Open access • Proceedings Article • DOI:10.1109/ICDE.2005.186

An Analysis of Spatio-Temporal Query Processing in Sensor Networks

Alexandru Coman, Jörg Sander, Mario A. Nascimento

Institutions: University of Alberta

Published on: 05 Apr 2005 - International Conference on Data Engineering

Topics: Visual sensor network, Key distribution in wireless sensor networks, Mobile wireless sensor network and Wireless sensor network

An Analysis of Spatio-Temporal Query Processing in Sensor Networks

Alexandru Coman Jörg Sander Mario A. Nascimento
Department of Computing Science
University of Alberta, Edmonton, Canada
{acoman,joerg,mn}@cs.ualberta.ca

Abstract

Sensor networks are an emerging technology that provide new means to monitor and understand various phenomena. Nodes in a sensor network typically have a limited power supply, thus energy-efficient processing of the queries over the network is a critical issue. We propose analytical models to evaluate the performance of three methods for processing historical spatio-temporal queries in sensor networks. The models allow us to better understand the trade-offs of the investigated method, as well as to recommend the most energy efficient one at query time.

1. Introduction

A sensor network is comprised of a set of sensor nodes which can measure, store and process data and are able to communicate wirelessly. Sensor networks are suitable for many applications, including environmental monitoring, traffic organization, warehouse management, and battlefield surveillance. Sensor nodes are typically battery operated, which highly constrains their life-span. Hence, energy efficient data processing and networking protocols are required for the long-term use of such devices. While the network research community has studied energy efficient protocols in the context of ad-hoc networks, the database community has been confronted mostly with time and size constraints, but rarely with energy limitations. Therefore, the ability to apply traditional data processing techniques in sensor networks is limited, and different solutions must be found.

In [3], we presented techniques for energy-efficient processing of historical spatio-temporal queries, *HST(sw,tw)*. The answer to a *HST(sw,tw)* query is formed by the measurements of all sensors located in the area *sw* taken during the time range *tw*. We studied this problem in a peer-to-peer sensor network environment where each sensor is only aware of the existence of the other sensors located within its communication range, and the query can be initiated at any sensor. We introduced a basic query processing algorithm based on network flooding, and proposed two solutions that minimize the number of nodes that must be contacted during query processing.

An application where such a sensor network environment can be used is micro-climate monitoring in national parks. The sensor nodes could be deployed from a plane over a forest area. Upon activation, each node would start observing periodically various physical phenomena, e.g., temperature and humidity. Park rangers patrolling through the forest can access the network through any node in their proximity using a laptop or PDA. For instance, when certain events such as small fires are observed, park rangers could query the network, from about anywhere, for historical observations, which may help understanding what have caused such events or learn about other areas that are threatened by similar events.

In this paper we develop analytical models to measure the query processing costs for the methods investigated only via simulations in [3]. As sensor nodes spend most of their energy for communication [1], we aim at modelling the amount of energy consumed by nodes for communication during query processing. The models allow us to better understand the trade-offs of the investigated methods, as well as design better query processing solutions for various sensor environments when the query window covers only a subset of all sensor nodes. In addition, the models can be used for recommending at query time the most energy efficient query processing method.

The remainder of this paper is organized as follows. Section 2 summarizes the methods proposed in [3]. The analytical models for these methods are developed in Section 3. Section 4 presents the evaluation of the investigated methods based on the analytical model and discusses the the trade-offs between them. Section 5 describes some of the research related to ours and Section 6 concludes the paper.

2. HST query processing

We assume a network with fixed sensor nodes that have equal roles in the functionality of the network. Due to the

wireless network characteristics, a sensor node can communicate directly only with the nodes located within its wireless range, which form its neighborhood. A node can send a message individually (unicast) to one of its neighbors, or it can send it simultaneously to all of its neighbors (broadcast), and it can communicate with nodes other than its neighbors using a multi-hop routing protocol. Sensors take measurements periodically, and the collected values are stored locally for future querying. Each measurement has attached a timestamp corresponding to the time of measurement and a sensor location, which gives spatio-temporal properties to data.

