Interplay of processing and routing in aggregate query optimization for sensor networks

Niki Trigoni   Alexandre Guitton   Antonios Skordylis

Computing Laboratory, University of Oxford, Oxford OX1 3QD, UK

Abstract. This paper presents a novel approach to processing continuous aggregate queries in sensor networks, which lifts the assumption of tree-based routing. 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 for a variety of query workloads and network topologies.

1 Introduction

A typical way of extracting information from a sensor network is to disseminate declarative aggregate queries from a gateway node to sensor nodes, asking them to periodically monitor the environment, and return aggregate results 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 long-running queries, since 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 [6,8]. Tree construction is performed using simple flooding algorithms [8], data-centric reinforcement strategies [6] or energy-aware route selection schemes [13,16]. After a tree is constructed, sensor nodes forward

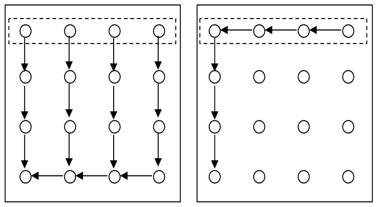
Fig. 1. Example with one query.

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 research has focused on *optimizing multiple aggregate queries given a routing tree* [12]. Query commonalities are taken into account to reduce the communication cost of result propagation, but without making any attempt to select suitable tree routes [12].

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. 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 [6,8,12].
– Two novel heuristic algorithms, SegmentToGateway (STG) and SegmentToSegment (STS), for optimizing multiple aggregate queries (Section 4), by carefully interweaving routing and processing decisions at each node.
– Experimental results that compare the performance of the proposed algorithms with the most efficient existing algorithm for multi-query optimization [12] (Section 5).

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.

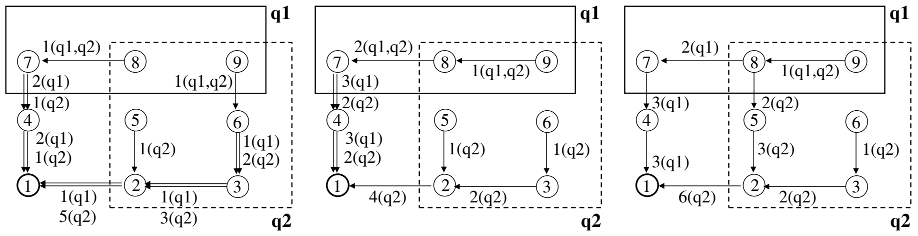

Fig. 2. Example with two queries: (i) the left plan is based on a randomly selected tree, (ii) the middle plan is the output of STG, and (iii) the right plan is the output of STS.

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 an 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).

3 Problem definition

Sensors and queries: Consider a set of sensor nodes $S = \{s_1, \ldots, s_n\}$ with known location coordinates. 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. We consider a commonly used subclass of aggregate queries, which we refer to as spatial range queries (SRQs). SRQs evaluate the aggregate $aggr$ of all sensors in a rectangular area, where $aggr$ is a distributive or algebraic aggregate function (*e.g. sum, count, avg, max, min* but not *median*) [8,4]. A query is denoted by a tuple $(aggr, x_0, y_0, x_{dim}, y_{dim})$, where x_0 and y_0 are bottom left coordinates of the rectangular area and x_{dim} and y_{dim} are the area's x and y dimensions respectively. Let $Q = [q_1, \ldots, q_m]$ be the vector of SRQ queries gathered for execution at the gateway $G \in S$. Queries that evaluate the same aggregate function over different 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 and communication: Nodes receive input values from their neighbors and the local sensors, and generate output values at a negligible cost. One-hop data propagation is represented as a directed edge, labeled with the pair *(value, semantics)*, where the *semantics* denotes how the *value* contributes to each one of the queries. For uniformity, the generation of a reading locally at a node is also represented as a directed edge with a dangling starting point. Such edges are called *initial directed edges*.

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 Figure 3, the initial directed edge 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.

As the initial (value,CV) pairs are pushed towards the gateway, they can be partially processed at intermediate nodes. 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 plan, there should be no loss of information as data is routed through a node, *i.e.* the result of a query when evaluated based on the input edges must be equal to its result based on the output edges. Formally, each node must satisfy the *content preservation property, i.e.* for every query $j = 1, \ldots, m$, $aggr_{i=1}^{k}(v_i * CV_{v_i}[j]) = aggr_{i=1}^{\ell}(v'_i * CV_{v'_i}[j])$. This property is satisfied in Figure 3.
Theorem 1 *If every node in the graph satisfies the content preservation property 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 specifically, 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.*
Optimization goal: *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 property is satisfied at each node.*

4 Algorithms

We now study the existing approach for processing aggregate queries, and propose two novel energy-efficient algorithms to improve its performance. All three algorithms consist of two phases: (i) a network configuration phase and (ii) a result propagation phase. The role of the former phase is to set up routes to prepare the ground for the second phase, i.e. the forwarding of results to the gateway in regular rounds.

