A Framework for Spatio-Temporal Query Processing Over Wireless Sensor Networks

Alexandru Coman   Mario A. Nascimento   Jörg Sander

University of Alberta
Edmonton, Canada
{acoman,mn,joerg}@cs.ualberta.ca

Abstract

Wireless sensor networks consist of nodes with the ability to measure, store, and process data, as well as to communicate wirelessly with nodes located in their wireless range. Users can issue queries over the network, e.g., retrieve information from nodes within a specified region, in applications such as environmental monitoring. Since the sensors have typically only a limited power supply, energy-efficient processing of the queries over the network is an important issue. In this paper, we introduce a general framework for distributed processing of spatio-temporal queries in a sensor network that has two main phases: (1) routing the query to the spatial area specified in the query; (2) collecting and processing the information from the nodes relevant to the query. Within this framework, different algorithms can be designed independently for each of the two phases. We also propose novel algorithms for this framework, one for the first phase and two for the second phase. In an extensive experimental evaluation we study the performance of these algorithms in terms of energy consumption, under varying conditions. The results allow us to recommend the most energy efficient solution, given a network and a spatio-temporal query.

1 Introduction

Recent technological advances, decreasing production costs and increasing capabilities have made sensor networks suitable for many applications, including environmental monitoring, biological contamination detection, warehouse management, traffic organization and battlefield surveillance.

Today's sensors are no longer just simple sensing devices wired to a central monitoring site, but they have computation, storage and wireless communication capabilities. Most of these devices are battery operated, which highly constrains their life-span, and it is often not possible to replace the power source of thousands of sensors. Energy efficient data processing and networking protocols must therefore be developed to make the long-term use of such devices possible. While the network research community has studied energy efficient protocols in the context of ah-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 this paper we focus on energy efficient query processing in sensor networks. In particular, we are interested in answering historical spatio-temporal queries such as "What was the temperature yesterday morning in Edmonton's west end?". We study this problem in a sensor network where each sensor is only aware of the existence of the other sensors located within its communication range, and the query can be initiated locally at any sensor. To the best of our knowledge, query processing in such a sensor network environment has not been investigated before. The advantages of this environment are network robustness, a balanced use of sensors' energy resources and a wide range of application scenarios that can take advantage of the proposed solutions.

We propose the STWin framework for processing historical spatio-temporal queries in sensor networks, i.e., queries that specify the spatial area and temporal range the answers must belong to. As sensor nodes spend most of their energy supply during communication [1], we aim at minimizing the amount of data exchanged among nodes during query processing. Our framework has two phases. First, we search for a path from the query originator node to a sensor located within the query's spatial window. Second, the located sensor assumes query coordinator role, gathers the answers from all query relevant sensors and ships them back to the query originator. We use a greedy routing algorithm in the first phase, while for the second phase we propose two algorithms, one based on parallel flooding, the other using a depth first strategy.

In summary, the contributions of this paper are:

- we study the processing of spatio-temporal queries in a sensor network where each node only knows about the network nodes located within its wireless range;
- we introduce the STWin query processing framework;
- we propose three algorithms to be used within the STWin framework;
- we evaluate experimentally the performance of these algorithms and discuss their benefits and drawbacks.

The remainder of this paper is organized as follows. Section 2 describes some of the research work related to ours.

Section 3 presents the sensor network settings as well as the characteristics of the query and data. Section 4 details our solution for spatio-temporal query processing. Section 5 presents the experimental evaluation of the proposed algorithms, and Section 6 concludes the paper.

2 Related Work

The Cougar project [17] is one of the most related to ours as it also investigates techniques for query processing over sensor data. However, unlike ours, their research focuses on a sensor networks environment where there is a central administration that knows the location of all sensors. In [4] the authors focus on processing long-running queries, modelled as persistent views over the distributed sensor database. As the sensors are connected to a continuous power source, they are not concerned about the energy use. In the same scenario but with emphasis on energy efficient query processing, they extend their work in [18] and analyze a wider range of problems, such as routing and crash recovery, basic query plans and in-network aggregation.