A straightforward way to answer a *HST(sw,tw)* query, called *FullFlood*, is contacting every network node. The query originator node, which can be any node in the network, broadcasts the query to its neighbors, which in turn broadcast the query to their neighbors, and so on, until all nodes have received the query. A consequence of broadcasting is that each sensor node may receive the same query several times. For each query, a node processes only the first query message received, discarding subsequent messages. The query answers are returned only to the neighbor the query was first received from. To all the other neighbors, an empty answer is returned. When a query is received, the node broadcasts the query, selects the locally stored data relevant to the query (if any), waits for its neighbors' answers and merges them with its own, and finally it returns the answer to the neighbor that it received the query from. Once the query originator node has received answers from all of its neighbors, it can answer the query to the user.

When the spatial area sw of the query covers less than the whole monitored area, i.e., only a subset of nodes, contacting all sensor nodes as in *FullFlood* may not be the most energy efficient approach. Since it is not feasible[1]to compute an optimal solution for query processing at the sensor nodes, we proposed two heuristic methods in [3], both based on the STWIN query processing framework. In the STWIN framework, the query processing is divided into two phases. In the first phase, a routing path must be discovered from the query originator node to a sensor node located inside the query's spatial window, called query coordinator. In the second phase, the query coordinator node must disseminate the query to all nodes located in the query's spatial window, called query relevant nodes. Then, the query answers are gathered from the relevant nodes to the coordinator and are returned to the query originator over the path discovered in the first phase.

For the first phase of STWIN, a greedy approach (called *GreedyDF*) is used to discover a routing path from the query originator node to the coordinator node located near the center of the query's spatial window. At each step, the query is forwarded to the neighbor located closest to the center of the query window. Greedy-based routing methods for position based routing in ad-hoc networks have been shown to nearly guarantee delivery for dense network graphs [14], as it is the case for sensor networks [13]. If the sensor network is not dense, more advanced geographic routing techniques such as [7] (slightly modified to accommodate the lack of a node at the destination location) could be used to improve the reliability of message routing.

For the second phase, two different approaches were studied in [3]. The first heuristic, called *WinFlood*, consists of a constrained parallel flooding initiated by the query coordinator, where a node broadcasts the query to its neighbors only if its own location is inside the query's spatial window. Similar to *FullFlood*, nodes wait to receive the neighbors' answers (including empty answers) before returning the merged query answers to the neighbor that the query was first received from. In the second heuristic, called *WinDepth*, each node may forward the query only to those neighbors located within the query's spatial window. When a node receives a query, it adds its node identifier in the query header so that the query path is remembered. Then it selects a neighbor located within the spatial window that has not received the query yet (determined based on the query header), and forwards the query to this neighbor. When the neighbor returns the query and the query answers, the node checks for any other neighbors that are relevant to the query and have not received it yet. If there is one, it forwards the query to that node and waits for the neighbor's answer. This process is repeated until all of a node's neighbors located within the query's spatial window have answered the query, at which point all the answers received are merged with the locally stored answers and are returned to the neighbor that the node received the query from.

3. A cost model for HST query processing

We start by defining a few notations and estimating some of the basic values used in the models. The area covered by the wireless range[2] W of a node is $A_w = \pi W^2$. Assuming a sensor network with N nodes uniformly deployed over a monitored region of width X and height Y, each point in the region is covered in average by the wireless ranges of $\frac{NA_w}{XY}$ nodes. Each sensor node is covered by the wireless ranges of its neighbors, therefore the average number of neighbors for a node is $N_n = \frac{NA_w}{XY} - 1$. The number of nodes relevant to a query is proportional to the area covered by the query's spatial window, and it can be expressed

1 It would require that each sensor node obtains and stores information about the whole network. Also, the query originator would have to do expensive computation to fi nd the optimal route for each particular query.

2 As typical sensors do not have sophisticated communication electronics capable of adapting the transmission range [4], we assume all messages are transmitted as far as the wireless transmission range W.

as: $N_r = N\frac{Q_x Q_y}{XY}$, where Q_x and Q_y denote the width and height of the query window. Table 1 summarizes the notations used in the models. For the *WinFlood* and *WinDepth* algorithms, we consider a coordinator node C located at the center of the query window. We are interested in the behavior of the algorithms for variations in the number of sensors N, the size of the query's spatial window $Q_x Q_y$ and the temporal range Q_t of the query.

