consider two methods for populating the RESULTDATA component of a result message that a node sends to its parent. In the *Query-data* approach, RESULTDATA is the set of values of updated queries. In the *EC-data* approach, RESULTDATA is the set of values of updated EC values.

One can combine the two dimensions above to obtain four different algorithms for the RP phase: QueryQuery, QueryEC, ECQuery and ECEC, respectively, where the first part of the name refers to the encoding, and the second part to the data. EC-encoding results in messages with smaller RESULTDATA components than Query-encoding, independent of whether the Query-data or EC-data policy is used. This is because both (row and column) dimensions of $UpdCols(Q_N)$ are smaller than those of $UpdRows(Q_N)$.

Basis Evaluation Step at Node N:
Input:
i) $\{BB_{N_k}\},\ k = 1, \ldots, ch$ // child bounding boxes
ii) BB_N // my bounding box
iii) $UQ_N = UpdCols(Q_N)$
iv) $UQ_{N_k} = UpdCols(Q_{N_k}),\ k = 1, \ldots, ch$
Pseudocode:
1. repeat for each child k
$(B(UQ_{N_k}), A_{N_k})$ = linear_reduction(UQ_{N_k});
// s.t. $B(UQ_{N_k})$ is a basis of UQ_{N_k},
// and $UQ_{N_k} = A_{N_k} \cdot B(UQ_{N_k})$
2. do $(B(UQ_N), A_N)$ = linear_reduction(UQ_N);
// s.t. $B(UQ_N)$ is a basis of UQ_N,
// and $UQ_N = A_N \cdot B(UQ_N)$

Fig. 5. Basis Evaluation Step

Result Evaluation Step at Node N:
Input:
i) $\{V(B(UQ_{N_k}), N_k)\}, k = 1, \ldots, ch$ // from children
ii) $\{A_{N_k}\}, k = 1, \ldots, ch$ // from basis eval. step
iii) $B(UQ_N)$ // from basis eval. step
iv) UQ_N, $\{UQ_{N_k}\}, k = 1, \ldots, ch$ // from basis eval. step
Pseudocode:
1. repeat for each child k
-$V(UQ_{N_k}, N_k) = A_{N_k} \cdot V(B(UQ_{N_k}), N_k)$
-$V(UQ_N, N_k)$ = assign_values($UQ_N, UQ_{N_k}, V(UQ_{N_k}, N_k)$)
2. V = evaluate_my_contrib_to_queries (UQ_N)
3. $V(UQ_N, N) = V + \sum_{k=1}^{ch} V(UQ_N, N_k)$
4. $V(B(UQ_N), N)$=select_basis_values($V(UQ_N, N), B(UQ_N)$)
// where $V(B(UQ_N), N) \subseteq V(UQ_N, N)$
5. forward_basis_values_to_parent($V(B(UQ_N), N)$)

Fig. 6. Result Evaluation Step

Therefore, if the computational capabilities of the sensor nodes allow EC-computations, then we only consider ECQuery and ECEC. On the other hand, if the computational limitations of the sensor nodes do not allow them to compute the ECs, then QueryQuery is the only algorithm of interest. Consequently, we focus our attention on three of these four algorithms, namely, QueryQuery, ECQuery, and ECEC.

- **ECQuery:** In the RESULTCODE component, each node N sends to $P(N)$ the identifiers of the updated ECs in the subtree rooted at N. In the RESULTDATA component, node N includes delta values only of the distinct row vectors of matrix $UpdCols(Q_N)$. That is, query vectors are projected only onto the updated ECs (columns), and one value is sent for each distinct projected query vector.
- **QueryQuery:** In the RESULTCODE component of the message that N sends to $P(N)$, it includes the identifiers of updated queries. In RESULTDATA, node N includes delta values of the distinct row vectors of matrix $UpdRows(Q_N)$. Since the number of distinct query (row) vectors in $UpdRows(Q_N)$ is larger or equal to their number in $UpdCols(Q_N)$, the size of RESULTDATA in QueryQuery is larger or equal to its counterpart in ECQuery.
- **ECEC:** The RESULTCODE component is identical to that of ECQuery. Unlike ECQuery, ECEC sends up EC values in the RESULTDATA component of the message. For each updated EC in the subtree, it sends up the aggregate value of all sensors in the intersection of the EC and the subtree T_N.