4.1 The *NoOptimization* algorithm

The existing state-of-the-art in optimizing multiple aggregate queries is the ECReduced algorithm proposed in [12]. It outperforms Tag [8] and Cougar [15] in the context of multiple queries, since these approaches were originally designed to process a single

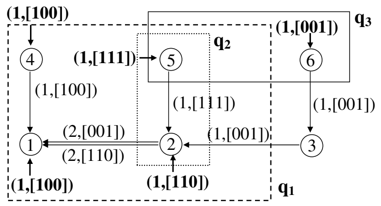

Fig. 3. NoOptimization: Node 2 linearly reduces the three input (value,CV) pairs into two output pairs

query, as shown in [12]. 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.

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 one 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 5, *NoOptimization* is implemented as in [12], *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 [2].

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) 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 (and corresponding values) [12]. 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 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.

4.2 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 *segment* S is a maximal set

```
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 ((SGDistance > hopCount)
    || (SGDistance == hopCount &&
    b → source == parent &&
    b → source! = SGParent &&
    closer(myLoc, leaderLoc))){
       mustRebroadcastBeacon = TRUE;
       SGParent = b → source;
       SGDistance = hopCount;
       distToSGLeader = 0;
       leaderLoc = myLoc;
}}}
```

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

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. For example, the queries in Figure 5 form five segments $\{s_1, s_4\}$, $\{s_2\}$, $\{s_3, s_5, s_6\}$, $\{s_7\}$ and $\{s_8, s_9\}$. A segment S (or a node n in S) is represented by a bit vector that denotes which queries cover the nodes of S (*e.g.*, *SGVector*($\{s_3, s_5, s_6\}$)=[010]). STG performs aggregation of local sensor data by building a tree per segment, instead of building a tree per query, or a tree for all queries. The segment tree is rooted at the *SGLeader*, i.e. the node with the smallest hop count to the gateway. We refer to: (i) the number of hops from the SGLeader to the gateway as the *SGDistance* and (ii) the number of hops from a node to its SGLeader as the *distToSGLeader*. For instance, SGDistance(s_6)=2 and distToSGLeader(s_6)=1.

Network Configuration Phase: This is similar to the corresponding phase of the *NoOptimization* algorithm, except that in this case each node identifies not only a parent neighbor but also a SGParent (*i.e.*, a neighbor on a path to the SGLeader). 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 (Figure 4).

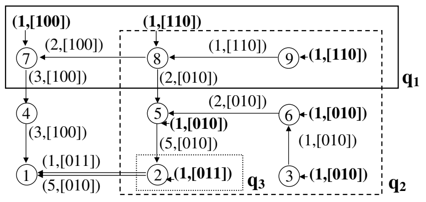

Fig. 5. Value-semantic pairs in the evaluation plan of the sum queries q_1, q_2 and q_3

Result Propagation Phase: 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 gateway's neighbors send all their messages without exception directly to the gateway.
Discussion: 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.

4.3 The *SegmentToSegment (STS)* algorithm

Although STG performs well in terms of merging readings of the same segment, it often fails to merge sub-aggregates of the same query that come from different segments. In the worst case, these sub-aggregates are propagated from the *SGLeader* nodes to the gateway through long disjoint paths. STS addresses the weakness of STG by sending messages towards neighbors that are likely to *reduce* them.
Network Configuration Phase: The configuration phase of *STS* is similar to the corresponding phase of STG except that each node selects as a segment parent a node on the shortest (instead of on any) path to the SGLeader (Figure 4).
Result Propagation Phase: Initially, each node converts input to output (value,CV) pairs exactly as in NoOptimization and STG. It then interleaves two novel steps: 1) *neighbor-message matching*, which selects a suitable neighbor to forward each output pair, and 2) *message splitting*, which often splits the output pair before forwarding it.
Step 1: Neighbor-message matching. The idea behind the first feature is to forward output (value,CV) pairs towards nodes that are most likely to reduce them by merging them with their local or route-thru data. The first (value,CV) pair considered for matching is the one that contributes to most queries (with the greatest number of 1-bits in the CV). The process of matching it with the best neighbor node is detailed below:

Step 1.1: 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) all queries that cover these nodes are also included in the message CV. For instance, s_3 also considers s_6 to forward its initial data $(1, [010])$ to, 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 $[010]$.

