Routing and Processing Multiple Aggregate Queries in Sensor Networks

Niki Trigoni Alexandre Guitton Antonios Skordylis

Abstract

Tree-based routing is an energy-efficient technique for evaluating aggregate queries in sensor networks, since it enables partial aggregation of results in their way to the root. Tree construction is currently performed using simple flooding algorithms, data-centric reinforcement strategies or energy-aware route selection schemes.

This paper presents a query-driven approach to routing and processing continuous aggregate queries in sensor networks. Given a query workload and a special-purpose gateway node where results are expected, the query optimizer exploits query correlations in order to generate an energy-efficient distributed evaluation plan. The proposed optimization algorithms identify common query sub-aggregates, and propose common routing structures to share the sub-aggregates at an early stage. Moreover, they avoid routing sub-aggregates of the same query through long-disjoint paths, thus further reducing the communication cost of result propagation. The proposed algorithms are fully-distributed, and are shown to offer significant communication savings compared to existing tree-based approaches. A thorough experimental evaluation shows the benefits of the proposed techniques both in real and simulated sensor networks, for a variety of query workloads and network topologies.

1 Introduction

Recent advances in micro-electro-mechanical systems (MEMS) have enabled the inexpensive production and deployment of nodes with communication, computation, storage and sensing capabilities. Sensor nodes can be deployed in large areas to monitor the ambient environment, and they communicate their readings to one or more basestations (referred to as *gateways*) in a wireless multihop manner. Sensor network applications range from habitat monitoring, to security, surveillance and emergency scenarios. Sensors are also envisaged to play an important role in industry, by providing asset management services and product tracking.

In most of the aforementioned application scenarios, the typical way of extracting information from a sensor network is to disseminate declarative aggregate queries into the network, asking nodes to periodically monitor the environment, and return aggregate results to the gateway in regular rounds. An example of such long-running queries is *"select avg(temperature) from Sensors where loc in Region every 10 min"*.

Since nodes are battery-powered, energy preservation is a major consideration in system design, as it directly impacts the lifetime of the network. Recent studies have shown that radio communication is significantly more expensive than computation or sensing in most existing sensor node platforms. Hence, the main consideration in designing query processing algorithms is to minimize the communication overhead of forwarding query results from the sources to the gateway node. The cost of disseminating query information into the network is assumed to have a secondary role for two reasons: Firstly, it is realistic to assume that the gateway is equipped with a long-range radio that is able to broadcast query requests to the entire network. The gateway accumulates queries for a short duration and sends them together for evaluation during an interval dedicated for query dissemination. During that interval, nodes pause other activities to avoid causing interference and loss of query request messages. Secondly, in the case of long-running queries, query dissemination occurs once, whereas result propagation occurs repeatedly at regular rounds. Moreover, many monitoring scenarios apply a pure push model, in which nodes are programmed to proactively send specific information to the gateway. The communication cost of result propagation thus dominates the communication cost of query dissemination.

Tree-based routing has been proposed as an energy-efficient mechanism for processing aggregate queries in sensor networks [10, 12]. Tree construction is performed using simple flooding algorithms [12], data-centric reinforcement strategies [10] or energy-aware route selection schemes [23, 26]. After a tree is constructed, sensor nodes forward their readings along the paths of the tree, evaluating partial query results at intermediate nodes. The aforementioned research focused on processing a *single aggregate query given a routing tree*; the tree is generated using a tree selection scheme and is thereafter used for result propagation. More recent re-

search has focused on *optimizing multiple aggregate queries given a routing tree* [21]. Query commonalities are taken into account to reduce the communication cost of result propagation, but without making any attempt to select suitable tree routes [21].

Unlike previous approaches, this paper considers the more general problem of multi-query optimization lifting the assumption of an existing aggregation tree. The objective is to find efficient routes that minimize the communication cost of executing multiple aggregate queries, by studying the interplay between the processing and routing aspects of query evaluation. Unlike previous work, there is no limitation for the selected routes to form a tree structure. The only requirement is that the optimizer must operate in a distributed manner, and should scale gracefully with the network size. In summary, the contributions of this paper are as follows:

- A demonstration of the interplay between the processing and routing aspects of single- and multi-query optimization (Section 2)
- A formal definition of the multi-query optimization problem for aggregate queries (Section 3), which lifts the assumption of a communication tree used in previous work [10, 12, 21].
- Complexity results for the routing and processing aspects of multi-query optimization (Section 4 and Appendix A).
- Two novel heuristic algorithms, named SegmentToGateway (STG) and SegmentToSegment (STS) for optimizing multiple aggregate queries (Section 5). Existing query evaluation algorithms use tree routes constructed independent of the query workload. Given a tree, they focus on in-network partial processing of one query [12, 10] or multiple queries [21]. STG and STS are the first algorithms that attempt to select suitable routes for a workload of multiple queries, and carefully interweave routing and processing decisions in the optimization process.
- A rich set of experimental results that compare the performance of the proposed algorithms with the most efficient existing algorithm for multi-query optimization [21] (Section 6). The benefits of STG and STS are demonstrated under a variety of query workloads and network topologies, both in terms of network-wide communication cost and in terms of local communication cost in the critical area around the gateway.

2 Illustrative examples

The potential advantages of carefully selecting a routing and processing plan for executing aggregate queries are shown in the following examples. Figure 1 shows an example of processing a single aggregate query, which asks for the sum of all readings in the dotted rectangular area. Notice that a total number of 15 messages are sent along the left minimum-hop tree of Figure 1, whereas only 6 messages are forwarded along the carefully selected right tree of the same figure. The right routing tree is better not only in terms of total communication cost, but also in terms of communication cost in the critical area around the gateway. Informally, the benefit of the second plan is that it aggregates all readings of a query early and avoids sending different subaggregates through disjoint paths.

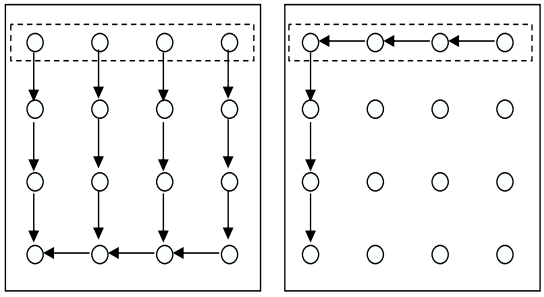

Figure 1. Example with one query

Figure 2 illustrates the benefits of building a suitable execution plan in the case of processing multiple *count* queries. For ease of understanding the graphs also include node ids and messages forwarded through network links. Messages have the format $v(q_1,\ldots,q_n)$, which denotes that value v contributes to queries $q_1,\ldots,q_n$. The left plan does not exploit query commonalities, and therefore fails to aggregate together readings (of nodes 8 and 9) within the intersection area. The middle plan incurs smaller communication cost, because it exploits query commonalities, but still forwards the subaggregate of the intersection area separately all the way to the gateway. This behavior is similar to the first heuristic proposed in this paper called SegmentToGateway (STG). The right plan has optimal behavior because it exploits query commonalities and it avoids sending partial aggregates through long disjoint paths. Notice that the optimal plan does not follow a tree structure, as node 8 sends the partial aggregate of the intersection area to two parents. The intersection partial aggregate is thus merged immediately with the other two query subaggregates and, eventually, only two partial results are sent to the gateway. This would be the plan identified by the second proposed algorithm, called SegmentToSegment (STS).

Although the examples above use a grid topology, both STG and STS are designed to work well for random topologies with potential empty areas (or holes). The two algorithms will be explained in detail in Section 5 and they will be compared with the existing approach of randomly choosing a minimum hop tree.

3 Problem definition

Sensor deployment: Let us consider a set of sensor nodes $S=\{s_1,\ldots,s_n\}$ located at positions $\{(x_1,y_1),\ldots,(x_n,y_n)\}$ respectively. Two nodes capable of bi-directional wireless communication are referred to as *neighbors*. Every node knows its location, as well as the identifiers and locations of its neighbors.