In [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 authors focus on minimizing the used energy by adapting the sensors' sampling and data package transmission rates. They introduce the Fjord architecture for management of multiple queries. Designed for Berkley Mica motes and running on top of TinyOS, TinyDB [11] is a distributed query processor that runs on each of the sensor nodes. The authors focus remains on optimizing data acquisition for long-running queries, no data being stored locally at the nodes. To reduce the energy consumption, they also propose TAG [10], an aggregation service for networks of TinyOS motes.

Beaver et al. [2] propose a solution for in-network aggregation, which exploits the temporal correlation in a sequence of sensor readings to reduces the energy used during query processing. Their solution, called TiNA, also allows the user to specify a temporal coherency tolerance when an approximate answer is sufficient, which lowers the energy costs. Similar to TinyDB and Cougar, TiNA is designed for a sensor environment where sensors simply forward their measurements to answer a long-running query, without storing any historical data.

The authors in [8] focus on sensor data processing, and propose solutions for data stream joins over the sensor data in tracking or monitoring application. The performance of their solution decreases sharply with increasing number of sensors, with more evaluation being required to establish the validity of their methods for large scale sensor deployments. In [6], the authors propose one of the first index structures for sensor networks. The solution is based on the R-tree and it seems to be energy and time efficient, but no evaluation is presented. Xu and Lee [16] propose a window-based query processing technique in a network of moving sensors, where sensors only take measurements and provide data upon users' request. Though interesting, their solution has no experimental evaluation.

In the area of networks research, much work has been done in ad-hoc wireless networks. One of the most relevant issues for efficient query processing in sensor networks is position based routing, that is, network routing when the destination node is known and addressed by means of its location. Stojmenovic [15] offers an excellent survey of techniques for position based routing in ad-hoc networks. In our scenario, a sensor node is only aware of the network nodes located within its wireless communication range, which complicates the routing decisions. In such a case position based routing algorithms with guaranteed delivery cannot be readily re-used.

3 Background and Settings

We assume a sensor network with fixed nodes that have equal roles in the network's functionality. Each node has a CPU, long term storage, its own energy source and it is connected to other members of the network through wireless communication. All sensor's components have limited capabilities and the power source can be depleted quickly if not used efficiently.

Due to the wireless network characteristics, a node can communicate directly only with the sensors located within its wireless range, which form its neighborhood. A node can send a message individually to one of its neighbors, or it can use broadcasting to send the message simultaneously to all of its neighbors. When a message has to be sent to all or most neighbors, it is cheaper to broadcast the message than to send it individually to the neighbors. A sensor communicates with nodes other than its neighbors using a multi-hop routing protocol over the network. There are two main types of messages in query processing: query messages (which transport the query) and answer messages (which transport the query answers).

Each node knows its location (e.g., it may use GPS during node activation), as well as the location of its neighbors (collected during network activation). Sensor nodes may have several sensors, e.g., for temperature, humidity, magnetism, and light. In this paper we consider sensors that observe the state of a measured entity at the sensor location only. This is different from range sensing (e.g., movement sensing used in tracking [7, 14]), which measures the state of an entity not necessarily located at a sensor's position, but in its vicinity. Sensors take measurements periodically, and the collected values are stored locally for future querying. Data collection is performed continually, which can be viewed as an infinite stream. As infinite data cannot be fully stored, we adopt the stream storage solutions for fixed storage space proposed in [19]. The solution uses temporal aggregations over the data stream at multiple time granularities. The aggregation level for a data record is dependent on the age of the record, with only the most recent data fully stored. The aggregation levels and their granularity are decided before the network deployment and are dependent on the measured data and the storage size.

