AN ENERGY AWARE SENSOR NETWORK QUERY PROCESSING SYSTEM

MOHAMED WATFA[1]
WILLIAM DAHER[2]
HISHAM AL AZAR[2]

[1]University of Wollongong
mwatfa@uow.edu
[2]American University of Beirut
(wsd03,hga08)@aub.edu.lb

Abstract. Recent developments in hardware have enabled the widespread deployment of sensor networks consisting of small sensor nodes with sensing, computation, and communication capabilities. Sensor data are subject to several sources of errors resulting from power limitations, wireless communication, latency, throughput, and various environmental effects. Such errors may seriously impact the answer to any query posed in the network. An energy efficient approach to query processing is introduced in this paper by implementing new optimization techniques applied to in-network aggregation. We first discuss earlier approaches in sensors data management and highlight their disadvantages. We then present our approach and evaluate it through several simulations to prove its efficiency, competence and effectiveness.

Keywords: Sensor Networks, Data Base, Data Fusion, Aggregation, Energy Efficiency

(Received October 02, 2008 / Accepted January 19, 2009)

1 Introduction

Recently IC and MEMS have matured to the point where they enable the integration of communications, sensors and signal processing all together in one low-cost package. It is now feasible to fabricate ultra-small sensor nodes that can be scattered to gather useful information. The events detected by these nodes need to communicated to gateways or users who tap into the network. This communication occurs via multi-hop routes through other sensor nodes. Since the nodes need to be unobtrusive, they have a small form-factor and therefore can carry only a small battery. As a result, they have a limited energy supply and low-power operation is a must. Sensor networks have become an important source of data acquisition with numerous applications being developed in monitoring various real-life phenomena. Unfortunately, sensor data is subject to several sources of errors resulting from power limitations, wireless communication, latency, throughput, and various environmental effects. In-network aggregation is a well known technique to achieve energy efficiency when propagating data from information sources to multiple sinks. The main idea behind in-network aggregation is that, rather than sending individual data items from sensors to sinks, multiple data items are aggregated as they are forwarded by the sensor network. Data aggregation is application dependent, i.e., depending on the target application, the appropriate data aggregation operator (or *aggregator*) will be employed. From the information sink's point of view, the benefits of in-network aggregation are that in

general it yields more manageable data streams avoiding overwhelming sources with massive amounts of information, and performs some filtering and preprocessing on the data, making the task of further processing the data less time and resource consuming. Be-

cause of its well-known power efficiency properties, in-network aggregation has been the focus of several recent research efforts [3][6][9]. In contrast, our approach is a general one unrelated to any application. Although all researches and approaches that were done earlier in this field provide some advantages over traditional centralized approaches, they still face some pitfalls and disadvantages. Studying the disadvantages of previous work, we concluded that we need to work on several major areas such as: power consumption, reliability, less aggregation overhead, less contention, fault-tolerance, and concurrency. The queries should also consider data integrity, security and efficiency. Algorithms should take into consideration that nodes might have unequal battery levels opposed to unrealistic assumptions made in previous work. Taking into consideration these facts, our algorithm should evaluate the remaining power on each node and the varying power consumption that might differ from node to node to be able to maintain longer network lifetime to transmit useful data. We propose a new combinational improvement of all the available solutions taking into consideration the above constraints to develop the sensor node's ability to handle data locally in a very efficient way. The rest of the paper is organized as follows: Section 2 presents previous approaches focusing on their disadvantages. Section 3 presents the query processing approach. In Section 4, we evaluate and simulate and analyze our proposed algorithms using our own simulator. We conclude this paper in Section 6 with possible future improvements.