An optimal pure algorithm using linear reduction: Both ECQuery and ECEC decrease the communication cost of result propagation by explicitly encoding irregular updates. Additional communication savings can be achieved by carefully applying the linear reduction technique (introduced in Section III) in a distributed manner to reduce the size of propagated irregular updates. We now present the algorithm **ECReduced** which uses EC-encoding, and is provably optimal with respect to the amount of result data that is communicated. The RP phase of ECReduced at each node consists of two steps: i) a basis evaluation step and ii) a result evaluation step. The basis evaluation step is executed whenever the set of active queries changes or the set of updated ECs changes. Thus, if every query has the same frequency and all sensors are updated regularly (D-scenario), then the basis evaluation step is executed only once at the beginning of the RP phase. This step is the most computationally demanding part of our algorithm since it involves matrix linear reduction; the complexity of reducing a matrix with m rows and n columns is $O(mn^2)$.

Basis evaluation step: Consider a node N with ch children nodes. Node N initially performs ch row-based linear reductions on matrices $UpdCols(Q_{N_k})$, $k = 1, \ldots, ch$, in order to interpret the results received from its children $N_1, \ldots, N_{ch}$. Node N then reduces its own query-EC matrix $UpdCols(Q_N)$ into a set of linearly independent query vectors. Overall, N performs $ch + 1$ matrix reductions. An illustration of this step is shown in Figure 5, if we set UQ_N to $UpdCols(Q_N)$ and UQ_{N_k} to $UpdCols(Q_{N_k})$. In task 1 of Figure 5, node N does not only derive the basis vectors $B(UQ_{N_k})$ for each child k, but also a coefficient matrix A_{N_k}, such that the product of A_{N_k} and $B(UQ_{N_k})$ yields the original projected queries UQ_{N_k}. Matrix A_{N_k} can be easily computed if we use the Gauss-Jordan elimination method for linear reduction, and observe the row transformations performed. The coefficients in matrix A_{N_k} will become input of the next result evaluation step.

Result evaluation step: This step is executed once per round of the RP phase, and it includes simple operations wrt time and memory space. Relying on the output of the basis evaluation step, node N combines the incoming (delta) values received from its children and forwards a minimum number of values to its parent $P(N)$. Details of this step are shown in Figure 6, where UQ_N is set to $UpdCols(Q_N)$ and UQ_{N_k} is set to $UpdCols(UQ_{N_k})$ as in Figure 5. First, for each child N_k, node N evaluates the values of queries (row vectors of) UQ_{N_k}, based on the values of the basis vectors $B(UQ_{N_k})$ and matrix A_{N_k}. More specifically, $A_{N_k}[i,j]$ denotes how the value of the j-th row vector of $B(UQ_{N_k})$ contributes to the value of the i-th query in UQ_{N_k}. Node N then assigns the contribution of subtree T_{N_k} to queries UQ_N as follows: if a query $q_i \in UQ_N$ has an empty projection to (the updated EC regions of subtree) T_{N_k}, then the assigned (delta) value is 0. Otherwise, q_i takes the value of the corresponding projected query onto T_{N_k}. In task 2 (Figure 6), N evaluates the contribution of its own reading to queries UQ_N. Combining the values from the previous two tasks, node N proceeds to evaluate the results of queries UQ_N for the entire subtree T_N (task 3). As shown in task 4, it is sufficient to evaluate only the values of queries that belong to the basis $B(UQ_N)$. Only those values, denoted as $V(B(UQ_N), N)$, are finally forwarded to the parent node $P(N)$ (task 5). Notice that if all nodes use the same algorithm to linearly reduce a query matrix, there is no need to communicate the selected basis vectors; a node only forwards up the values of these vectors to its parent. The following result is derived from Theorem 1.

Theorem 3: The size of the RESULTDATA component in the ECReduced algorithm is optimal; it is a lower bound on the size of the optimal result message.

Hybrid algorithms with no reduction: The algorithms introduced so far are executed in an identical manner at all nodes. We now consider two hybrid algorithms that perform differently across nodes, depending on the load of results contributed by the underlying subtrees. The first algorithm, referred to as HybridBasic, attempts to approximate the optimal cost achieved by the ECReduced algorithm, while avoiding the high computational requirements for linear reduction.