Each sensor node stores and manages locally its measurements. Each measurement has attached to it a timestamp corresponding to the time of measurement. Each type of sensor has associated a measurement interval, which defines the interval between successive data collections and is identical for all sensor nodes. We consider the data in the sensor network as a specialized distributed database, with temporal data stored in a node's database. Each node has a location, which gives spatial properties to data. Thus, on a global view, the sensor networks is a distributed database storing spatio-temporal data.

We are interested in processing historical spatio-temporal queries, denoted by *HSTQ(qID,sw,tw)*, where *sw* represents a spatial window, *tw* represents a temporal window and *qID* uniquely identifies a query. The answer to the query is formed by all sensor measurements from the given area *sw* during the time range *tw*. Sensor nodes have equal capabilities and therefore a query can originate at any node with query answers located in some (possibly all) of the nodes. Some sensor network scenarios [11, 17] consider the so-called long-running queries, where a user wants the continuously monitor the measured entities. We do not consider this type of query in this paper.

4 Spatio-Temporal Query Processing

Given a historical spatio-temporal query *HSTQ(qID,sw,tw)* at a sensor node, the problem is to efficiently locate and retrieve the answers, given the limited knowledge each node has about the overall network. As a major constraint on sensor nodes is their limited energy supply, we focus on energy efficient techniques. It has been shown that the energy required by sensing and computation is up to three orders of magnitude less than the energy used for communication [12]. Therefore we use the energy cost of communication as the measure of efficiency. This cost is proportional to the number and the size of exchanged messages.

In this section we discuss first a basic query processing algorithm for sensor networks. Next, we present an original framework for processing spatio-temporal queries and, within this framework, we propose three new algorithms.

4.1 Basic Query Processing Algorithm

A straightforward way to locate the query answers, which we call FullFlood, is contacting every network node. The query originator node 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. Due to query message broadcast, each node will receive the same query several times. For each query, a node processes only the first query message received, discarding subsequent query messages. When a query is received, the node first broadcasts the query, then it selects the locally stored data relevant to the query (if any), it 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 the relevant data from all nodes, it can answer the query. The messages flow for FullFlood algorithm is shown in Figure 1.

The FullFlood algorithm is guaranteed to find the query answer for a connected sensor network, but it incurs high communication costs due to the large number of messages required to contact all nodes. The algorithm is similar to a parallel breadth first search in a network graph, where sensor nodes are vertices and edges represent direct communication links between sensors. Assuming there is no communication delay, the query will reach each node on the shortest path (with respect to number of hops) from the query originator. As query messages are broadcast along all paths, the first message reaching a node must have travelled over the shortest path. After a query is processed locally, each node returns the answer to the neighbor it received the query from, and therefore answers are returned over the shortest path to the query originator.

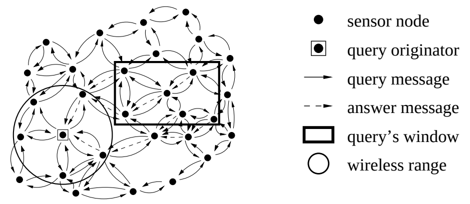

Figure 1: The FullFlood algorithm - message flow

4.2 Query Processing with STWin

If there is only one node relevant to the query, the optimal solution is contacting the node on the shortest path from the query originator and returning the answers over the same path. When the query answer involves several nodes, communicating with these nodes on the shortest path between the query originator and each of them is no longer optimal. Figure 2 shows an example. Forwarding the query over the shortest paths (routes (a) and (c)) requires 6 query messages in order to reach both relevant nodes, while route (b) requires only 5 messages. On the other hand, returning the nodes' answer over the shortest path is still optimal (assuming there is no aggregation of answers). As the energy usage is proportional to the message size and the same amount of answer data must be transfered over any of the possible return paths, sending the answers over the shortest path is the cheapest. Finding an optimal solution when there are several query relevant nodes requires each network node to know the network layout, as well as possibly expensive local computation for finding the optimal route for each particular query. Due to sensors' limitations, it is not feasible for each node of a large sensor network to find and store the full network layout, as well as make expensive processing.