Query workload: We first define a *general* class of aggregate queries, and then focus on a subclass that is particularly useful for many sensor network applications. The *general* class includes queries of the form $aggr(S')$, where $S'\subseteq S$ and *aggr* is a summary aggregate function (e.g. $sum, count, avg, max, min$) [12]. A commonly used sub-

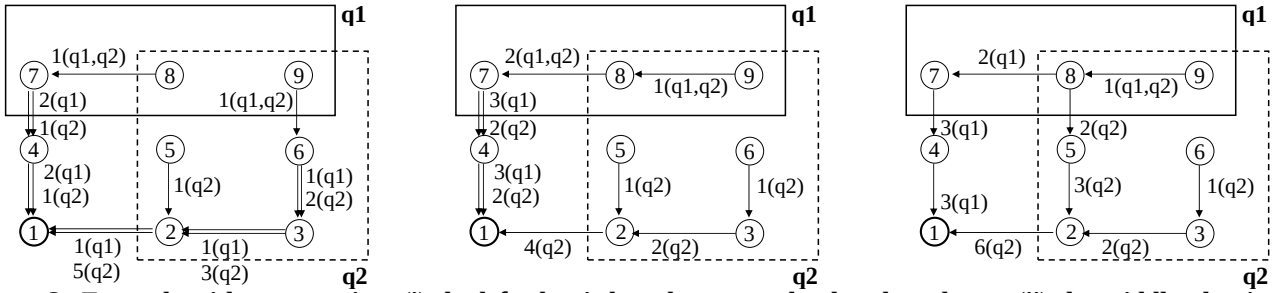

Figure 2. Example with two queries: (i) the left plan is based on a randomly selected tree, (ii) the middle plan is the output of STG, and (iii) the right plan is the output of STS.

class of aggregate queries consists of spatial range queries (SRQs); SRQs evaluate the aggregate *aggr* of all sensors in a rectangular area. They are denoted by a tuple $(aggr, x_0, y_0, x_{dim}, y_{dim})$, where x_0 and y_0 denote the bottom left coordinates of the rectangular area and x_{dim} and y_{dim} denote the area's x and y dimensions respectively. Let $Q = [q_1, \ldots, q_m]$ be the vector of SRQ queries that have been gathered for execution at the gateway $G \in S$. Queries that evaluate the same aggregate function over different sensor sets (or regions) are grouped together for periodic evaluation for a large number of rounds. Each node knows the identifiers (q_i) and descriptions of queries that *cover* itself and its neighbors.

Computation: Nodes can process values with negligible cost. A node is aware of its own sensor value, as well as the partially processed values received from its neighbors. We refer to these values as input values. A node processes the input values taking into account how they contribute to the query results, and converts them into output values. The contribution of a value (either input or output) to the query results is referred to as *semantics*. In this paper, the propagation of data across an edge will be represented as a directed edge, labeled with the pair (value, semantics). For uniformity, the generation of a reading locally at a node is also represented as a directed edge (pointing to the node, but with a dangling starting point). Such dangling edges are referred to as *initial directed edges* and will be drawn in bold.

Let u_i be the sensor reading generated locally at a node s_i. The semantics of u_i consists of the set of queries that access the particular node, and is represented as a bit vector of size m (equal to the number of queries). The j-th entry of the vector is 1, if query q_j accesses node s_i, and is 0 otherwise. Vectors that determine the contribution of a value to the queries are referred to as *coefficient* vectors (CVs). For example, in the evaluation plan depicted in figure 3, the initial directed edge (dangling edge in bold) of node s_2, which holds information about the locally generated reading, is labeled $(1, [110])$ to denote that the local sensor value 1 contributes to the queries q_1 and q_2, and does not contribute to the value of q_3. The reader should note that the result of a query q_j must be equal to the aggregate of all values of initial directed edges factored by their j-th coefficients, i.e. $Result(q_j) = aggr_{i=1}^{n}(u_i * CV_{u_i}[j])$, where n is the number of sensor nodes.

Communication: As the initial (value,semantics) pairs are pushed towards the gateway node, they can be partially processed at intermediate nodes through which they are routed. The intermediate values are also annotated with coefficient vectors denoting their contribution to the final query results. In a candidate plan, (i) each output (value,semantics) pair of a node is computable as some function of the input pairs; and (ii) it should be possible to evaluate all query results based on the gateway's input pairs.

As queries are evaluated periodically over multiple rounds, the propagated values can potentially change at every round. Depending on whether we also allow the semantics of these values to change, we distinguish two different models of value propagation: static and dynamic. In the *static* model, the semantics remains unchanged at each edge, and it is propagated through the directed edge only once, as opposed to the corresponding value, which is propagated at every round. In the dynamic model, values must always be annotated with their semantics, to reflect changes due to network dynamics or variable sensor updates.

The assumption in this paper is a dynamic model in which a constant number of bits is dedicated to storing the semantics (CV - Coefficient Vector) of a value. For scalability, a node can use CVs of variable length to mark only queries affected by the values forwarded through the node. For simplicity, however, we consider a fixed CV length (equal to the number of queries m) for all nodes, and defer the study of compressing CVs to future work.

Content Preservation Principle: Let $InAnnot = [(v_1, CV_{v_1}), \ldots, (v_k, CV_{v_k})]$ be the labels of the input edges and $OutAnnot = [(v'_1, CV_{v'_1}), \ldots, (v'_\ell, CV_{v'_\ell})]$ be the labels of the output edges of a sensor node. In any query execution plan, there should be no loss of information as data is routed through a node. Informally, the result of a query when evaluated based on the input edges must be equal to its result based on the output edges. Formally, $\forall j = 1, \ldots, m,\ aggr_{i=1}^{k}(v_i * CV_{v_i}[j]) = aggr_{i=1}^{\ell}(v'_i * CV_{v'_i}[j])$. We call $aggr_{i=1}^{k}(v_i * CV_{v_i}[j])$ the *projection* of query q_j onto $[(v_1, CV_{v_1}), \ldots, (v_k, CV_{v_k})]$. The projection of a query onto the input edges of a node must be equal to its projection onto the output edges. This principle is satisfied in the network of Figure 3.

THEOREM 1. *If every node in the graph satisfies the content preservation principle except for the gateway, then the values of all queries in the workload are given by the annotated input edges of the gateway node. More specifi-*

cally, if the gateway has k input edges labeled with the pairs $(v_1, CV_{v_1}), \ldots, (v_k, CV_{v_k})$*, then the value of a query* q_j *is* $Result(q_j) = aggr_{i=1}^{k}(v_i * CV_{v_i}[j])$*. The proof is omitted for space reasons*[1].

Optimization goal: Based on the observation about content preservation, the problem at hand is defined as follows: *Start with a graph that consists of all sensor nodes and one directed dangling edge per node, carrying its source value. Minimize the number of directed edges that we need to add in the graph (excluding the initial dangling edges) such that the content preservation principle is satisfied at each node.*

4 Processing and Routing: Complexity

The previous definition reveals two important aspects of the problem, processing and routing, and it suggests a strong interaction between the two. Regarding the *processing aspect*, every node with given input (value,CV) pairs must decide how to select output (value,CV) pairs whilst satisfying the content preservation property. The goal is to minimize the number of directed edges overall in the graph. This does not mean that we must necessarily minimize the number of output edges at every node. For example, node s_8 in Figure 5 selects more output edges than input edges in the optimal solution (counting input edges with distinct CVs). Regarding the *routing aspect*, each node with given output (value,CV) pairs must decide where to send them, i.e. where to direct the corresponding output edges.

Appendix A includes complexity results about the processing and routing aspects, when the two aspects are viewed independently, and ignoring their interaction. It summarizes the previous result that, given fixed tree routes, the processing aspect has polynomial cost for algebraic aggregate (sum, avg and count) queries, but is NP-hard for min (max) queries [21]. In addition, it presents a new proof that the routing aspect is NP-hard for both classes of aggregates. The problem of jointly making processing and routing decisions to minimize communication cost is therefore also NP-hard.