Description	Notation	Default value
Size of monitored region on X axis	X	1000 meters
Size of monitored region on Y axis	Y	1000 meters
Number of nodes	N	2000
Wireless range	W	50 meters
Size of query's window on X axis	Q_x	100 meters
Size of query's window on Y axis	Q_y	100 meters
Size of query's temporal range	Q_t	60 measurements
Size of query message	Q_s	192 bits
Size of a measurement tuple	T_s	64 bits
Size of the empty query answer	$\emptyset_s$	8 bits
Energy used to unicast (broadcast) a bit	E_u (E_b)	$\alpha + \gamma d^n$ nJ/bit [10]
Energy used to receive a bit	E_r	β nJ/bit [10]
Other notations		
Query originator node	O	-
Center of the query's area (coordinator)	C	-
Average number of relevant nodes	N_r	-
Average number of neighbors per node	N_n	-
Number of hops between O and C	$\overline{hOC}$	-
Number of hops from a relevant node to C	$\overline{h2C}$	-

Table 1. Notations and default values

3.1. Estimating the cost of GreedyDF

The energy consumed by *GreedyDF* for discovering a path between the query originator node O and coordinator C and sending the query over this path is: $E_{GDF} = (E_u + E_r)Q_s\overline{hOC}$, where $\overline{hOC}$ is the number of hops between O and C. We assume a dense sensor network [13], which allows us to approximate $\overline{hOC}$ as the distance $\overline{dOC}$ between the originator and coordinator nodes divided by the average advance $\overline{a2C}$ from a node towards C: $\overline{hOC} = \overline{dOC}/\overline{a2C}$. We assume that both the locations of the query originators and the query areas are uniformly distributed over the monitored region. Since the query area falls inside the monitored region, the centers of the query areas are uniformly distributed in a window of size $(X - Q_x)(Y - Q_y)$. We approximate (without boundary effects) the average distance between O and C as half of the maximum possible distance between two such points: $\overline{dOC} = \frac{1}{2}\sqrt{(X - \frac{Q_x}{2})^2 + (Y - \frac{Q_y}{2})^2}$. The average distance $\overline{dN}$ between a node N_i and its neighbors is equal to the sum of distances d from N_i to all possible neighbor locations divided by the number of these locations: $\overline{dN} = \frac{\int\int_{A_w} d(N_i, N_j(x,y))dxdy}{A_w} = \frac{2W}{3}$. The probability that a neighbor is located on the direction of C is low, and therefore the query will be forwarded to a neighbor located at an angle from this direction. The average angle between two successive (angle-wise) neighbors N_j and N_k of a node N_i is $\angle N_j N_i N_k = \frac{2\pi}{N_n}$. The direction from N_i to the selected neighbor will make in average an angle of $\frac{\angle N_j N_i N_k}{4}$ radians from the direction towards C. It follows that the average advance from a node towards C is $\overline{a2C} = \overline{dN}\cos\frac{\angle N_j N_i N_k}{4} = \frac{2W}{3}\cos\frac{\pi}{2N_n}$. An increase in the number of sensor nodes allows a better neighbor selection, helping *GreedyDF* decrease its costs. When the size of the query area is increased, the distance $\overline{dOC}$ decreases, reducing the cost of the algorithm. Variations in the query's temporal range do not affect the cost of *GreedyDF*.

3.2. Estimating the cost of WinFlood

For estimating the energy cost E_{WF} of the *WinFlood* algorithm, we divide the cost into three components: the cost to forward the query to the relevant nodes, the cost to return their answers to the coordinator C, and, finally, the cost to send the answers from C to the query originator O: $E_{WF} = E^q_{WF} + E^{a2C}_{WF} + E^{aC2O}_{WF}$.

During query forwarding, each relevant node will broadcast the query once, and receive the query from all its neighbors (we do not consider the boundary effects): $E^q_{WF} = E_b Q_s N_r + E_r Q_s N_r N_n$. Even though E^q_{WF} grows quadratically in N, for small query windows the slope of the increase is small, since the fractions in N_r and N_n are small. However, for large query windows, this cost will increase substantially.