Step 1.2: Among neighbors selected in Step 1.1, 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, send the message to its *parent*. Node s_3 has two candidate neighbors, s_2 and s_6, to send (1,[010]) (from Step 1.1). 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 1.3: Among neighbors selected in Step 1.2, 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]) to. 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.

Step 2: Message splitting. The rationale behind this step is that it is often beneficial to divide data into its components in order to give it greater potential for later merging. Let p be the pair considered for neighbor-message matching in the previous step. The pair p is split 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 $p_2 = (2, [010])$. Pair p_1 is sent to the selected neighbor and p_2 is re-inserted into the list of output pairs. During insertion, pairs with equal CVs are merged. If the list of (value,CV) pairs is not empty, steps 1 and 2 are repeated.

Discussion: 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.

5 Experimental evaluation

A thorough experimental evaluation was performed to compare the proposed heuristic algorithms with the existing *NoOptimization* approach using a home-grown simulator. The experimental results below show the performance of the three algorithms varying: (i) the number of queries, (ii) the number of nodes, (iii) the radio communication range, and (iv) the number of holes (unpopulated areas in the network). The graphs below illustrate the communication benefits of STG and STS compared to NoOptimization. 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 re-

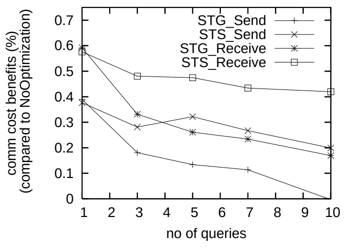

Fig. 6. Vary rect. quer. 1h

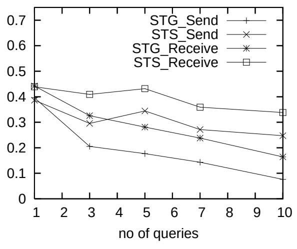

Fig. 7. Vary rect. quer. 2h

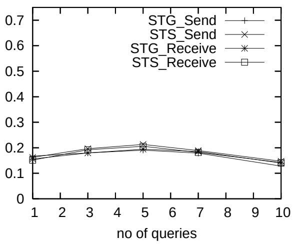

Fig. 8. Vary rect. queries

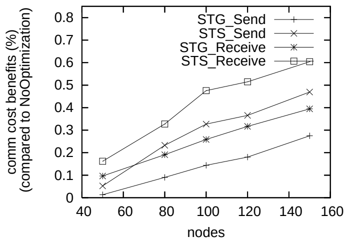

Fig. 9. Vary nodes 1h

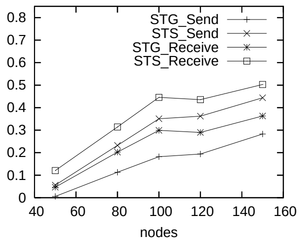

Fig. 10. Vary nodes 2h

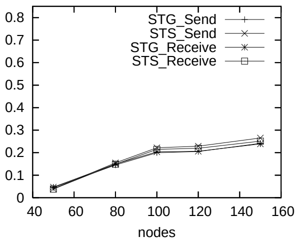

Fig. 11. Vary nodes

sulting 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 nodes at most one hop away from the gateway (left), (ii) by nodes at most two hops away from the gateway (middle) and (iii) by all nodes in the network (right). 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.

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 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