5 Algorithms

Given the problem complexity, the next step is to study the existing approach for processing aggregate queries, and attempt to improve its performance by proposing two novel energy-efficient algorithms. All three algorithms consist of two phases: (i) a network configuration phase and (ii) a result propagation phase. The role of the network configuration phase is not to propagate queries, but to prepare the ground for routing query results (although the two tasks are often combined by piggybacking queries to network configuration messages). This paper does not focus on query propagation as discussed in Section 1. The result propagation phase is divided into regular rounds, the frequency of which is indicated by the query definitions.

Before presenting the three algorithms in detail, it is worth providing some intuition about their main differences. The existing state-of-the-art in optimizing multiple aggregate queries is the ECReduced algorithm proposed in [21]; it outperforms Tag [12] and Cougar [24] in the context of multiple queries, since these approaches were originally designed to process a single query, as shown in [21]. ECReduced is therefore a good basis for comparing the two proposed algorithms. In this paper, it is hereafter referred to as *NoOptimization*, to denote that it does not jointly optimize routing and processing taking into account the query workload. NoOptimization uses a predefined tree, and only optimizes the processing aspect of query execution.

The first proposed heuristic, named *SegmentToGateway (STG)* exploits query commonalities, by identifying common query sub-aggregates and building common routing structures to share the sub-aggregates at an early stage. The second proposed heuristic, named *SegmentToSegment (STS)* further exploits query knowledge by ensuring that subaggregate values of the same query are merged early, instead of being dispersed towards the gateway through disjoint paths. The remainder of this section presents in detail the network configuration and result propagation phases of the three algorithms.

5.1 The *NoOptimization* algorithm

Existing approaches to processing aggregate queries first build a spanning tree, which links each node to the gateway through the best-quality path, and then partially aggregate query results at intermediate nodes of the tree. Path quality can be determined in different ways, the simplest of which is to select the shortest (minimum-hop) path to the gateway ([12, 21]). Directed diffusion uses data-centric reinforcement strategies, that can be tuned to choose paths based on their delay or consistency [10]. Energy-aware algorithms suggest the use of paths that avoid nodes with low energy reserves [23, 26].

This subsection includes a description of the ECReduced algorithm, which is proposed in [21] and is referred to as NoOptimization in this paper. Details of how it selects a communication tree and how it processes and propagates query results are given below:

Network Configuration Phase: Control messages are first flooded into the network, and every node selects as its *parent* the neighbor in the shortest path to the gateway node. If there are more than one candidate parents, the node selects its parent in either of the following ways: (i) randomly, (ii) the first node from which it received a query request, (iii) the node with which it consistently maintains better communication. In the experimental evaluation of Section 6, *NoOptimization* is implemented as in [21], i.e. breaking ties by random parent selection. Dynamic node or link failures are handled by a local flooding phase to repair affected tree routes, as in AODV [1].

Result Propagation Phase: The routes of query results are predefined in the network configuration phase, and the only decision that a node needs to make in this phase is how to convert its input (value,CV) pairs into output pairs. All output pairs, irrespective of their content, are forwarded to the node's *parent*. A naive application of the in-network aggregation technique to processing multiple queries would be to forward one partial aggregate value per query, and denote the query identifier in the coefficient vector.

The NoOptimization algorithm uses a more elaborate technique to reduce the number of propagated (value, CV)

[1]The 2-paragraph proof will be included at the reviewers' request

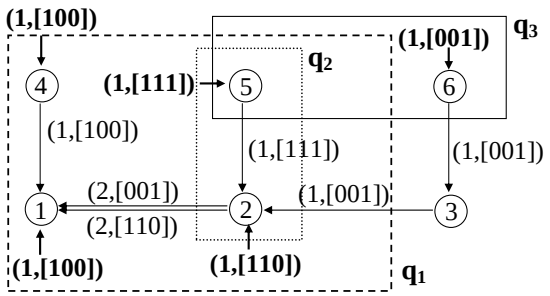

Figure 3. NoOptimization: Node 2 linearly reduces the three input (value,CV) pairs into two output pairs

pairs. In the case of algebraic aggregate functions, like *sum*, *count* or *avg*, a node running NoOptimization computes a basis of its input coefficient vectors and sends to its *parent* the basis vectors as output CVs [21] (along with corresponding values). An example of the effect of linear reduction is shown in Figure 3, where node 2 receives three input (value,CV) pairs and reduces them to two output pairs. The linear reduction technique yields the optimal solution for these aggregates in terms of communication cost. The implementation of *NoOptimization* in Section 6 is identical to that of ECReduced in [21]. The NoOptimization algorithm, which is used as a basis for comparison, is to our knowledge the most sophisticated existing approach to processing multiple algebraic aggregate queries.

5.2 The *SegmentToGateway (STG)* algorithm

The first proposed heuristic algorithm exploits the fact that the intersecting query rectangles naturally divide the network into smaller segments. A definition of *segment* and of other important concepts for the understanding of STG are given below:

- A *segment* (abbreviated SG) S is a maximal set of nodes, s.t. $\forall s_i \in S, s_j \in S$, s_i and s_j are covered by the same set of queries and they are *internally connected*, i.e. there exists path from s_i to s_j consisting only of nodes in S. The queries in Figure 5 form five segments $\{s_1, s_4\}$, $\{s_2\}$, $\{s_3, s_5, s_6\}$, $\{s_7\}$ and $\{s_8, s_9\}$.
- The *SGVector* of a segment is a bit vector that denotes which queries cover the segment nodes, e.g. $SGVector(\{s_3, s_5, s_6\}) = [010]$. The *SGVector* of a node $s \in S$ is equal to the *SGVector* of S, e.g. $SGVector(s_3) = [010]$.
- The *SGLeader* of a segment S is selected to be the node in S with the minimum-hop path to the gateway node. Ties are broken by selecting the node with the smallest x coordinate, and if those are equal the node with the smallest y coordinate (e.g., $SGLeader(\{s_3, s_5, s_6\}) = s_5$). The *SGLeader* of a node $s \in S$ is equal to the *SGLeader* of S.
- The *SGDistance* of S denotes the number of hops from the *SGLeader* of S to the gateway (e.g., $SGDistance(\{s_3, s_5, s_6\}) = 2$). The *SGDistance* of a node $s \in S$ is equal to the *SGDistance* of S.
- The *distToSGLeader* of a node in S denotes the number of hops from this node to the *SGLeader* of S (e.g., $distToSGLeader(s_6) = 1$).

```
event TOS_MsgPtr RcvBeacon.rcv(TOSMsgPtr m)
{
 bool mustRebroadcastBeacon = FALSE;
 BeaconMsg* b = (BeaconMsg*)m → data;
 addBeaconSenderToNeighbors(b);
 if (b → hopCount + 1 < hopCount) {
    mustRebroadcastBeacon = TRUE;
    hopCount = b → hopCount + 1;
    parent = b → source;
 }
 if (equalVectors(SG, b → SG)) {
  if ((SGDistance > b → SGDistance)
   || (SGDistance == b → SGDistance &&
   b → distToSGLeader + 1 < distToSGLeader)
    || (SGDistance == b → SGDistance &&
   strictlyCloser(b → leaderLoc, leaderLoc)){
       mustRebroadcastBeacon = TRUE;
       SGParent = b → source;
       SGDistance = b → SGDistance;
       distToSGLeader = b → distToSGLeader + 1;
        leaderLoc = b → leaderLoc;
 }} else {// not equal vectors
   if ((b → SGDistance > hopCount)
    || (b → SGDistance == hopCount &&
    b → source == parent &&
    b → source! = SGParent &&
    closer(myLoc, b → leaderLoc))){
       mustRebroadcastBeacon = TRUE;
       SGParent = b → source;
       SGDistance = hopCount;
       distToSGLeader = 0;
       leaderLoc = myLoc;
}}}
```

Figure 4. NesC code for the network configuration phase of STG (excl. lines in bold) and STS (incl. lines in bold)

Network Configuration Phase: This is similar to the corresponding phase of the *NoOptimization* algorithm. Beacon messages are flooded to establish the shortest path from each node to the gateway. In addition to the *hopCount* value beacon messages include the following state information of the sender node: (i) the *SGVector*, (ii) the location of the currently known *SGLeader*, and (iii) the currently known *SGDistance*. The reason for flooding such beacons is for each node to identify a parent node (as in NoOptimization), and, in addition, a SGParent (i.e. a neighbor in a path to the SGLeader). The path to the SGLeader does not need to be the shortest one, as long as it has no cycles and all intermediate nodes are in the same segment.