2 Related Work

Many researchers such as Yao et al., Bonnet et al., Gray et al., and Madden et al. [1][10][8][5] tackled the data management topic in wireless sensor networks including query processing and data handling but none generic useful results and findings were originated for implementation. There has been a lot of work and approaches on query processing in distributed database systems [10][8], but most related work on distributed aggregation did not consider the physical limitations of sensor networks. In addition, the TinyDB Project at Berkeley [7] conducted by Madden et al. also investigates query processing techniques for sensor networks including an implementation of the system on the Berkeley motes and aggregation queries. The basic approach used in both TinyDB [7] and TAG [4] is to compute a partial state record at each intermediate node in the routing topology. During the epoch after query propagation, each mote listens for messages from its children during the interval it specified when forwarding the query. It then computes a partial state record consisting of the combination of any child values it heard with its own local sensor readings. Finally, during the transmission interval requested by its parent, the mote transmits this partial state record up the network. Previous studies [7][4] have shown that aggregation dramatically reduces the amount of data routed through the network, increasing throughput and extending the lifetime of battery powered sensor networks as less load is placed on power-hungry radios. Previous simulation studies have shown that aggregation can reduce energy consumption by a factor of 5 in a large network (150-250 nodes) with five active sources and five sinks. Previous networking research [2] approached aggregation as an application specific technique that can be used to reduce the amount of data that must be sent over a network. In a previously proposed data dissemination scheme (directed diffusion with opportunistic aggregation), data is opportunistically aggregated at intermediate nodes on a low latency tree. In [3], the authors explore and evaluate greedy aggregation, an approach that adjusts aggregation points to increase the amount of path sharing. The greedy aggregation approach was implemented. Greedy aggregation differs from opportunistic aggregation in path establishment and maintenance. To construct a greedy incremental tree, a shortest path is established for only the first source to the sink whereas each of the other sources is incrementally connected at the closest point on the existing tree. In [6], they didn't explore all their techniques relative to mobility, and multiple queries. Thus, we can't be sure if their techniques are more efficient and reliable than the old techniques. In addition, they mentioned that in some cases that in-network aggregation performs worst than even the simplest approach "the naïve approach" [10][6]. Most of the conclusions that the above researchers are credited for can be described as: *"We described a vision of processing queries over sensor networks"* [9]. Some presented a prototype or some techniques they used without any actual implementation and simulation results. For example, the Cornell COUGAR system prototype [9] is a first effort towards sensor database system. Thus, a lot of improvements are still needed in this field to achieve better generic approaches for implementation in wireless sensor networks plus taking into consideration all the drawbacks and pitfalls of earlier techniques. Unlike other networks, wireless sensor network still need an international standard ISO to be build upon and all these future researches are directed towards this goal. One part is related to finding a general applicable approach for data management in sensor networks which will become a self-aware, self-configuring and reliable system

with respect to all nodes' resource constraints. As various groups around the country have begun to deploy large networks of sensors, a need has arisen for tools to collect and query data from these networks. Of particular interest are aggregates, operations which summarize current sensor values in some or all of a sensor network. For example, given a dense network of thousands of sensors querying temperature, users want to know temperature patterns in relatively large regions encompassing tens of sensors where individual sensor readings are of little value. Sensor networks are limited in external bandwidth, i.e. how much data they can deliver to an outside system. In many cases the externally available bandwidth is a small fraction of the aggregate internal bandwidth. Thus computing aggregates in-network is also attractive from a network performance and longevity standpoint: extracting all data over all time from all sensors will consume large amounts of time and power as each individual sensor's data is independently routed through the network. As noted before, aggregation dramatically reduces the amount of data routed through the network, increasing throughput and extending the life of battery powered sensor networks as less load is placed on power-hungry radios. Also, the fact that every message is effectively broadcast to all other sensors within range enables a number of optimizations that can significantly reduce the number of messages transmitted and increase the accuracy of aggregates in the face of transmission failures.

3 The Query Processing System

Our approach consists of providing a new distributed algorithm for query processing in wireless sensor networks which is an optimized energy efficient distributed algorithm with respect to all the sensor's resource constraints. Some similarities to recent approaches are also used such as upgrading the TinyDB approach, an ACQP engine that is a distributed query processor which runs on each of the nodes in a sensor network, and the TAG approach. Our goal is to provide significant reductions in power consumption through reducing the number of query related messages in the whole network. Low energy consumption, and limited storage and memory usage are the three main constraints which we focus on in our approach. This section will provide a detailed explanation of our approach by presenting the problem and the corresponding solution. We evaluate the approach in the next section through simulation. Sensor networks have very limited power, small memory computational power and limited bandwidth so some possible problems including decreasing the overall power consumption of the data management algorithm, minimizing the number of collisions, and establishing a self adaptive network. Our first aim is to decrease the packet size and the second is to decrease the number of packets sent. To decrease the packet size each sensor should have values in its buffer of all its children nodes to perform partial aggregation before sending this value to its parent. In large sensor networks, aggregation of data having small packets and small values decreases the power consumption and the computation overhead. Our second approach is to index the network so to be able to query data with minimum number of exchanged packets. We will start by building an index tree (IST) that is similar to the SRT of TinyDB but for not only fixed attributes but also variable ones. The problem with building such a tree is the maintenance overhead, but we will prove that our algorithm maintains the tree with little or no maintenance overhead. Our algorithm starts by building an index routing tree. Then, each child in the tree sends its index to its parent. Since the parent knows the number of children it has, it compares the indexes received from each child, if they are equal, the parent indexes itself with their index and sends the index to its parent otherwise it does nothing. Upon a change in the index of one node, this node sends the new index to its parent, the parent checks again to see if the indexes are equal; if not and this parent is indexed, it removes its index and informs its parent, but if this parent is not indexed, it doesn't have to inform its parent. With our network, indexing a query could take less time and computation power to return the result. For example if we have a query that asks for the average temperature where the temperature is above 36. When this query reaches a node with index 1, the node doesn't forward the query to its children. Our third idea is to conserve energy as much as possible using indexing with the power evaluation criteria available in TinyDB at each node. We can use an index of 0 to note that a certain node is low in power try eliminate it in the execution flow of the query.