- **HybridBasic:** Consider the bounding box of a node and the set of queries and ECs intersecting with the bounding box. For a given sensor update rate, when the number of (projected) queries is small, the number of (projected) ECs is greater than the number of queries. In this case, the ECQuery algorithm is expected to outperform the ECEC algorithm. However, for a large number of queries the equivalence classes might be fewer than the queries. In this case, the ECEC algorithm is expected to outperform the ECQuery algorithm. The point where the two algorithms cross depends on the sensor update frequency. The HybridBasic algorithm combines the ECEC and ECQuery approaches. A node selects the approach that locally yields the least cost, and sends an additional bit to denote its choice. The only constraint is that if a child uses the ECQuery approach, it only provides information about the values of updated queries; hence, its parent can only implement the ECQuery approach. On the contrary, a parent of a node that implements ECEC can implement either of the two approaches.

Surprisingly, HybridBasic performs extremely well in terms of communication; as will be shown in Section V, it closely approximates the cost of the ECReduced algorithm, without requiring a linear reduction task. In fact, HybridBasic can be viewed as an approximate application of linear reduction in the following sense: the rank of a matrix is always smaller or equal to the smallest dimension of the matrix; given a query-EC matrix, HybridBasic effectively chooses to propagate values of row vectors (queries), or of column vectors (ECs) depending on which ones are fewer. In practice, this policy works well, since the cardinality of the smallest matrix dimension often coincides with the matrix rank.

HybridBasic assumes that each node is able to evaluate equivalence classes within the bounding box of its subtree. The following algorithm, named HybridWithThreshold, lifts this requirement for nodes close to the gateway, whose bounding boxes overlap with many queries.

- **HybridWithThreshold:** If the input query workload is light, EC evaluation for the entire network is easy to perform locally at each node. Otherwise, nodes close to the leaves may opt for local EC computation, i.e. computation of ECs within the context of the bounding box of the node's subtree. As we approach the gateway, the bounding box of a node's subtree increases, and so do the number of query rectangles that intersect with the bounding box. The computational cost of evaluating ECs may become prohibitively expensive for nodes close to the gateway. The HybridWithThreshold algorithm behaves like the HybridBasic algorithm at nodes that are able to perform EC computation. When the effort for EC computation exceeds a certain threshold at a node (its computational capability), the node switches to Query-encoding and sends up one result per updated query.

D. Handling failures

We now briefly consider the impact of failures on the result propagation schemes. In the general case, a change in the dissemination tree will change the bounding boxes for a subset of the nodes. Since in each of the result propagation schemes, a node projects queries to its bounding box, it follows that the computation of this projection must be updated for each node whose bounding box has changed. In addition, the calculation of the EC-ids has to be performed at some of the nodes for all algorithms that use EC-encoding; if instead of "local EC-ids", we use global EC-ids (which require more bits), then this computation can be avoided at the expense of increasing the per-round communication. Finally, in the ECReduced algorithm, all the nodes for which the bounding boxes have changed, and their parents, need to recompute the basis to achieve result size reduction. It is important to note, however, that no additional communication is incurred in the result propagation schemes.

V. Experimental Evaluation

We first measure the communication cost of the proposed algorithms under synthetic query and sensor update workloads using a home-grown simulator (Sections V-A and V-B). In Section V-B. we also present our feasibility test of the linear reduction technique, which we performed on the Mica2 mote. In Section V-C, we evaluate how the proposed algorithms trade communication for computation. Finally, in Section V-D we show the benefits of our techniques by drawing data from a real sensor network infrastructure deployed in the Intel Berkeley Research Lab.

A. Synthetic experimental setup

We deploy 400 sensors in a square region of 400 m^2 and randomly select their x and y coordinates to be any real numbers in $[0, 20]$. We ensure that with a communication range of $2m$ the random deployment of nodes results in a (100%) connected network (otherwise the random deployment is repeated). A flooding algorithm is used to generate a minimum spanning tree that connects all nodes to the gateway. Each node selects as its parent a randomly chosen neighbor that lies on a shortest path to the root. The queries considered in our framework are *sum* queries that cover all sensors in a rectangular area. In our experiments we test a number of different query workloads, each defined as a set of tuples of the form (numberOfQueries, minQueryWidth, maxQueryWidth, minQueryHeight, maxQueryHeight). We assume that all the queries in a workload have the same frequency. We set the minimum values of the query dimensions (minQueryWidth, minQueryHeight) to $1m$ and the maximum values to $20m$. Given query input patterns, a random workload generator generates specific *instances* in each epoch that satisfy the patterns. The sensor update workload defines the probability that a sensor is updated at the end of a round. Given a sensor update input *pattern*, a random workload generator selects a specific set of sensors to be updated in a round.