The implementation of the network configuration phase is depicted in Figure 4. The reader should ignore the lines in bold that are only pertinent to the description of STS. Upon receiving a beacon message, a node updates the local list of neighbors and, if necessary, the *hopCount* value (as in NoOptimization). The next step depends on whether the beacon is sent from a node in the same or in a different segment. In the former case, the node compares the local knowledge about the *SGLeader* with that in the beacon. If the beacon knows of a *SGLeader* closer to the gateway (with smaller *SGDistance*), the local *SGDistance* value is updated and the sender node is selected to be the local *SGParent*. In the latter case

where a node receives a beacon from a node in a different segment, it realizes that it is on the border of the segment and thus it is eligible to become a *SGLeader*. It elects itself to be a *SGLeader* if its *hopCount* is smaller than the local *SGDistance*. The beacon message is updated accordingly and is rebroadcasted.

Experiments showed that the difference between the cost of the network configuration phase of STG and that of NoOptimization is negligible (Section 6). An advantage of STG is that insertion (or deletion) of queries has a local effect on segment formation, and does not require global network reconfiguration. Dynamic network failures can be handled by adjusting the local repair mechanism used by NoOptimization to also update the segment-related state variables of nodes.

Result Propagation Phase: The goal of this phase is to partially aggregate all nodes within the same segment, gather the partial aggregate at the *SGLeader* node and forward it to the gateway through the shortest path. This works as follows: By the end of the network configuration phase, every node knows its parent and *SGParent*. In the result propagation phase, a node merges duplicate input CVs into the same output CV, aggregating values accordingly. An output (value,CV) pair is sent to the *SGParent* if and only if the CV is equal to the current node's *SGVector*. The remaining output (value,CV) pairs are forwarded to the *parent* node (after they have been linearly reduced in the case of algebraic aggregates). The neighbors of the gateway send all their messages without exception directly to the gateway.

Discussion: To summarize, STG identifies query commonalities (segments) and aggregates the values of all nodes within each segment separately following a mini-tree rooted at the *SGLeader*. The remaining values (whose CVs are not equal to the *SGVector*) are forwarded through the *parent* node (instead of the *SGParent*) and reach the gateway through the shortest path. By definition, STG performs better than NoOptimization. Although STG performs well in terms of merging readings of the same segment, it takes no action towards efficiently merging sub-aggregates of the same query that come from different segments. In the worst case, these sub-aggregates may be propagated from the *SGLeader* nodes to the gateway through long disjoint paths.

5.3 The *SegmentToSegment (STS)* algorithm

STS addresses the weakness of STG by sending messages towards neighbors that are likely to *reduce* them. STS manages to combine *segment-based* aggregation (introduced in STG), with merging of CVs from different segments, into a uniform mechanism.

Network Configuration Phase: The configuration phase of *STS* is similar to the corresponding phase of *STG* with one additional feature. In the flooding process, a node aims to select as its *SGParent* the node in the shortest path to the *SGLeader*. The number of hops from a node to its *SGLeader* is recorded in the local variable *distToSGLeader*. When a *SGLeader* broadcasts a message it sets the value of *distToSGLeader* to 0. Upon receiving a beacon, a node compares its local *distToSGLeader* with the corresponding beacon value (incremented by one). If the local value is greater than the incremented beacon value, then the *SGParent* becomes the sender of the beacon and the local *distToSGLeader* is updated with the beacon value plus one (see lines in bold in Figure 4). Query workload updates and network failures are handled as in STG.

Result Propagation Phase: STS aggregates values within each segment first (like STG) using a uniform routing mechanism, instead of forwarding segment-related messages to a special-purpose *SGParent* node. STS takes one step further to ensure that the segment aggregates are not propagated to the gateway through disjoint paths, but they are efficiently merged at an early stage. The steps of STS are described below:

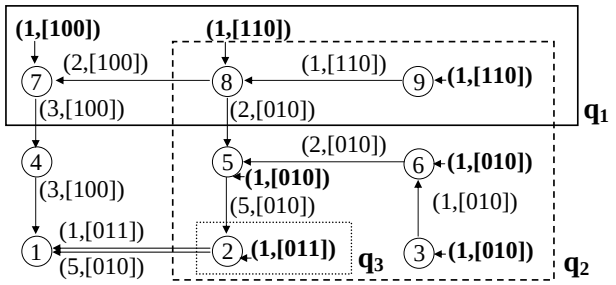

Figure 5. Value-semantic pairs in the evaluation plan of the sum queries q_1, q_2 and q_3

Step 1: Convert input to output (value,CV) pairs, by merging duplicate CVs and aggregating corresponding values - or by means of linear reduction in the case of algebraic aggregates. For example, in Figure 5 the input messages of s_8 are merged into the pair (2,[110]). This processing step is also part of NoOptimization and STG. In the latter algorithms it is followed by a well-defined routing step, namely NoOptimization forwards all output pairs to the *parent* node, whereas STG forwards them to either the *SGParent* or the *parent* node. The distinctive feature of STS is that it continuously interweaves processing and routing decisions as shown in the following steps.

Step 2: Sort the output (value,CV) pairs, based on the number of queries (1-bits) in their CVs in descending order. Ties are broken by ordering CVs in lexicographic order.

Step 3: Neighbor-message matching: Pop the first (value,CV) pair p from the ordered list (the one contributing to most queries) and select the best neighbor to forward it. The selection process is an elaborate algorithm described later on in the section.

Step 4: Message splitting: Split the pair p into two pairs p_1 and p_2, based on the *SGVector* of the selected neighbor. Assume that node s_8 chooses s_7 to forward $p = (2, [110])$ in Step 3. Notice that the CV of p has more queries (q_1 and q_2) than the SGVector of the selected neighbor ($SGVector(s_7) = [100]$ denotes that s_7 is covered only by q_1). In this case, p is split into two pairs, one contributing to the common queries $p_1 = (2, [100])$, and another contributing to the remaining queries $p_2 = (2, [010])$.

Step 5: Message dispatching: Send p_1 to the selected neighbor and re-insert p_2 into the ordered list of output pairs. Insert p_2 only if it has a non-zero CV vector. During insertion, preserve order (Step 2) and merge pairs with equal CVs. If

the ordered list of (value,CV) pairs is not empty go back to Step 3.

It now remains to describe the neighbor-message matching process, in which a node selects the best neighbor for a (value,CV) pair. It consists of three steps.

Step 3.1: To ensure that messages are not forwarded away from the gateway, only neighbors closer to the gateway than the current node are considered, i.e. with lexicographically smaller (hopCount, SGDistance, distToSGLeader, xCoord, yCoord). For instance, node s_3 considers sending messages to s_2 (Figure 5). As an exception, a node also considers neighbors in the same segment that are not closer to the gateway, if (i) they are closer to their *SGLeader* and (ii) their SGVectors are sub-vectors of the message CV (i.e. queries marked in their *SGVectors* are also marked in the message CV). For instance, s_3 also considers s_6 to forward its initial data $(1,[010])$, because *distToSGLeader*(s_6)<*distToSGLeader*(s_3) and the *SGVector*$(s_6) = [010]$ marks queries $\{q_2\}$ that are all marked in the message CV $[010]$. This exception cannot result in sending messages in cycles or away from the gateway, because the message is merged immediately with the receiving node's local data (input pairs of s_6 have equal CVs).

Step 3.2: Among neighbors selected in Step 3.1, consider only those that best match the message CV, i.e. which are covered by the maximum number of common queries with the message CV. If this number is 0, or the node is next to the gateway, it sends the message to its *parent*. Node s_3 has two candidate neighbors, s_2 and s_6, to send (1,[010]) (from Step 3.1). The *SGVectors* $[011]$ and $[010]$ of s_2 and s_6 both have one common query with the message CV ($[010]$). Among neighbors with equal number of common queries, select the one with the minimum number of queries (s_6).