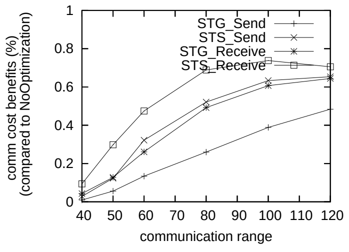

Fig. 12. Vary range 1h

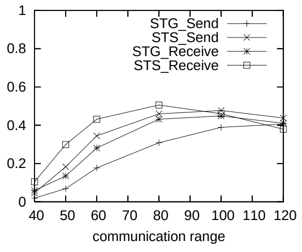

Fig. 13. Vary range 2h

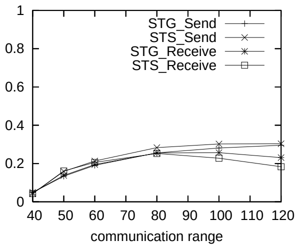

Fig. 14. Vary comm.range

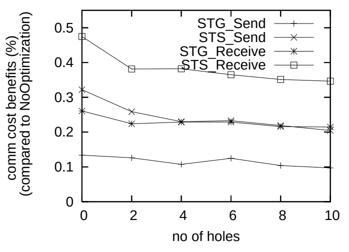

Fig. 15. Vary holes 1h

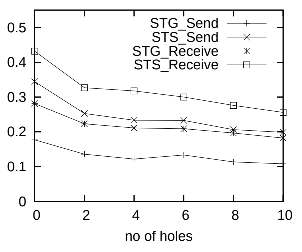

Fig. 16. Vary holes 2h

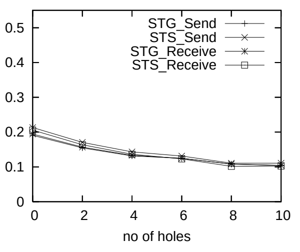

Fig. 17. Vary holes

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 number of nodes: Another experiment was done to measure the effect of the node cardinality in the performance of the proposed heuristic algorithms. Figures 9, 10 and 11 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 communication range: The next step is to monitor the role of the radio communication range in the performance of the three algorithms (Figures 12, 13 and 14). The increase in network connectivity (without increasing the number of nodes) initially increases the benefits of STS and STG compared to NoOptimization. Figure 12 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 outperforms NoOptimization, but it is inferior to STS.

Vary number of network holes: We also measured the ability of STG and STS to cope with network holes, *i.e.* areas completely void of sensors. Figures 15, 16 and 17 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 9. Holes do not cause the proposed algorithms performance to deteriorate dramatically in unexpected ways.

6 Related work

There has also been a plethora of work on energy-aware routing [3,13,16] but without considering the interplay of routing and query processing. The TinyDB [8,9] and Cougar [14,15] projects investigate tree-based routing and scheduling techniques for processing aggregate queries like *avg*, *count*, *sum*, *min* and *max* in sensor networks. The concept of semantic routing trees (SRTs) [9] is used to forward queries only to children that satisfy the query predicate. Zhao et al. [17] compute aggregate summaries over a reliable tree, utilizing a tree construction scheme based on high-quality links, similar to the one used in this work. More sophisticated aggregates are supported in [5], and the benefits of in-network aggregation are discussed in [1]. Directed diffusion [6] is a data-centric protocol that deals with continuous aggregate queries; the network is flooded with an interest for named data and the sources that contain the relevant data respond with the appropriate stream. Madden et al. consider the problem of managing multiple queries in [7], 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. Trigoni et al. [11,12] propose energy-efficient plans for optimizing multiple algebraic aggregate queries in a sensor network. The aforementioned efforts rely on tree-based aggregation and do not exploit the knowledge of the query workload to set up efficient routes for result propagation. The study of decentralized operator placement by Bonfils et al. is closer to our work since the idea is to place operators carefully in the network to minimize the communication cost. Their work considers optimizing a single query, and is more relevant to holistic aggregates, such as correlation or median, and materialized aggregates, such as storage points. Sharaf et al. propose a query-aware *tree* selection scheme, but for processing a different class of (GROUP-BY) queries [10]. We extend previous work on data aggregation in that we depart from the model of tree-based routing, and consider the interaction of processing and routing in reducing the volume of propagated data.

7 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 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 study the effect of local route repairs on the cost of STS, as well as extensions of the algorithm to handle approximate aggregates. Another exciting direction is to explore multi-query optimization techniques for non-summary aggregates (*e.g. median*) and for queries without well-defined spatial coverage.

References

1. B. B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in wireless sensor networks. In *Int. Conf. on Distributed Computing Systems*, pages 575–578, 2002.
2. 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.
3. J.-H. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *Infocom*, volume 1, pages 22–31, 2000.
4. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.
5. J. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: towards sophisticated sensing with queries. In *IPSN*, volume 1, pages 63–79, 2003.
6. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobicom*, pages 56–67, 2000.
7. S. Madden and M. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. In *ICDE*, pages 555–566, 2002.
8. 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.
9. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, 2003.
10. 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.
11. 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.
12. N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *DCOSS*, pages 307–321, 2005.
13. M. Woo, S. Singh, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Mobicom*, pages 181–190, 1998.
14. Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.
15. Y. Yao and J. Gehrke. Query processing in sensor networks. In *CIDR*, pages 233–244, 2003.
16. 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.
17. J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In *Int. Workshop on Sensor Network Protocols and Appl.*, 2003.