For simplicity, we assume long-running queries that are propagated once at the beginning of an epoch (in the QP phase) and are evaluated at every round of the RP phase until the end of the epoch. Since the query propagation cost occurs once per epoch, it is negligible compared to the result propagation cost and is not accounted for. In our evaluation, we measure the result (communication) cost *per round*, averaged over 200 rounds (10 epochs of 20 rounds each).

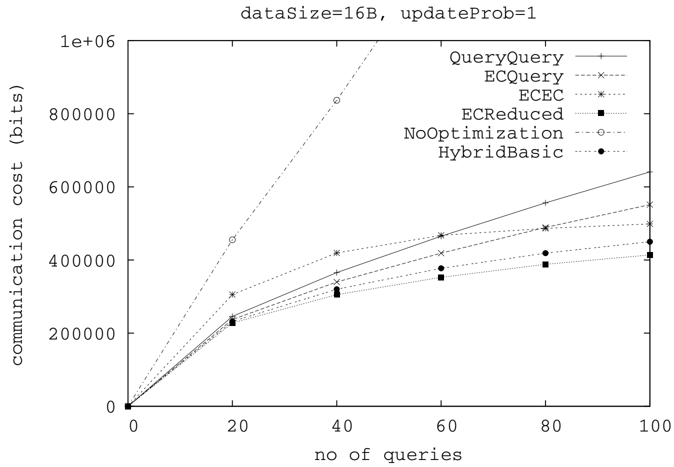

Fig. 7. Communication cost of algorithms in the D-scenario.

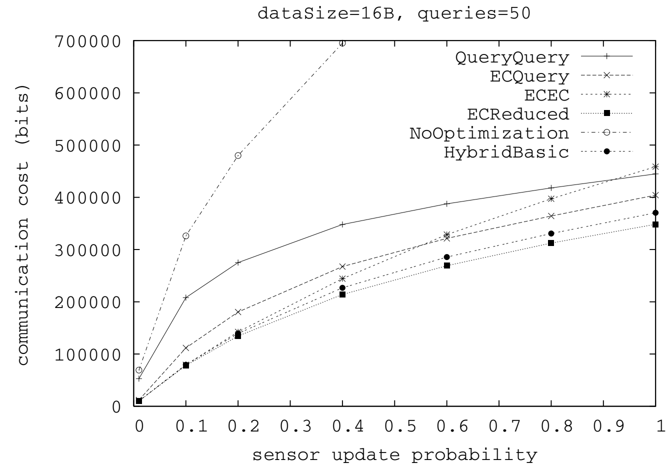

Fig. 8. Communication cost of algorithms in the I-scenario.

B. Communication cost

In the first set of experiments we ignore the computation cost. To make the measurements of communication cost realistic, we consider a packet size of 34 bytes (similar to the size of TOS_Msg used for Mica motes) that consists of a 5-byte header and a 29-byte payload. If the number of bits in a message is x, then the communication cost is $\lceil x/29 \rceil \times 34$; that is, we account for a fixed header cost (5 bytes) and only consider fixed size packets. The size of each query result is set to 16 bytes.

Deterministic sensor updates: In Figure 7, we compare the performance of different algorithms as we increase the number of queries sent together for evaluation at the beginning of an epoch. In this initial experiment, we assume that all sensors are updated in each round with probability 1 (D-scenario). We first compare our techniques with the existing approach, namely an extension of the TAG algorithm [3] to process multiple queries. Since this algorithm, which we refer to as NoOptimization, performs in-network aggregation independently for each query, the average (per round) result propagation cost increases linearly in the number of queries. The performance advantage of our proposed techniques is apparent even for light query workloads. In fact, our most basic optimization approach QueryQuery offers communication cost benefits of up to 50% for 20 queries (and the gap increases significantly with the number of queries). We also note here that an alternative approach in which all sensors push their data up to the gateway without in-network aggregation is always inferior to ECEC.

Figure 7 validates our analysis of Section IV-C that EC-encoding outperforms Query-encoding, if we restrict our attention to communication cost. Between ECQuery and ECEC, Figure 7 shows that ECQuery outperforms ECEC for query workloads with less than 80 queries, but as we increase the number of queries, the number of ECs became smaller than the number of queries and ECEC wins.