3.1 Building the Routing Tree

After the nodes are randomly deployed, an index routing tree is built. The routing tree is built as follows. The closest node to the base station is chosen to be the root of the tree *(level = 0)*. Once chosen, the root broadcasts requests containing its level to all its one hop neighbors (within its transmission range). When receiving the request, a neighbor node assigns itself a *level = level + 1* and chooses its parent to be the level up node from which it received the request, then re-broadcasts new requests containing its new level to all its neighbors and so on until no neighbors are found; thus the last nodes

become leaf nodes. Whenever a node receives two requests from two different nodes, if it has a level, it discards the second request; and if not, it selects the first arrived request. Thus it chooses one parent and one new level *(level = level + 1)*. In our tree algorithm, we intend to let every node have only a single parent. After building the tree each node sends its reading value to its parent starting from the leaf and up. Every node stores its last sent value. Every parent node receiving values from multiple children calculates the average of the values received and sends it to its parent and so on until the value reaches the base station.

3.2 Building the Index Table in the Routing Tree

When the base station receives the values, it sends a packet containing the index table to all the nodes. The first time the index table is sent, the value ranges of each index will be large; the reason behind this approach is not to send large packets in the network. If the index table is large, it may lead to collisions. When a node receives the index table, it compares its readings to the index table and indexes itself accordingly. In the second round the index table changes as value for index ranges becomes smaller. After couple of rounds the index values will be more accurate. The number of rounds depends on the size of the index table decided once the network is deployed. The final index table will be derived on each node. Deciding on the index ranges of the system depends on the type of sensor node; sensor nodes with readings that vary in large ranges should have index ranges with large values. The child sends its index to its parent. When a parent receives an index from its children, the parent compares all its indexes with its own, if they are all similar, the parent indexes itself as such. However, if all indexes are not the same, the node examines the percentage of the similarity, if the similarity is larger or equal to $75\div$ *(based on the simulation results in Section V)*, it indexes itself with the dominant index and ignores the others. If the index similarities are lower than $75\div$ then the parent indexes itself as between the smallest and largest index After all the nodes are indexed in the network, the parents and children agree on a common value.

3.3 Common Value Agreement (CV)

After a parent receives values from its children, it first calculates the average of the values; it stores the calculated average and sends it back to its children. We call this value *"common value"* and we denoted it by *cv* (Figure 8). Each node stores two values: the *cv* of its parent and the *cv* of its children (except leaf nodes). When a node needs to send a new reading to its parent, it subtracts the *cv* from its reading and forwards the value to its parent. The *cv* will be updated in case there is a major change in the average of the children. When a parent notices a large change between its children and the *cv*, the parent resends the new average to its children as the new *cv*. With our *cv* approach, sent packets are smaller and therefore leading to less collisions, more energy efficiency and less calculation overhead.

3.4 Aggregate Functions Evaluation

The calculation approach defers between different aggregate functions. In our algorithm, we evaluate the 5 basic aggregate functions Sum, Average, Count, Min, Max. We start with the Average function. To elaborate more on this function let us examine the following query:

SELECT Avg(temp)

FROM Sensors

Since this query asks for the average temperature of the whole network, the query should reach all the nodes where values will be extracted. In our algorithm this is not the case; our algorithm offers the user two approaches to calculate the average. In the first approach, when the query reaches the root node, the root node doesn't forward the query to its children but returns his *cv* since the *cv* is the common average between it and its children. In the second approach the query reaches all the nodes but not all the nodes return a value. When a query reaches a node, the node examines its current reading and index. If his current reading still lies within the same index the node doesn't forward any value since his value will not have a noticeable change to the final result. If the current reading doesn't lie in the same index the node changes its index, and sends the *cv* subtracted from his reading to the parent node. After receiving the new reading the parent notices a large value from his child thus updates his index status and *cv* if needed according to the previously discussed approaches. Then the parent node calculates the new average. Assume *Avg* is the old average value, Avg_{new} the new average, *nv* the new value received from the child and p the children count involved in the query. The parent calculates the new average using the following formula: $Avg_{new} = \frac{(Avg \times p)+nv}{p}$

In the second approach, sending the value depending on the index change decreases the overhead of sending packets where the change in reading will not cause a notable change to the overall value; thus, using this approach results in sending a small number of packets.