Step 3.3: Among neighbors selected in Step 2, select the one with the lexicographically smaller (*SGDistance,distToSGLeader,xCoord,yCoord*). For instance, s_8 has two candidate neighbors s_5 and s_7 to send the output pair (2,[110]). Both have *SGDistance* equal to 2 and *distToSGLeader* equal to 0 (both nodes are segment leaders), so s_7 is selected because it has a smaller x coordinate.

Discussion: Although STS is tailored specifically for optimizing spatial aggregate queries, two if its features - namely *neighbor-message matching* and *message splitting* - have broader applicability. The idea behind the first feature is to forward messages towards nodes that are most likely to reduce them (i.e. to merge them with their local or route-thru data). The principle is applicable to other scenarios, for example in the context of optimizing GROUP-BY queries [18], where data is preferentially routed towards nodes that hold data belonging to the same group. The second noteworthy feature of STS is message splitting. Most recent research efforts have focused on merging data to reduce their size. STS's novelty lies in offering further communication savings by means of data splitting. It is often beneficial to divide data into its components in order to give it greater potential for later merging.

By means of careful message routing, merging and splitting, STS ensures that all query subaggregates are merged together before they leave the query area, thus offering significant benefits wrt STG and NoOptimization. The benefits of STS are more pronounced in the critical area around the gateway and increase with the number of nodes ensuring scalability (Section 6).

6 Experimental evaluation

A thorough experimental evaluation was performed to compare the proposed heuristic algorithms with the existing *NoOptimization* approach under a variety of network topologies and query workloads. The implementation was done initially on TinyOS's TOSSIM, but a home-grown simulator was later developed to run many repetitions of large-scale experiments. The conversion from NesC to C++ was relatively easy, as the code performed on Berkeley nodes was transfered to a Node class in C++, and a new Simulator class was developed to pass messages from one node to another, and monitor network traffic. We first present our experiments on our home-grown simulator and then on a real sensor network testbed.

6.1 Simulation experiments

The experimental results below show the performance of the three algorithms varying: (i) the number of queries, (ii) the type of queries, (iii) the number of nodes, (iv) the network area (keeping node density constant), (iv) the radio communication range, and (v) the number of holes (unpopulated areas in the network).

A variety of performance metrics are considered. Most graphs illustrate the communication benefits of STG and STS compared to the existing approach. The benefit of STG is $(cost(NoOptimization) - cost(STG))/cost(NoOptimization)$ and the benefit of STS is defined similarly. It remains to define how the cost of an algorithm is calculated. In each graph two costs per algorithm are considered, the number of messages sent and the number of messages received during result propagation, thus resulting in four different measures of benefit (*STG_Send*, *STS_Send*, *STG_Receive* and *STS_Receive*). Depending on which nodes are monitored, we provide three different types of graphs, those based on counts of messages sent (or received) (i) by all nodes in the network (right), (ii) by nodes at most one hop away from the gateway (left) and (iii) by nodes at most two hops away from the gateway (middle). The figure position and caption indicate whether global or local communication savings are considered.

The default simulation settings are as follows: We deploy 100 nodes uniformly at random in a 300m$\times$300m network area. The radio communication range is set to 60m. The default query workload consists of five rectangular queries with randomly chosen dimensions ($x,y \in [30,300]$). In our experiments below we vary the values of one parameter at a time, keeping the default values for the remaining parameters. Each point in a plot is drawn by averaging 40 repetitions in which we vary the query workload and network topologies within the scope of the experiment's settings.

In the experiments below, the cost of the network configuration phase is very similar for the three algorithms, with NoOptimization sending 4%-10% less messages than STG and STS. This overhead is paid infrequently, and is counterbalanced by the benefits offered by STG and STS during the