We now consider the cost and benefit of the reduction technique in the D-scenario. Figure 7 shows that the proposed ECReduced algorithm performs better than all the other algorithms, thus validating Theorem 3. In addition to our simulations, we implemented the linear reduction technique on the Berkeley Mica2 motes (4MHz ATMEL processor128kB flash, 4kB RAM, 4kB ROM) using the NesC programming language. We measured the time in seconds required for reducing an $m \times m$ matrix of floats as a function of m. The observed time grows as $\Theta(m^3)$, which is consistent with the complexity of the reduction algorithm. The algorithm completes successfully despite the limited storage and processing capabilities of the mote. The code for matrix reduction was compiled with ”make mica” to 12604 bytes in ROM and 428 bytes in RAM. For matrices of dimension from 5 to 15, the linear reduction algorithm takes 0.07 to 1 seconds, but the algorithm time increases rapidly for larger matrices.

A final important observation from Figure 7 is that the HybridBasic algorithm performs almost as well as the ECReduced algorithm, without requiring any computational cost for linear reduction (Figure 7). This is encouraging since it shows that a very simple distributed algorithm, which can easily be implemented

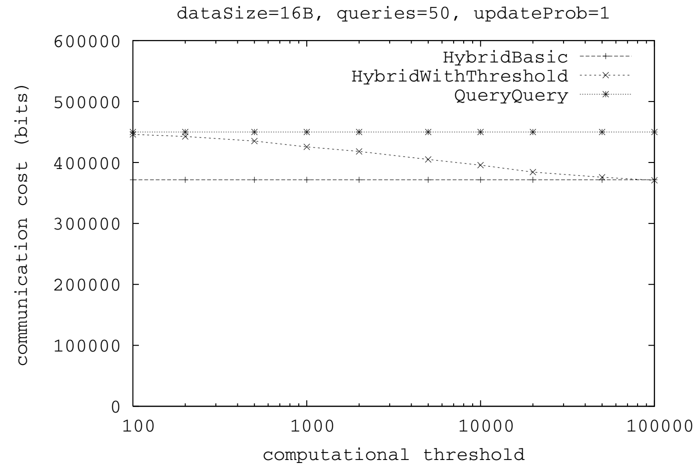

Fig. 9. HybridWithThreshold in the D-scenario.

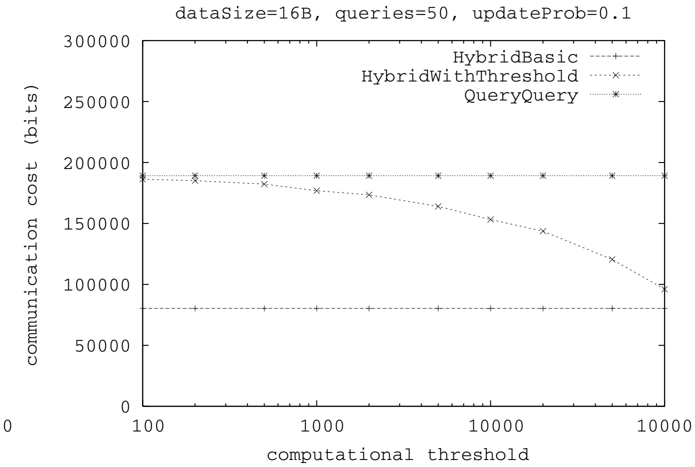

Fig. 10. HybridWithThreshold in the I-scenario.

on constrained sensor nodes, gives a very good approximation of the optimal solution.

Probabilistic sensor updates: In Figure 8, we compare the performance of different algorithms as we vary the probability that a sensor generates an updated reading in a given round. We set the number of queries to 50. Recall that in the D-scenario (Figure 7), which corresponds to the I-scenario with probability 1, ECQuery is preferred to ECEC for workloads of less than 80 queries. As we decrease the sensor update probability to less than 0.6, however, Figure 8 shows that it becomes beneficial for nodes to send up EC values instead of query values in RESULTDATA. For an update rate of 10%, ECEC is 30% cheaper than ECQuery. The ECReduced algorithm, which applies the linear reduction technique in a distributed manner, outperforms all other algorithms (Figure 8). Moreover, the HybridBasic algorithm has a very good performance, approaching closely the cost of ECReduced.