The query answers from the relevant nodes are returned over the shortest path (in number of hops) to the coordinator (due to flooding, nodes are first contacted over the shortest path). We estimate the average distance between C and a relevant node as half of the maximum distance between any two such points: $\overline{dC} = \frac{1}{2}\sqrt{(\frac{Q_x}{2})^2 + (\frac{Q_y}{2})^2}$. The average advance $\overline{a2C}$ from a relevant node to C is calculated in the same way as for the *GreedyDF* algorithm. Therefore, the average number of hops between C and a relevant node is $\overline{h2C} = \frac{\overline{dC}}{\overline{a2C}} = \frac{\sqrt[3]{Q_x^2 + Q_y^2}}{8W\cos\frac{\pi}{2N_n}}$. The energy used for gathering the answers at C is proportional to the size of the query's temporal range Q_t and the size of a measurement tuple T_s: $E^{a2C}_{WF} = (E_u + E_r)T_s Q_t (N_r - 1)\overline{h2C}$. Note that the product $T_s Q_t$ represents the size of the query answer returned by a node, and the product $T_s Q_t (N_r - 1)$ represents the size of all query answers from all relevant nodes except the coordinator node C (since C is one of the relevant nodes).

Finally, the coordinator C sends the query answers collected from all (N_r) relevant nodes to the originator O over the path discovered by *GreedyDF*: $E^{aC2O}_{WF} = (E_u + E_r)T_s Q_t N_r \overline{hOC}$. The costs of returning the answers in-

creases linearly with N, Q_xQ_y and Q_t. As the cost of E^q_{WF} does not depend on Q_t, the size of the temporal range decides which of the three costs has a larger weight in the total cost of *WinFlood*. Even though E^q_{WF} is quadratic in N, when $Q_s \ll Q_t$ the cost of *WinFlood* is determined by the cost to return the query answers.

3.3. Estimating the cost of WinDepth

The performance of the *WinDepth* algorithm is highly dependent on the layout of the network formed by the relevant nodes. To estimate its cost, we assume that the algorithm can route the query and receive the answers in a single path[3] connecting all relevant nodes. Therefore, each relevant node receives and forwards the query twice (once from/to its parent, once to/from its child), as well as participates in the return of the answers for all relevant nodes located farther away from the coordinator on the contacting path. We divide the estimation of the energy cost E_{WD} of *WinDepth* into three components: to forward the query to the relevant nodes, return their answers to C, and send the answer from C to the query originator O: $E_{WD} = E^q_{WD} + E^{a2C}_{WD} + E^{aC2O}_{WD}$.

Since the forwarding path is saved in the query, the query is forwarded in average with $N_r/2$ node id entries (in addition to the query data), while on the return path it is forwarded with N_r node id entries, as all relevant nodes were already contacted. We assume 16 bits are used to store a node id. Therefore, the cost for disseminating the query to the relevant nodes is: $E^q_{WD} = (E_u+E_r)(Q_s+16\frac{N_r}{2})(N_r-1) + (E_u+E_r)(Q_s+16N_r)(N_r-1)$.

When returning the answers to the coordinator node C, the last contacted node will return the answer of one node, the next node will return the answers of two nodes, until reaching C, which receives $N_r - 1$ node answers (since C is also a relevant node): $E^{a2C}_{WD} = (E_u+E_r)T_sQ_t\frac{(N_r-1)(N_r-2)}{2}$. Note that the product T_sQ_t represents the size of the query answer returned by a relevant node.

Finally, the coordinator C sends the query answers to the originator node O over the path discovered by *GreedyDF*: $E^{aC2O}_{WD} = (E_u + E_r)T_sQ_tN_r\overline{hOC}$, where the product $T_sQ_tN_r$ represents the size of all query answers from all relevant nodes. Both E^q_{WD} and E^{a2C}_{WD} costs depend quadratically in N_r, and therefore are strongly affected by variations in N and Q_xQ_y. If the query covers a large fraction from the monitored region, an increase in N leads to a quadratic increase of these costs. Similar to *WinFlood*, *WinDepth* is linear in Q_t, which partly determines the weight of each cost in the total cost of the algorithm.

3 For dense sensor networks, *WinDepth* contacts all relevant nodes in a single path or in a very deep tree with only a few branches [3].