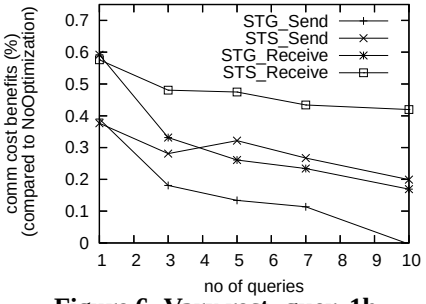

Figure 6. Vary rect. quer. 1h

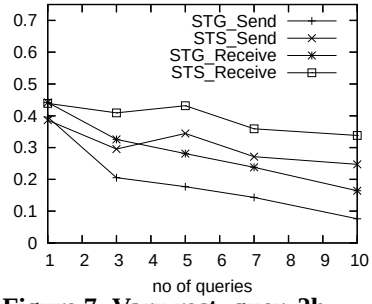

Figure 7. Vary rect. quer. 2h

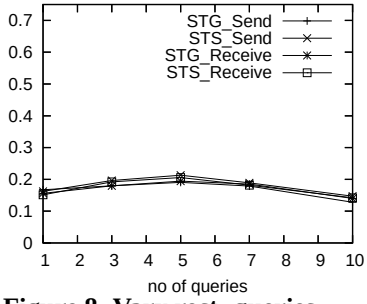

Figure 8. Vary rect. queries

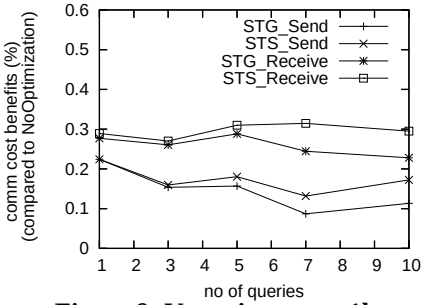

Figure 9. Vary circ. quer. 1h

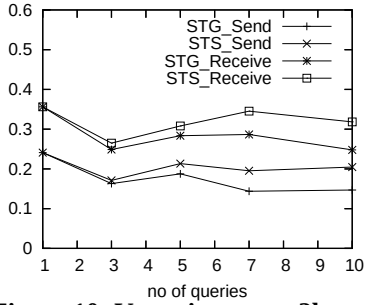

Figure 10. Vary circ. quer. 2h

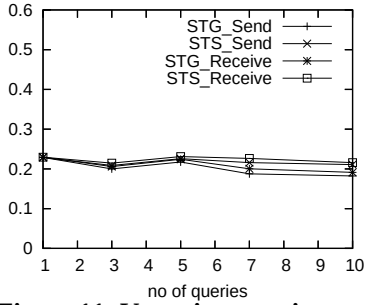

Figure 11. Vary circ. queries

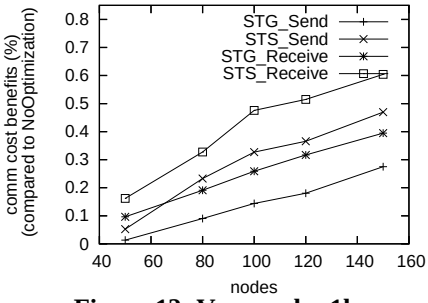

Figure 12. Vary nodes 1h

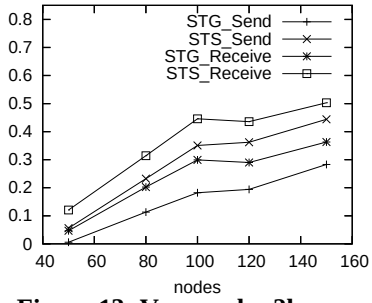

Figure 13. Vary nodes 2h

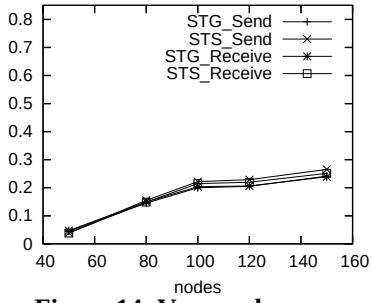

Figure 14. Vary nodes

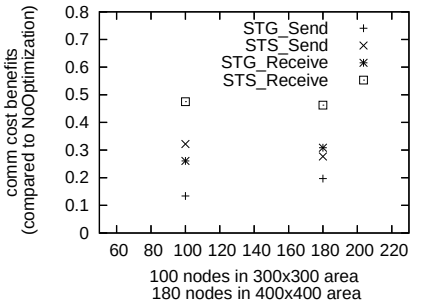

Figure 15. Vary netw. area 1h

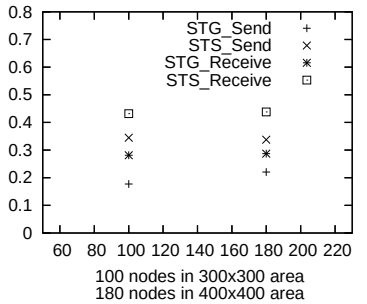

Figure 16. Vary netw. area 2h

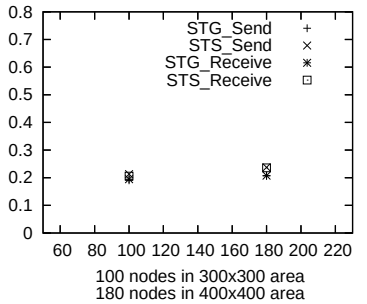

Figure 17. Vary netw. area

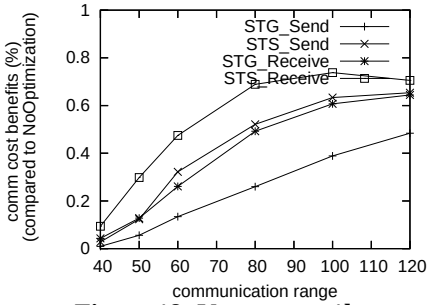

Figure 18. Vary range 1h

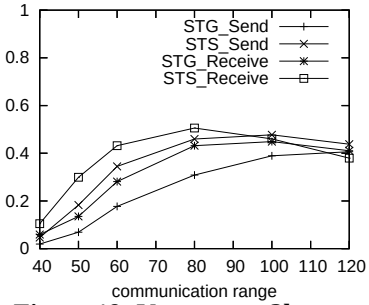

Figure 19. Vary range 2h

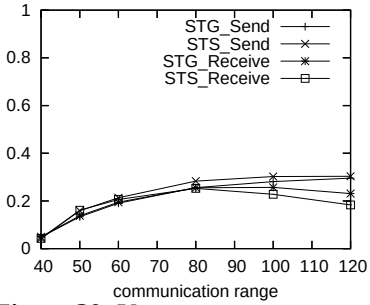

Figure 20. Vary comm.range

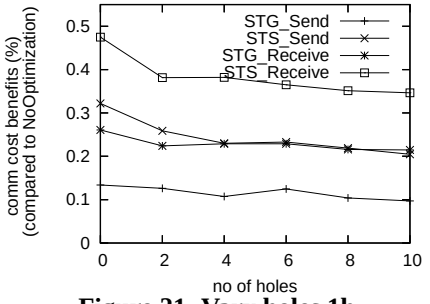

Figure 21. Vary holes 1h

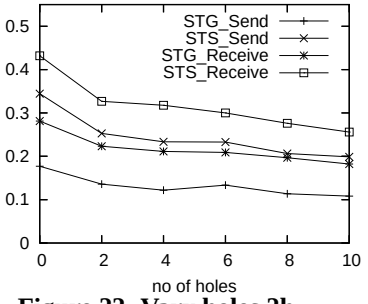

Figure 22. Vary holes 2h

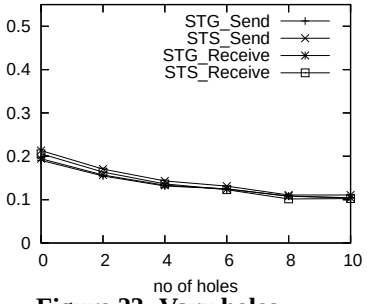

Figure 23. Vary holes

frequent result propagation phase.

Vary number of queries: The first experiment illustrates the effect of the number of rectangular queries (sent together to the network for evaluation) on the communication benefits of STG and STS compared to NoOptimization. Figures 6, 7 and 8 concern traffic monitored within 1-hop, 2-hops, and max-hops (entire network) respectively. Notice that the two proposed algorithms perform similarly in the context of the entire network (Figure 8) obtaining a relative benefit of up to 20% compared to the *NoOptimization* algorithm. However, STS outperforms STG if we take into account only the traffic near the gateway (Figures 6 and 7). Notice in Figure 6 how STS saves up to 60% receive messages compared to *NoOptimization* when the number of queries is 1, and the gap between the benefits of STS and the benefits of STG increases as we increase the number of queries. The performance of STG for 10 queries falls considerably whereas STS continues to have a 42% advantage (for receive messages) and a 20% advantage (for send messages) over *NoOptimization* (Figure 6).

Vary query type: The performance benefits of the proposed heuristic algorithms were also measured using a different query type in Figures 9, 10 and 11. The queries are now circular with radius ranging between 30 and 60. After randomly selecting their radius, their center is chosen so that the entire query is within the boundaries of the network. Notice that we observe similar trends as in the case of rectangular queries. The benefits of STG and STS are almost identical in the entire network, but STS clearly outperforms STG in the critical nodes around the gateway. As in the case of rectangular queries, both algorithms have a significant advantage over *NoOptimization* in the hotspot areas around the gateway. Similar trends were observed by trying rectangular queries of different sizes than the default rectangular query workload, and located at various parts of the network. Given that the advantages of STG and STS pertain for different query shapes, the experiments that follow use the default rectangular query workload.

Vary number of nodes: Another experiment was done to measure the effect of the node cardinality in the performance of the proposed heuristic algorithms. Figures 12, 13 and 14 clearly show that as the number of nodes increases, and the network density increases, STG and STS demonstrate greater benefits compared to NoOptimization. Intuitively, when the number of nodes is very small (less than 60) the number of disjoing paths from a node to the gateway becomes small, leaving no flexibility for further reducing the communication cost. As the number of nodes increases, NoOptimization routes data through a large number of disjoint paths, whereas STG and STS manage to aggregate results earlier by selecting suitable common paths.

Vary network area: Besides the default network size of 300m by 300m, the performance of the three algorithms was tested on a 400m by 400m network (Figures 15, 16 and 17). The number of nodes is increased to 180 nodes (from 100 nodes) in order to keep the node density equal in both settings. The dimensions of rectangular queries are kept in the range $[30, 300]$. Figure 17 shows that the benefits of STS and STG in terms of the global communication cost increased slightly. The performance of the two algorithms was not considerably affected near the gateway, with a very small improvement of STG and an almost stable performance of STS.

Vary communication range: The next step is to monitor the role of the radio communication range in the performance of the three algorithms (Figures 18, 19 and 20). The increase

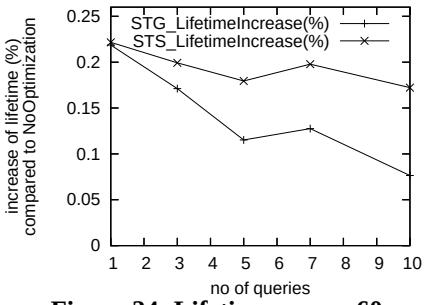

Figure 24. Lifetime range=60

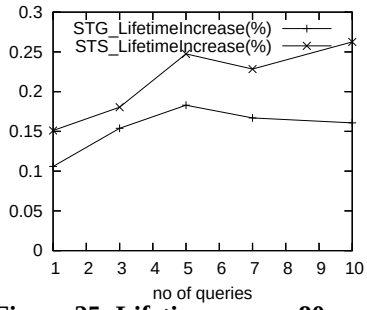

Figure 25. Lifetime range=80

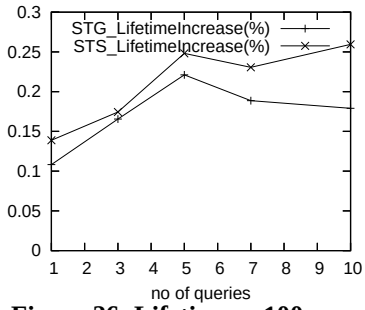

Figure 26. Lifetime r=100

in network connectivity (without increasing the number of nodes) initially increases the benefits of STS and STG compared to NoOptimization. Figure 18 shows that, for a communication range of 100m to 120m, nodes within one hop from the gateway receive up to 80% less messages with STS than with NoOptimization. STG also outperforms NoOptimization, but it remains significantly inferior to STS.