On the other hand, contacting all sensor nodes as in FullFlood algorithm is not the most energy efficient approach.

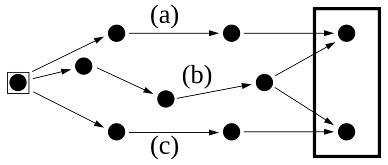

Figure 2: Routing example

A heuristic solution for query processing is contacting only the query relevant nodes, and a few extra nodes for routing the query and the answer if the query originator is not located inside the query's spatial window. A heuristic contacting only a subset of all network nodes should use a lower number of messages than FullFlood, which in turn may lead to lower energy consumption. An additional advantage of such a solution is reduced network congestion, which improves the query response time. Also, if only a subset of the network nodes is involved in processing each query, then several queries could be efficiently processed

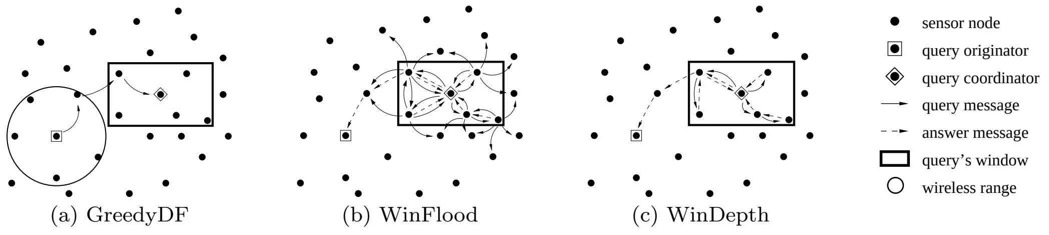

Figure 3: The algorithms within the STWin framework - message flow

simultaneously in different parts of the network. We propose the STWin (Spatio-Temporal WINdow) framework for query processing in which such a heuristic can be implemented. In this framework, we divide the query processing into two phases, one for locating a path from the query originator node to a sensor inside the query's spatial window, the other for gathering the query answer from the relevant nodes and returning it to the query originator.

- **Phase 1:** Given a query at a node N_Q, called query originator, we want to find a path to a node located in the query's spatial window. This node will assume query coordinator role N_C for Phase 2.
- **Phase 2:** The coordinator node N_C initiates the query processing within the query's spatial window. The processing algorithm must locate all relevant nodes, gather the results and return them to the query coordinator N_C. The coordinator will then return the answer to the query originator node N_Q on the routing path discovered in Phase 1.

These two phases form a general query processing framework, where various algorithms can be used in each phase. In the following we propose one algorithm for the first phase and two algorithms for the second phase.

4.2.1 Phase 1: GreedyDF

The *GreedyDF* algorithm uses a greedy technique to find a routing path from the query originator node to a node N_C located at the center of the query's spatial window. Other possibilities for choosing N_C exist, and which node is the best to select as N_C for a query is an open question. Choosing the center node is a good compromise between the likelihood of a heuristic to find at least a node in the query area and the length of the path over which answers from the coordinator node will be returned to the query originator node. The query originator forwards the query to its neighbor located closest to N_C, which in turn forwards the query to its neighbor closest to N_C, and so on. If node N_C is found, then node N_C initiates Phase 2. The routing may reach a sensor node that is closer to N_C than any of its neighbors, in which case the query cannot be forwarded. If the reached node is located in the query's area, the node assumes coordinator role N_C and initiates Phase 2, else an empty answer is returned. The *GreedyDF* algorithm uses a small number of messages, but it does not guarantee that a routing path to a node in the query's spatial window will be found. Greedy-based routing methods for position based routing in ad-hoc networks have been shown to nearly guarantee delivery for dense network graphs, but to fail frequently for sparse graphs [15]. Variants to this heuristic would include using a different neighbor selection method or backtracking the search when query forwarding cannot be done. We choose to not use backtracking solutions as they cannot guarantee answer location within a small number of steps, while ultimately degenerating to a slow network flood with higher communication costs due to the extra messages required for the backtracking steps. The message flow for the *GreedyDF* algorithm is depicted in Figure 3(a).