3.4. Estimating the cost of FullFlood

We divide the estimation of the energy cost E_{FF} of *FullFlood* algorithm into three components: to forward the query, to return the empty answer which signals that the query has already been processed, and finally, to send the query answers from the relevant nodes to the originator O: $E_{FF} = E^q_{FF} + E^{ea}_{FF} + E^{a2O}_{FF}$.

For disseminating the query, each node will broadcasts the query once, and receives the query from all its neighbors: $E^q_{FF} = E_bQ_sN + E_rQ_sNN_n$. After receiving the query, all nodes except the relevant nodes will return an empty answer to all their neighbors, while the relevant nodes will return the query answer to one of their neighbors and the empty answer to every other neighbor. Thus, $E^{ea}_{FF} = (E_u + E_r)\emptyset_s(NN_n - N_r)$. The relevant nodes send the answers over the shortest path (in number of hops) to the query originator O (due to flooding, nodes are first contacted over the shortest path). As both the query and the nodes are uniformly distributed in the monitored region, the average number of hops between O and a relevant node can be approximated by the number of hops between O and the center C of the query area (calculated for *GreedyDF*): $E^{a2O}_{FF} = (E_u + E_r)T_sQ_tN_r\overline{hOC}$.

Both E^q_{FF} and E^{ea}_{FF} costs depend quadratically in N, E^{ea}_{FF} being also slightly affected by the size of the query area (and not affected by Q_t). Thus, for denser networks a large increase in these costs is expected. The E^{a2O}_{FF} cost is linear in all three variables. Differently from *WinFlood* and *WinDepth*, both Q_t and Q_xQ_y affect the weights of the three costs of *FullFlood*. For small queries, the cost of *FullFlood* is dominated by the cost of query forwarding, while for large queries the cost of returning the answers prevails.

3.5. Cost Models Summary

To ease the comparison of the overall costs of the presented query processing solutions, we summarize the cost formulas of each algorithm into one. For the typical network flooding, the cost of processing HST queries is:

$$\begin{aligned} E_{FF} &= E_bQ_sN + E_rQ_sNN_n + \\ &\quad (E_u + E_r)\emptyset_s(NN_n - N_r) + \\ &\quad (E_u + E_r)T_sQ_tN_r\overline{hOC}. \end{aligned}$$

For the solutions within the STWIN framework, the total cost is equal to the sum of the cost of *GreedyDF* and the algorithm used for the second phase (i.e., *WinFlood* or *WinDepth*). When *WinFlood* is used, the total cost is:

$$\begin{aligned} E_{GDF+WF} &= (E_u + E_r)Q_s\overline{hOC} + \\ &\quad E_bQ_sN_r + E_rQ_sN_rN_n + \\ &\quad (E_u + E_r)T_sQ_t(N_r - 1)\overline{h2C} + \\ &\quad (E_u + E_r)T_sQ_tN_r\overline{hOC}, \end{aligned}$$

while for *WinDepth* we have:

$$\begin{aligned} E_{GDF+WD} &= (E_u + E_r)Q_s\overline{hOC} + \\ &\quad (E_u + E_r)(Q_s + 24N_r)(N_r - 1) + \\ &\quad (E_u + E_r)T_sQ_t\frac{(N_r - 1)(N_r - 2)}{2} + \\ &\quad (E_u + E_r)T_sQ_tN_r\overline{hOC}. \end{aligned}$$

4. Discussion

As examined in the previous section, the investigated algorithms behave differently for variations in the query size and number of neighbors. An increase in the number of sensors should strongly affect the *FullFlood* algorithm, while *WinFlood* should be only slightly affected. When the query area increases, *WinDepth* should have a quadratic energy increase, while *WinFlood* and *FullFlood* only a linear one. All algorithms (except *GreedyDF* which is used for routing the query to the coordinator) should be affected linearly by variations in the query's temporal range. We compare the costs of the algorithms using both the cost models and simulations (taken from [3]). In the cost models we used the parameter values as listed in Table 1 and the following values needed for E_u, E_b, and E_r [2]: $\alpha = 45$ *nJ/bit*, $\beta = 135$ *nJ/bit*, $n = 2$, and $\gamma = 10$ *pJ/bit/m*^{2}. Both *WinDepth* and *WinFlood* algorithms are combined with *GreedyDF* to form a complete query processing solution.