Vary no of network holes: We also measured the ability of STG and STS to cope well with network holes, i.e. areas completely void of sensors. Figures 21, 22 and 23 show that the number of holes (rectangles of dimension in the range $[40, 80]$) have a minor effect in the benefits of STS and STG over the NoOptimization algorithm. In the case of no holes, 48% less messages are received by the immediate 1-hop neighbors of the gateway (from the 2-hop nodes) in STS compared to NoOptimization, and this benefit decreases to 35% for 10 holes. The effect of holes is almost the same as the effect of decrease of nodes from 100 to 80 in Figure 12. Holes do not cause the proposed algorithms performance to deteriorate dramatically in unexpected ways.

Experiment with dynamic failures: It is critical to show that the proposed algorithms also work well in the context of dynamic failures. The previous experiment considered static holes, i.e. unpopulated areas in the network, whereas this experiment studies the effect of dynamic holes as a result of nodes depleting their energy and becoming inactive. All three algorithms rely either on localized route repair or on repetition of the network configuration phase in the occurence of a node or link failure. For ease of implementation, the latter solution was chosen to monitoring the impact of dynamic failures on the result propagation cost. After the initial network configuration phase, the simulator measures the number of epochs until the first node dies (say e_1); the network configuration phase is repeated and the network operation is resumed say for another e_2 epochs until the next node dies. The lifetime of the network is evaluated as the sum of e_i-s until more than 25% of the nodes become disconnected. Figures 24, 25 and 26 show the increase of the network lifetime by using STG or STS instead of NoOptimization. The communication ranges 60, 80 and 100 are tested respectively. As shown in the previous experiments, STS and STG are more effective than NoOptimization in the critical hotspot area around the gateway, which results in immediate lifetime increase for all communication ranges considered. Depending on the communication range used, the lifetime benefits are affected differently by the number of queries. The benefits of STS and STG increase with the number of queries given high connectivity networks, whereas a slight decrease is observed in networks with low node connectivity. In the future it would be interesting to study the effect of dynamic failures on the network lifetime by applying local repair mechanisms.

6.2 Experiments on a real sensor network

We implemented all three algorithms (NoOptimization, STG and STS) in NesC and run experiments in a network of Tmote Sky nodes. Tmote Sky is a wireless sensor platform featuring a low power 4MHz microcontroller, a 2.4GHz IEEE 802.15.4-compliant radio, and a suite of sensors[2]. The communication range indoors is up to 50m, but for the purpose of deploying a sensor network in our lab, we reduced the transmission power of the nodes, in order to set the communication range to a maximum of 2m. The MAC layer used throughout the tests is a light version of B-MAC. Among the main features of the original B-MAC [17], clear channel assessment (CCA) and low power listening (LPL), only CCA is used in our implementation. This has no effect on our experimental results since packet count is used as a way of measuring power consumption. Our network consists of 25 sensor nodes deployed in a 5m x 5m area according to Fig. 27. The gateway node is chosen to be at the bottom left-hand corner of the area.

In order to obtain accurate results over an unreliable medium, the tree construction algorithm was designed carefully to only use high quality links. In an initial phase, all nodes exchange control messages. A node selects as its neighbours only nodes with which it can exchange successfully a high percentage of its control messages, that is, with which it can establish high-quality symmetric links. A shortest path tree is subsequently built. Such a tree is shown on Fig. 27. In the experiments that follow, nodes are up to three reliable symmetric hops away from the gateway.

We used randomly generated query workloads for our experiments. Each query covers a rectangular area with one dimension ranging from 1 to 2m, and the other ranging from 1 to 5m. We varied the size of the query workload from 1 to 10 queries. The same set of query workloads has been reused for all three algorithms. The result propagation phase is set to 30 epochs. We use two performance metrics: (i) accuracy

[2] `http://www.moteiv.com/`

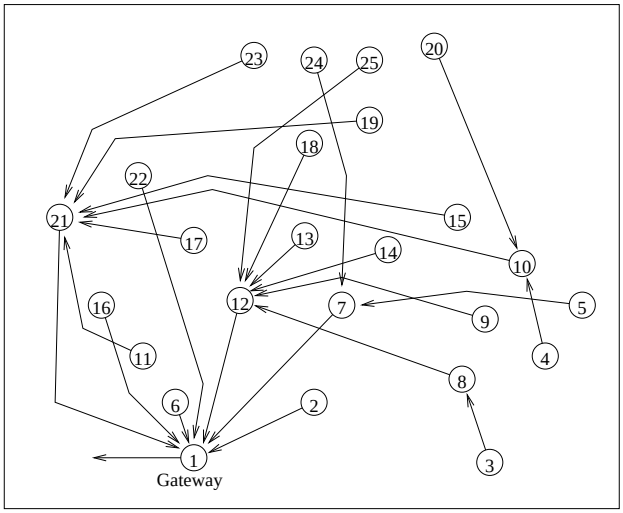

Figure 27. Our 5m x 5m deployment. Nodes 3**,** 4 **and** 20 **are three hops away from the gateway.**

of query results and (ii) communication cost of result propagation. In order to capture congestion, the communication cost is measured in the critical area around the gateway (as well as for the entire network).

Accuracy of query results: Result accuracy is computed as follows: Let e be the expected result for a query and o be the result obtained during the result propagation phase, averaged over 30 epochs. The accuracy for the specific query would then be equal to

$$100(1-\frac{|e-o|}{e}).$$

The overall accuracy for a multi-query experiment is computed as the average of all individual query accuracies. Throughout our experimental results, we observed a very good accuracy, averaging over 93% (see Fig. 28). This can be largely attributed to the tree construction algorithm, since a node can only choose to propagate results over reliable links. No significant difference can be observed between the three algorithms; STG and STS perform similarly to NoOptimization.

Communication cost of result propagation: In order to illustrate the benefits of using STG or STS over NoOptimization, we used our testbed to compare the number of packets sent and received by the three algorithms in the entire network (Fig. 29) and in the critical area around the gateway where congestion occurs (Fig. 30). Network-wide, the STG algorithm generates approximately 10% less packets than NoOptimization, while STS reduces traffic by an extra 5%.

The benefits of STG and STS are amplified for nodes one hop away from the gateway. Both algorithms reduce the number of packets received by 30% compared to NoOptimization. The STG algorithm sends up to 25% fewer packets than NoOptimization, whereas STS yields further savings of up to 35%.

In conclusion:

- our approach yields significant energy savings, without impacting the overall accuracy,
- STG and STS further reduce the energy spent by nodes, especially in the critical area around the gateway,
- using STG and STS eases the congestion around the gateway.

7 Related work

Routing: Several routing protocols for ad-hoc networks have been proposed in the literature [1]. There has also been a plethora of work on energy-aware routing [2, 23, 22, 26] but without considering the interplay of routing and query processing. Pearlman et al. [16] propose an energy-dependent scheme, where a node periodically re-evaluates its participation in the network based on the residual energy in its battery. The integration of their approach with our algorithms will be studied in future work.

Query Processing: The TinyDB Project investigates tree-based routing and scheduling techniques for processing aggregate queries in sensor networks [9, 12]. Tiny aggregation trees are built by considering jointly the routing and the processing aspects of processing a single query. Madden et al. consider the problem of managing multiple queries in [13], but without focusing on the routing aspect; they propose query plan data structures (Fjords) that handle both push-based and pull-based extraction of sensor data. An energy-efficient aggregation tree using data-centric reinforcement strategies is proposed in [10]. Sharaf et al. propose a query-aware *tree* selection scheme to process a single GROUP-BY query [18]. Existing multi-query optimization techniques for sensor networks assume a fixed tree routing structure [20, 21].