4.2.2 Phase 2: WinFlood and WinDepth

For the second phase of STWin we propose two algorithms. The *WinFlood* algorithm consists of a constrained parallel flooding, where a node broadcasts the query to its neighbors only if its own location is inside the query's spatial window. The constrained flooding starts at the query coordinator node N_C and stops when the query reaches nodes outside the spatial window. Figure 3(b) show the message flow for the *WinFlood* algorithm. The *WinFlood* algorithms is similar to a constrained parallel breadth first search in the network graph.

An alternative solution is the *WinDepth* algorithm, which is based on depth first search policy. In *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 ID 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 partial query answer, the node checks again if there is any other of its neighbors that is relevant to the query and has not received it yet. If there is such a neighbor, it forwards the query to this node and waits again for the neighbor's answer. This process is repeated until all of a node's neighbors located within the window have answered the query, at which point all the partial answers received are merged with the locally stored answers and the new partial answer is returned to the neighbor that the node received the query from. The message flow for the *WinDepth* algorithm is shown in Figure 3(c).

The *WinFlood* algorithm uses broadcast messages to forward the query, while in *WinDepth* nodes send individual messages to neighbors located within the window. As the cost of one broadcast message is generally lower than the cost for a group of one-to-one messages, it may be cheaper to use broadcasting and stop the query forwarding when an exterior node is reached. An advantage of *WinFlood* is that it is faster than *WinDepth* for the same number of contacted nodes and likely more cost efficient within a small window due to the use of broadcast messages. On the other hand, *WinDepth* contacts a smaller number of nodes,

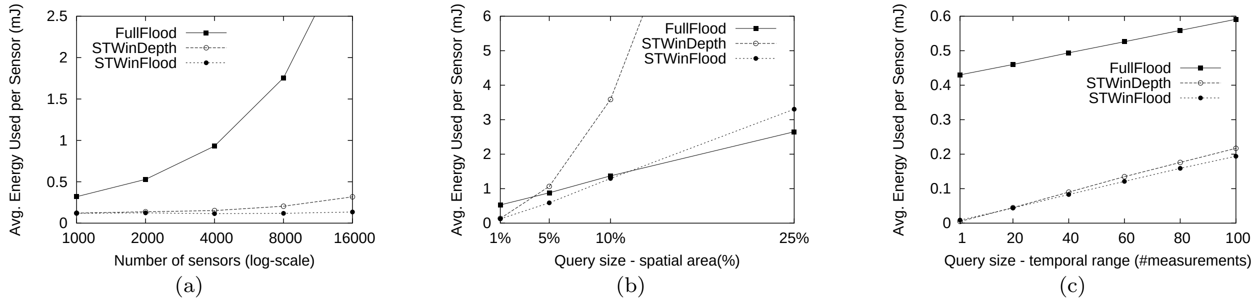

Figure 4: The effect of several parameters on the average energy used per network sensor for the investigated algorithms

Parameter	Default Value
Area covered	1000x1000 meters
Wireless range	50 meters
Number of sensors	2000
Tuple size <*value,time-stamp*>	8 bytes
Query size <*qID,sw,st*>	24 bytes
Query (spatial window)	1% (of area)
Query (temporal window)	60 measurements

Table 1: Parameters of query and sensor network

which makes more nodes available to answer other queries, and it causes less network congestion, which helps improve the query response time if several queries are processed simultaneously in the network.

In the following section we evaluate experimentally the proposed algorithms and discuss the effects of several factors on the energy used during query processing.