The increase in the number of sensors N (Figure 1(a)) affects strongly the average energy used by *FullFlood* due to the increased number of query messages each node receives, as well as the increase in the number of empty messages that are exchanged (the E^q_{FF} and E^{ea}_{FF} costs). While having more nodes affects *GreedyDF* only slightly, it affects *WinFlood* and *WinDepth* in different ways. Since the query window is small, the increase in the number of relevant nodes has a minor effect on *WinFlood*, and a stronger effect on *WinDepth* (due to E^q_{WD} and E^{a2C}_{WD} costs). For dense networks, minimizing the number of nodes contacted helps the algorithms within the STWIN framework keep the energy costs low. The experimental results (Figure 1(b)) are qualitatively the same, while quantitatively all the methods show a slightly lower energy usage than obtained with the cost models.

The effects of varying the size of the query area are shown in Figure 1(c). The increase in the size of the query area produces a linear increase in the number of relevant nodes N_r. Due to the E^q_{WD} and E^{a2C}_{WD} costs, the *WinDepth* algorithm shows a quadratic increase in its energy cost. The cost of *GreedyDF* with *WinFlood* increases faster than the cost of *FullFlood*, whose E^q_{FF} cost stays constant. When the query area reaches a certain relative value with respect to the monitored region, the cost of returning the query answers dominates all algorithms, giving an advantage to *FullFlood* which returns the answers over the shortest path

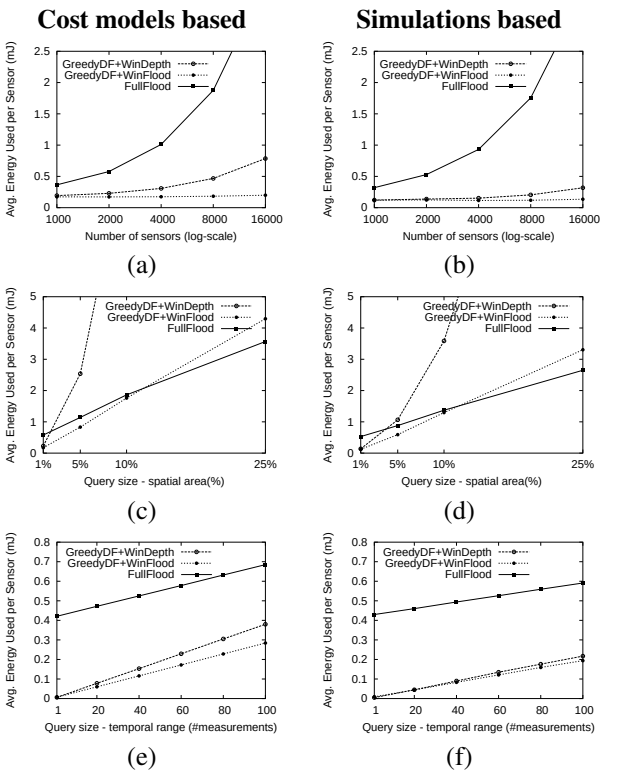

Figure 1. The average energy used per node using the cost models and simulations

to the query originator. In both the cost models and the simulations (Figure 1(c,d)), the *FullFlood* algorithm is the most energy efficient of the three methods after the query area covers around 12% of the monitored region, which also shows the qualitative accuracy of the cost model. For large queries, the overhead for discovering shortest path routes to the relevant nodes pays off through substantial energy savings. Basic processing algorithms such as *FullFlood* can be winners for such queries.

Finally, increasing the query's temporal range Q_t results in a linear increase in the costs of the algorithms (Figure 1(e)). The *WinDepth* algorithm is affected more than *WinFlood* due to the longer path over which a larger answer set must be returned to the coordinator node. As the size of the answer grows, the weights of E^q_{FF} and E^{ea}_{FF} in *FullFlood* decrease relatively, while the saving in energy due to returning the answers over the shortest path grow when compared with the other two methods.