In Figures 9 and 10, we take into consideration the limited computational power of sensor nodes. From the two algorithms that do not require EC computation (NoOptimization and QueryQuery), we only consider QueryQuery because it has smaller communication cost. Among the algorithms that do not perform reduction but require EC computation, we only consider HybridBasic because it has similar computational requirements with the others yet smaller communication cost. We omit the study of ECReduced because it requires matrix reduction without yielding noteworthy cost savings compared to HybridBasic. In the experiment of Figure 9, we set the sensor update probability to 1, and in Figure 10 we perform the same experiment for update probability 0.1. In both figures, QueryQuery has a higher cost than HybridBasic. The former algorithm does not require knowledge of ECs, whereas the latter assumes knowledge of ECs independent of the nodes' computational capabilities. We study the performance of the HybridWithThreshold algorithm, where the significance of the threshold value is as follows: if the threshold value at a node N is greater than the effort of computing ECs (measured as m^3, where m is the number of distinct projected queries onto the local bounding box BB_N), then EC computation cannot be performed at N, and the node switches to using the QueryQuery algorithm. Figure 9 shows that as we increase the threshold value (plotted on a logarithmic scale), more nodes are able to compute ECs, and the cost of HybridWithThreshold approaches the cost of HybridBasic.

C. *Tradeoff between communication and computation cost in the presence of failures*

In our next set of experiments (Figures 11 and 12), we consider both the computational and the communication cost of our algorithms. In particular, we evaluate the total cost (computation and communication) as a function of the ratio between the communication and computation costs. Our experiments for the D- and I-scenarios indicate that while linear reduction and EC-encoding together offer the most communication-efficient approach (ECReduced), a hybrid of EC- and Query-encoding without invoking

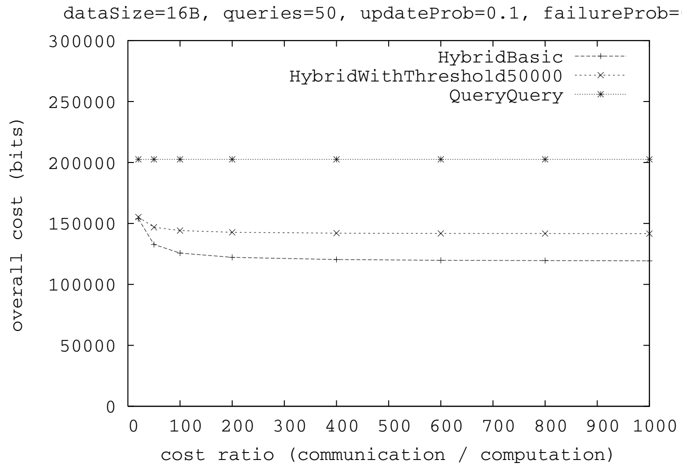

Fig. 11. Commun./Comput. tradeoff with 10% link changes.

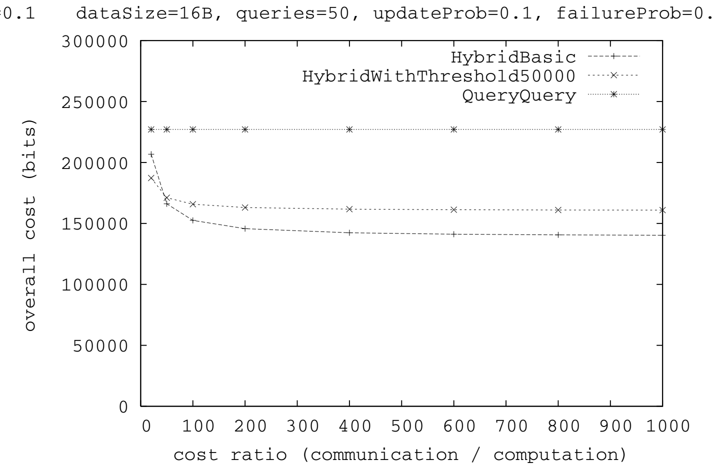

Fig. 12. Comm./Comput. tradeoff with 50% link changes.

linear reduction (HybridBasic or HybridWithThreshold) performs almost as well for our collection of synthetic workloads. Consequently, in this section, we only consider three algorithms: QueryQuery, which performs neither linear reduction nor EC computations, and HybridBasic and HybridWithThreshold.

The computation cost at a node is measured as the effort needed to evaluate ECs. Hence it is zero for QueryQuery, and m^3 for HybridBasic at a node N, where m is the number of distinct projected queries onto BB_N. For HybridWithThreshold, we adopt a threshold value of 50000. We set the sensor update probability to 0.1 and the number of queries to 50. Since computation cost is spent primarily on EC evaluation, we consider a link failure model, where ECs are reevaluated when the bounding box of a node's subtree changes. Figure 11 (resp. 12) shows the communication/computation tradeoff, when the link failure probability in a round is 0.1 (resp. 0.5). In both figures the cost of QueryQuery is constant, since the algorithm performs no EC evaluation. For the hybrid algorithms, as we increase the cost ratio, the relative importance of computation cost decreases, and thus the overall cost decreases. The decrease is more pronounced for HybridBasic, which performs EC computation at all nodes. For cost ratio less than 50 in Figure 12, HybridWithThreshold is cheaper than HybridBasic because it avoids expensive EC computation at some nodes. However, as the cost ratio increases, HybridBasic becomes more efficient - it uses ECs at all nodes to reduce the communication cost, at the expense of little computational effort.