5 Experimental Evaluation

We implemented a sensor network simulator in order to study the performance of the presented algorithms. The sensors' placement follows a uniform distribution over a two dimensional region. We represent a historical spatio-temporal query *HSTQ* by the coordinates of a spatial area (*sw*), a temporal range (*tw*) and its query ID (*qID*). The query's spatial window covers 1% of the monitored region (that is 10% on each spatial coordinate), unless otherwise noted. The temporal window covers 60 measurements, where each measurement is represented by a <*value,time-stamp*> pair. A summary of query and sensor network parameters and their default values used in our experimental evaluation is presented in Table 1.

We compare the algorithms in terms of the average energy used per network node for communication while processing a query. According to [13], the energy used to transmit and receive one bit of information in wireless communication is given by:

$$\text{Energy}_{\text{transmit}} = \alpha + \gamma \times d^n \qquad (1)$$

$$\text{Energy}_{\text{receive}} = \beta \qquad (2)$$

where d is the distance to which a bit is being transmitted, n is the path loss index, α and β capture the energy dissipated by the communication electronics and γ represents the energy radiated by the power-amp. In our experiments, we use the following values for these parameters [3]: $\alpha = 45\ nJ/bit$, $\beta = 135\ nJ/bit$, $n = 2$, and $\gamma = 10\ pJ/bit/m^2$. As typical sensors do not have sophisticated communication electronics capable of adapting the transmission range [5], we assume all messages are transmitted as far as the wireless communication range.

For the algorithms within the STWin framework, we call STWinDepth the combination of *GreedyDF* with *WinDepth*, and STWinFlood the combination of *GreedyDF* with *WinFlood*. All experimental measurements are averaged over 100 randomly generated sensor networks, with 10 random queries over each network.

First, we investigated the effect of node density on the performance of *GreedyDF*. For networks with 2000 or more nodes, *GreedyDF* is able to find a routing path from the query originator to a node inside the query's spatial window for most of the tested networks. In the majority of the successful cases, the reached node is located in the proximity of the center of query area. To have a fair comparison, the following measurements consider the energy used by an algorithm while processing a query only when each algorithm located all answers for that query.

Figure 4(a) presents the effect of the number of sensors on the energy usage of each algorithm. As node density increases, FullFlood sends a larger number of messages to nodes not relevant to the query, which leads to higher energy costs. The increase in sensor density leads to an increase in the number of nodes holding relevant data, which affects the costs of all algorithms, as a larger answer set must be returned to the query originator. With more nodes available for routing, the *GreedyDF* algorithm may be able to find a shorter path to the query coordinator node, an advantage for both STWinDepth and STWinFlood as less energy will be used for locating the query coordinator and a shorter path is used to return the answers from the coordinator node to the query originator. On the other hand, the coordinator node will send a larger answer set to the query originator in both STWinDepth and STWinFlood. The increase in the number of relevant nodes affects more STWinDepth than STWinFlood. This is due to the depth first policy used by *WinDepth* for query routing, as this policy contacts most relevant nodes on one query forwarding path. This behavior causes the larger answer set to be returned over a longer path to the query coordinator, which increases the energy usage.

The negative effects of this behavior of STWinDepth can be also seen in Figure 4(b), where the query size is increased. A larger query area affects the FullFlood algorithm less than the other two methods as only the communication cost for returning the answers increases for this algorithm, while the energy used for locating these answers stays constant. With the query's spatial window increasing, STWinFlood uses flooding over a larger set of nodes, ultimately degenerating into the FullFlood algorithm for large query windows. In both STWinDepth and STWinFlood, the answers from a larger spatial window are sent back to the query originator over a longer path (as the answers are first collected by the coordinator node) compared to the FullFlood method, which returns all answers over the shortest path. For large query windows, FullFlood uses less energy per node than STWinFlood for 2000 nodes, but STWinFlood performs better than FullFlood for large queries in denser networks (the corresponding graphs are not shown due to space limitations).