As shown in Figure 1(a-f), the cost models capture well the behavior of the algorithms. Therefore, the models can be used to determine the most energy efficient algorithm given a query and a sensor network. They can be easily implemented in the sensor nodes to help the query originator

determine which is the most energy efficient algorithm for processing a given query.

5. Related work

Directed Diffusion [6] investigates query processing in a sensor network environment similar to ours in the sense that the query can be originated at any node, and nodes are only aware of their neighborhood. Differently from us, nodes do not store historical data and sensing is only performed in response to a query request. A system focusing on query processing over historical data is DIMENSIONS [5]. Their focus is on multi-resolution summarization of data for data mining, where a query can first look at the data at a coarse resolution and then focus on a region of interest at a finer resolution. Several data-dissemination methods are discussed in [11, 12], and the GHT system for data-centric storage is introduced. In [11], the simulation results show that the local storage of measurements performs the best for scenarios like ours where a large number of observations is available with only a small subset of them being retrieved. In [8, 9], Madden et al. focus on query processing in a sensor environment where the information about the existing sensors is available in a catalog. Sensor nodes simply collect and transmit the raw data to the powered sensor proxies that are in charge of further processing and routing the answers to the users. The Cougar project [15, 16] also investigates techniques for query processing over sensor data. However, unlike ours, their research focuses on a sensor network environment where there is a central administration that knows the location of all sensors. A central optimizer has the tasks of building a query plan and disseminating it to the relevant sensor nodes.

6. Conclusions

In this paper we investigated energy efficient query processing in a peer-to-peer sensor network environment. In this scenario we studied three methods, each based on a different processing strategy. We built analytical models to capture the effects of various parameters on the methods, which helped us better understand their behavior. We compared the models with the experimental simulations and we showed that they capture well the behavior of the studied algorithms, which also makes it possible to recommend at query time the most energy efficient method.

Our current investigations looked into processing historical spatio-temporal queries for retrieving the relevant raw data. In the future work, we will study the effect of in-network data aggregation. Early data aggregation at the coordinator node in the STWIN based algorithms would reduce the energy costs, possibly making them more efficient that *FullFlood* for large queries. We will also study coordinator nodes located at other positions than the center of the query area as they may reduce the length of the path over which the query answers are returned, further reducing the energy costs.

Acknowledgments. This work was partially supported by NSERC. We would like to thank the anonymous reviewers for their useful suggestions to improve our work.

References

[1] I. Akyildiz et al. Wireless sensor networks: A survey. *Computer Networks*, 38(4):392–422, 2002.

[2] M. Bhardwaj. Power-aware systems. Master's thesis, MIT, 2001. http://www-mtl.mit.edu/research/icsystems/uamps/pubs/theses/.

[3] A. Coman et al. A framework for spatio-temporal query processing over wireless sensor networks. In *Proc. of DMSN Workshop (with VLDB)*, pages 104–110, 2004.

[4] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. Energy-efficient data management for sensor networks: A work-in-progress report. In *Proc. of Upstate New York Workshop on Sensor Networks*, 2003.

[5] D. Ganesan et al. An evaluation of multi-resolution storage for sensor networks. In *Proc. of ACM SenSys*, 2003.

[6] C. Intanagonwiwat et al. Directed diffusion for wireless sensor networking. *IEEE Trans. on Networking*, 11(1):2–16, 2003.

[7] B. Karp and H. Kung. Greedy perimeter stateless routing for wireless networks. In *Proc. of ACM MobiCom*, pages 243–254, 2000.

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

[9] S. Madden et al. The design of an acquisitional query processor for sensor networks. In *Proc. of ACM SIGMOD*, pages 491–502, 2003.

[10] T. Rappaport. *Wireless Communications: Principles and Practice*. Prentice-Hall Inc., 1996.

[11] S. Ratnasnmy et al. GHT: A geographic hash table for data-centric storage. In *Proc. of WSNA Workshop*, 2002.

[12] S. Shenher et al. Data-centric storage in sensornets. In *Proc. of HotNets Workshop*, 2002.

[13] E. Shih et al. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks. In *Proc. of ACM MobiCom*, pages 272 – 287, 2001.

[14] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communications Magazine*, 40(7):128–134, 2002.

[15] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[16] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. of CIDR*, 2003.