D. *Experiment with real data*

In this section we compare the communication cost of our preferred algorithm, HybridBasic, with the NoOptimization algorithm, by drawing data from a real network setting. Data is collected from 54 sensors deployed in the Intel Berkeley Research lab in a 31 by 41 area (Figure 13). Sensors are equipped with weather boards that allow them to measure humidity, temperature, light and voltage values once every 31 seconds. The cost of the different techniques was measured as the average result propagation cost over 40 rounds (of one epoch). Every round lasts for 30 minutes and a node reports an update at the end of a round, if its new value differs from the previously reported value by more than a given *update percentage*. In our experiment, we vary the *update percentage* from 0 to 50% (x-axis in Figure 14). We posed 30 queries, each of which asks for the average temperature reading within a rectangular area. The query regions were selected to reflect the boundaries of different rooms (areas) of the Lab. For an update percentage threshold of 5%, HybridBasic has a significant advantage over the existing NoOptimization approach.

VI. Related work

We classify the related work into two categories: query processing and communication protocols.

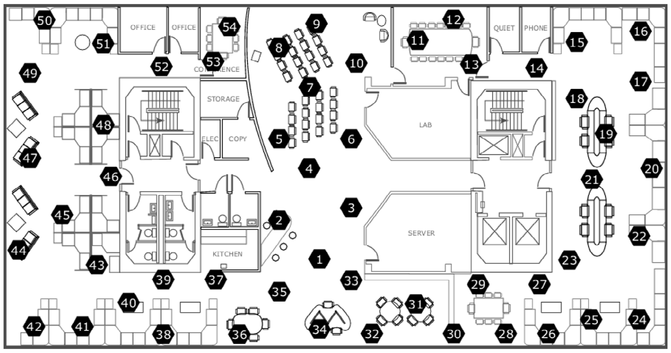

Fig. 13. Network setup in the Intel Berkeley Research Lab.

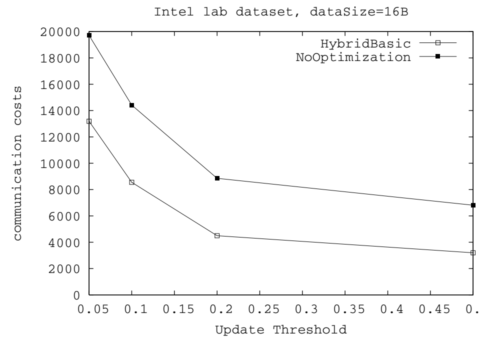

Fig. 14. Cost of HybridBasic on real data.

Query processing in sensor networks. Several research groups have focused on in-network query processing as a means of reducing energy consumption. The TinyDB Project at Berkeley investigates query processing techniques for sensor networks including an implementation of the system on the Berkeley motes and aggregation queries [1], [2], [3], [4], [5]. An acquisitional approach to query processing is proposed in [8], in which the frequency and timing of data sampling is discussed. The sensor network project at USC/ISI group [9], [10] proposes an energy-efficient aggregation tree using data-centric reinforcement strategies (directed diffusion). A two-tier approach (TTDD) for data dissemination to multiple mobile sinks is discussed in [11]. An approximation algorithm for finding an aggregation tree that simultaneously applies to a large class of aggregation functions is proposed in [12]. Duplicate insensitive skethches for approximate aggregate queries are discussed in [13], [14]. Our study differs from previous work in that we consider multi-query optimization for sensor networks.

Communication protocols for sensor networks. The data dissemination algorithms that we study in this paper are all aimed at minimizing energy consumption, a primary objective in communication protocols designed for sensor (and ad hoc) networks. A number of MAC and routing protocols have been proposed to reduce energy consumption in sensor networks [15], [16], [17], [18], [19], [20], [21], [22] While these studies consider MAC and routing protocols for arbitrary communication patterns, our study focuses on multi-query optimization to minimize the amount of data.