Figure 4(c) shows the effect of a query's temporal range on the energy consumption. A variation in the query's temporal range only affects the size of the answer messages, and leads to a linear variation of the energy used by these messages. The increase in energy usage is the smallest for FullFlood as the algorithm returns the answers over the shortest path to the query originator. STWinFlood performs better than STWinDepth because the relevant answers are returned on the shortest path to the query coordinator in STWinFlood, and both algorithms share the answer return path (discovered by *GreedyDF*) from the coordinator node to the query originator.

6 Conclusions

While the technological advances have lead to sensors with reduced sizes and increased capabilities, the sensor data management is still in its incipient stages. The challenges are multiple, and the database research has to move its focus from considering time as a main optimization goal toward energy efficiency or a combination of both time and energy. The size of the database is no longer a primary challenge, with the focus moving to the distributed nature of the database and query processing.

In this paper we made a few steps toward energy efficient query processing in a sensor network environment where each sensor is aware of only its neighbors. In this scenario, we proposed the STWin query processing framework, where the query is first forwarded to a query coordinator node within the query's spatial window, followed by an efficient query processing involving only the relevant nodes. Within this framework, we proposed the *GreedyDF* algorithm for the first phase, and *WinDepth* and *WinFlood* algorithms for the second phase.

The experimental results showed that STWinFlood is more energy efficient in most situations than simple flooding as well as the solution involving just depth-first based algorithms. Only for very large query windows in networks with low sensor densities, the FullFlood algorithm performs slightly better in terms of energy usage, and it is more robust for locating all relevant answers (however, it causes network congestion, reducing the network's capability to process several queries simultaneously). STWinFlood performs slightly better than STWinDepth for small query windows, the differece in performance dramatically increasing for queries over large areas. An advantage of STWinDepth is that there are at most two nodes working in each query processing step, which allows the rest of the network to process other queries or simply sleep to save energy. For most cases, STWinFlood has shown low energy usage consistently, and therefore we recommend it for sensor networks where each node is only aware of the other nodes located within its wireless range. The STWinFlood combines the strengths of both depth first and breadth first techniques while limiting their drawbacks.

In this paper we introduced techniques for query processing when the user in interested in retrieving all the relevant information. In other situations, an aggregated query answer may be sufficient. We are currently investigating new algorithms within the STWin framework that would allow efficient in-network aggregation during query processing.

Acknowledgments. This work was partially supported by NSERC. We would like to thank Ioanis Nikolaidis for his valuable comments and fruitful discussions.

References

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

[2] J. Beaver et al. Power-aware in-network query processing for sensor data. In *Proc. of Hellenic Data Management Symposium*, pages 1–17, 2003.

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

[4] P. Bonnet et al. Towards sensor database systems. In *Proc. of IEEE MDM*, pages 3–14, 2001.

[5] A. Demers et al. Energy-efficient data management for sensor networks: A work-in-progress report. In *Proc. of Upstate New York Workshop on Sensor Networks*, 2003.

[6] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of Intl. Conf. on Peer-to-Peer Computing*, pages 32–39, 2003.

[7] Q. Fang et al. Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo Alto Research Center, 2002.

[8] M.A. Hammad et al. Stream window join: Tracking moving objects in sensor-network databases. In *Proc. of SSDBM*, pages 75–84, 2003.

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

[10] S. Madden et al. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of OSDI*, pages 131–146, 2002.

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

[12] V. Raghunathan et al. Energy aware wireless microsensor networks. *Signal Processing Magazine*, 45(2):40–50, 2002.

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

[14] S. Shakkottai et al. Unreliable sensor grids: Coverage, connectivity and diameter. In *Proc. of INFOCOM*, 2003.

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

[16] Y. Xu and W.C. Lee. Window query processing in highly dynamic sensor networks: Issues and solutions. In *Proc. of Workshop on GeoSensor Networks*, 2003.

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

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

[19] D. Zhang et al. Temporal and spatio-temporal aggregations over data streams using multiple time granularities. *Information Systems*, 28:61–84, 2003.