Approximate queries and answers: An approximation algorithm for finding an aggregation tree that simultaneously applies to a large class of aggregation functions is proposed in [6]. Duplicate insensitive sketchches for approximate aggregate queries are discussed in [4, 14]. A compressing technique that exploits correlations and redundancies in historical data is described in [5]. Greenwald et al. propose techniques for power-conserving computation of approximate order statistics [8]. Time-decaying aggregates of stream data for scenarios where the significance of data decreases over time are in [3].

Distributed query processing: Although there has been a lot of work on query processing and optimization in distributed database systems [11, 15, 25], there are major differences between sensor networks and traditional distributed database systems. Most related is work on distributed aggregation, but existing approaches do not consider the physical limitations of sensor networks [19]. Aggregate operators are classified by their properties in [7]; an extended classification for sensor networks is proposed in [12].

8 Conclusions and Future Work

This paper shows the interplay of routing and processing in evaluating aggregate queries in sensor networks, and proposes two novel algorithms that significantly outperform the existing approach. STG exploits the new concept of segment-based aggregation, and offers up to 60% energy savings compared to NoOptimization. STS, which avoids

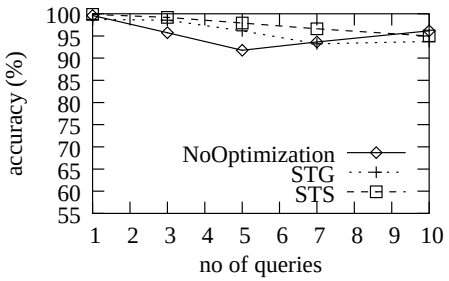

Figure 28. Query accuracy.

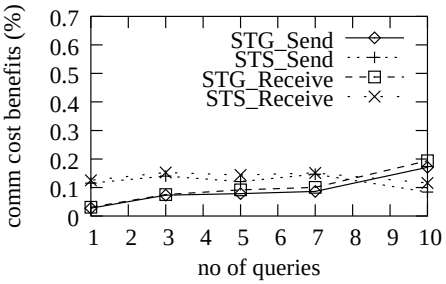

Figure 29. Vary rect. queries

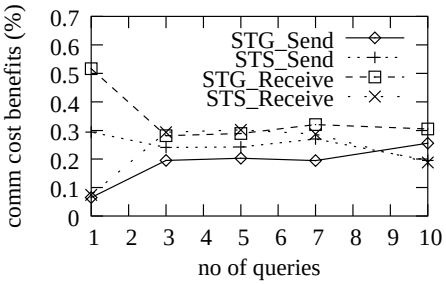

Figure 30. Vary rect. queries 1h

sending query sub-aggregates through disjoint paths, offers even higher savings (up to 80%). It consistently behaves better than STG, especially in the presence of many queries. The greatest savings of STG and STS are observed in the critical area around the gateway, which means that these savings directly reflect an increase in the network lifetime. In the future, we plan to extend STS to become energy-aware, i.e. to optimize aggregate query evaluation taking into account the nodes' residual energy. The effect of local route repairs on the cost of STS is also worth exploring. An exciting direction for future work is also to investigate multi-query optimization techniques for non-summary aggregates (e.g. MEDIAN), other query types that are not geographically local (e.g. GROUP-BY queries), and approximate aggregate queries.

A Processing and Routing: Complexity

A.1 Processing aspect

Consider a fixed tree that connects all sensor nodes and is rooted at the gateway node. Consider a set of long-running spatial range queries (SRQ) that are sent together to the network for evaluation. For simplicity, assume that all queries require results with the same frequency (e.g. every 2 minutes) and for a large number of rounds (e.g. for 60 rounds, i.e. for the next 2 hours). Each node can only forward result values to its parent in the fixed tree. The goal is to minimize the number of result values forwarded up the tree edges, such that all queries can be evaluated at the gateway node. The following results summarize a complexity analysis drawn from [21] for the centralized version of the problem.

THEOREM 2. *Given a fixed tree, the problem of finding the execution plan that minimizes the number of result values forwarded up the tree is NP-hard for min and max queries. The proof includes a reduction from the Set Basis problem and can be found in [21].*

THEOREM 3. *Given a fixed tree, the problem of finding the execution plan that minimizes the number of result values sent up the tree can be solved in linear time for sum, count and avg queries (moments and linear combinations).*

The optimal solution for sum (count and avg) queries is obtained by means of linear reduction. The underlying idea is that if a query value can be derived as a linear combination of other query values, it is redundant for a node to forward this value to its parent. A node should forward up only the values of a maximal subset of (linearly) independent queries. Linear dependency among queries is considered in the context of the node's subtree. For instance, all queries that access a leaf node are identical when projected to the (singleton) subtree rooted at the leaf node, and therefore the leaf node forwards to its parent only one value.

A.2 Routing aspect

The next theorem states the hardness of the routing aspect. The proof relates the routing problem to that of finding an optimal multicast tree (which is NP-hard) using a simple example in which the processing aspect presents no optimization challenges.

THEOREM 4. *The problem of finding the routes of the execution plan that minimizes the number of propagated values is NP-hard.*

Proof. Finding a multicast tree that connects n nodes with a minimum number of edges can be modeled as the NP-complete Steiner problem in networks. Note that the Steiner problem is NP-complete even for grid topologies. Let N be a set of nodes organized in a grid, such that each node has four edges connecting it with its neighbors on the north, east, south and west. Each edge has weight 1. The goal is to reduce the problem of finding an optimal multicast tree for a subset M of these nodes to an instance of our problem, namely to the problem of determining an optimal result propagation plan for a query workload instance. Set the query workload to consist of singleton queries for all nodes in $N - M$. Add to the workload a query defined as the sum of all nodes in N. The gateway is selected to be one of the nodes in M. The optimal solution for the result propagation problem requires all results of the singleton queries to follow the shortest paths to the gateway. The optimal solution always determines an optimal multicast tree that connects all nodes in M. This implies that the problem of finding an execution plan that minimizes the result propagation cost is NP-hard.

B References

[1] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobicom*, pages 85–97, 1998.

[2] J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Infocom*, volume 1, pages 22–31, 2000.

[3] E. Cohen and M. Straus. Maintaining time-decaying stream aggregates. In *PODS*, pages 223–233, 2003.

[4] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, pages 449–460, 2004.

[5] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *Sigmod*, pages 527–538, 2004.

[6] A. Goel and D. Estrin. Simultaneous optimization for concave costs: single sink aggregation or single source buy-at-bulk. In *Soda*, pages 499–505, 2003.

[7] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and M. Pirahesh. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[8] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, pages 275–285, 2004.

[9] J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: towards sophisticated sensing with queries. In *IPSN*, volume 1, pages 63–79, 2003.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobicom*, pages 56–67, 2000.

[11] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[12] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI*, volume 1, pages 131–146, 2002.

[13] S. Madden and M. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. In *ICDE*, pages 555–566, 2002.

[14] S. Nath, P. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Sensys*, pages 250–262, 2004.

[15] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*. Prentice Hall, Englewood Cliffs, 1991.

[16] M. Pearlman, J. Deng, B. Liang, and Z. Haas. Elective participation in ad hoc networks based on energy consumption. In *Globecom*, volume 1, pages 26–31, 2002.

[17] J. Polastre, J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Sensys*, pages 95–107, 2004.

[18] 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):384–403, 2004.

[19] A. Shatdal and J. F. Naughton. Adaptive parallel aggregation algorithms. In *Sigmod*, pages 104–114, 1995.

[20] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Hybrid push-pull query propagation for sensor networks. In *GI Jahrestagung*, volume 2, pages 370–374, 2004.

[21] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *DCOSS*, pages 307–321, 2005.

[22] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Infocom*, volume 2, pages 585–594, 2000.

[23] M. Woo, S. Singh, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobicom*, pages 181–190, 1998.

[24] Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, pages 233–244, 2003.

[25] C. T. Yu and W. Meng. *Principles of database query processing for advanced applications*. Morgan Kaufmann, San Francisco, 1998.

[26] C. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: a recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, University of Southern California, 2001.