VII. Conclusions and Future Work

Our work addresses several issues in the area of Sensor Databases. We have introduced two major extensions to the standard model of executing a single long-running query: A workload of multiple aggregate queries and a workload of sensor data updates. We have given efficient algorithms for multi-query optimization, and tested their performance in several scenarios. To the best of our knowledge this is the first work to *formally* examine the problem of multi-query optimization in sensor networks.

The main conclusions drawn in this paper are the following: First, the notion of equivalence class (EC) is important for distributed query evaluation: encoding sensor updates in terms of ECs enables better compression of the result messages. Second, the result data size is minimized for a certain class of aggregate queries (sum, count and avg) by applying the linear reduction technique in a distributed manner. Third, in applications where the computationally expensive task of linear reduction is infeasible for the sensor nodes, a very good approximation of the optimal can be obtained by having each node select an appropriate local data encoding strategy. This local encoding strategy can itself be defined in terms of a threshold that specifies the computational limitation.

There are a number of directions for further research. First, we would like to extend our ideas to a wider class of aggregation functions. In this regard, fast approximate basis calculations for *min* queries may lead to more efficient processing of these queries. Second, our paper has focused on accurate query evaluation. It would be worthwhile to study approximate query processing and obtain error-energy tradeoffs. We would also like to adapt our techniques to multi-path aggregation methods that provide more fault-tolerance.

REFERENCES

[1] Hellerstein, J., Hong, W., Madden, S., Stanek, K.: Beyond average: Towards sophisticated sensing with queries. In IPSN. (2003)
[2] Madden, S., Franklin, M.: Fjording the stream: An architecture for queries over streaming sensor data. In ICDE. (2002)
[3] Madden, S., Franklin, M., Hellerstein, J., Hong, W.: Tag: A tiny aggregation service for ad-hoc sensor networks. In OSDI. (2002)
[4] Madden, S., Hellerstein, J.: Distributing queries over low-power wireless sensor networks. In SIGMOD. (2002)
[5] Madden, S., Szewczyk, R., Franklin, M., Culler, D.: Supporting aggregate queries over ad-hoc sensor networks. In WMCSA. (2002)
[6] Yao, Y., Gehrke, J.: The Cougar approach to in-network query processing in sensor networks. Sigmod Record **31** (2002)
[7] Yao, Y., Gehrke, J.: Query processing in sensor networks. In CIDR. (2003)
[8] Madden, S., Franklin, M., Hellerstein, J., Hong, W.: The design of an acquisitional query processor for sensor networks. In SIGMOD. (2003)
[9] Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., Ganesan, D.: Building efficient wireless sensor networks with low-level naming. In SOSP. (2001) 146–159
[10] Heidemann, J., Silva, F., Yu, Y., Estrin, D., Haldar, P.: Diffusion filters as a flexible architecture for event notification in wireless sensor networks. In ISI-TR-556, USC/ISI. (2002)
[11] Ye, F., Luo, H., Cheng, J., Lu, S., Zhang, L.: A two-tier data dissemination model for large-scale wireless sensor networks. In MOBICOM. (2002)
[12] Goel, A., Estrin, D.: Simultaneous optimization for concave costs: Single sink aggregation or single source buy-at-bulk. In SODA. (2003)
[13] Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In ICDE. (2004)
[14] Nath, S., Gibbons, P.: Synopsis diffusion for robust aggregation in sensor networks. In SENSYS. (2004)
[15] Heinzelman, W., Wendi, R., Kulik, J., Balakrishnan, H.: Adaptive protocols for information dissemination in wireless sensor networks. In MOBICOM. (1999)
[16] Johnson, D., Maltz, D.: Dynamic source routing in ad hoc wireless networks. In: Mobile Computing. Volume 353 of The Kluwer International Series in Engineering and Computer Science. (1996)
[17] Perkins, C.: Ad hoc on demand distance vector (aodv) routing. (Internet Draft 1999, http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-04.txt)
[18] Perkins, C., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In SIGCOMM. (1994) 234–244
[19] Xu, Y., Bien, S., Mori, Y., Heidemann, J., Estrin, D.: Topology control protocols to conserve energy inwireless ad hoc networks. In TR6, UCLA/CENS (2003)
[20] Xu, Y., Heidemann, J., Estrin, D.: Geography-informed energy conservation for ad hoc routing. In MOBICOM. (2001) 70–84
[21] Ye, W., Heidemann, J., Estrin, D.: An energy-efficient MAC protocol for wireless sensor networks. In INFOCOM. (2002) 1567–1576
[22] Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated, adaptive sleeping for wireless sensor networks. In ISI-TR-567, USC/ISI. (2003)