Deciding on what approach to use depends on how accurate the data needs to be. The Count function is evaluated in a normal approach where the node, if meeting the criteria, sends 1 to his parent where the parent adds the count of his children and forwards them to his parent and so on. The Sum function can also be evaluated using two approaches. The first approach is the usual one where values are sent to the parent node that in his turn sums them and sends them to his parent and so on. The second approach of evaluating sum is to break the Sum query into two queries, an average query and a count query. In this approach the advantages of average evaluation discussed previously can be used. After a node receives a sum function it sends it's reading as if it is calculating the average and then sends the count. The base station calculates the Sum as $Average \times Count$. Deciding on the approach to use depends on the query and the exactness of the result. Our engine on the base station decides what approach to use. The Min and Max function are evaluated in similar approach to the average where the node sends the *cv* subtracted from his value. The parent node in its turn chooses the largest or smallest (Min or Max) value received, adds to it the children *cv* then subtracts from it its parent *cv* and sends it to its parent.

3.5 Queries with conditions

For other types of queries that have a condition, our approach should increase the throughput of the query since indexing will help in the injection of the query. Let us elaborate more by examining the following query:

SELECT Avg(temp)

FROM Sensors

WHERE temp>35

Our engine on the base station will parse this query and translate the condition into index. The condition ***"Where temp>35"*** will be translated into ***"Where tempIndex > 5"*** assuming index 5 and its preceding indexes are between 0 and 30. After this translation the query is injected into the network. From the root and on, every parent node checks if it has an index smaller or equal to 5, if yes, it will not forward the query to its children. Thus the query is filtered through the injection state. The root broadcasts the query to its children. Once arrived to each child, they check if they have an index smaller or equal to 5. Thus, for node having the index 5, it ignores the query request, but in the case of the other node, it re-broadcasts the request to all of its children (index > 5) which in their turn, each of them forwards the request if its index is greater than 5. This approach removes the overhead of sending the query to unneeded nodes. This approach increases the energy efficiency of the network where nodes that do not satisfy the condition will not need to spend energy since filtering is happening in the injection of the query rather than the base station.

3.6 Power Management

Power consumption and network life time are major issues in the wireless sensor network design. In our algorithm, we try to increase the lifetime of the network through two different approaches. In the first approach, the node keeps track of the number of messages sent and number of messages received. From these numbers the node can approximate the energy consumed and therefore the amount of energy left. We can also incorporate a battery model in our algorithm. When the node reaches a state where its energy is close to a predefined threshold, it informs its parent. The parent will therefore decrease the number of packets sent to this child thus will send packets to this child every two rounds rather than every round hence decreasing the transfer data rate to this child. If the child reaches a very low state of energy, it informs its parent where the parent stops sending any packets to this child. The second approach is achieved by changing the root parent every two rounds. If a child can have multiple parents, the child, after couple of rounds, changes to another parent if the other parent has more energy than his current parent. When a parent is low on energy it informs its children, the children in their turn will ask another parent node if they could join it. If another parent with more energy is found, the child switches to this new parent. A child could know if the new parent has more energy than his old one by knowing the amount of energy consumed by both parents derived from the formula discussed before. We will just incorporate the first approach in our simulator leaving the second approach for future work. The first approach should reduce the energy consumption of every single node in the network thus increasing the network lifetime.

3.7 Query Optimization

In this section, we will discuss our approach in optimizing a query. The base station keeps record of the last queries with a time stamp. Upon issuing a new query, the query optimizer checks for similar queries issued before and their results. If the results are close, it concludes that the network readings are not changing so instead of sending the query to the whole network, the

query is sent to different parts of the network. To send the query to different parts of the network, the query optimizer sends the query to the *level 1* parent nodes which in turn will choose some of their children and forward them the query. The number of children nodes chosen depends on the query optimizer decision. With this approach a smaller amount of nodes participate in the query. This approach increases energy efficiency and throughput of the network but gives an approximation of the result. This optimization technique doesn't apply on all kinds of queries.

4 Simulation Results

To test the efficiency of our algorithm, we decided to model our own simulator to achieve our goals because of the lack of database simulators. Our simulator is written in VB. We randomly deploy a large number of nodes, then a routing tree is build in which the query is sent from the root to the leaf nodes to be evaluated, processed. The leaf nodes will send the results back to their parents where they are aggregated and sent over to the parent's parents until an aggregated value reaches the root which in its turn, sends the aggregated value back to the base station. In the tree, each node is randomly colored to present its level *(number of hops away from the base station)*. An edge connects two neighbor nodes if there are in the communication range of each other i.e. they can communicate by sending and receiving messages. Every node contains a cache in which it saves its level number, its index, its parent id and its children's ids. Our Algorithm will be evaluated based on two basic metrics: power consumption and network lifetime. The performance of the algorithm over time will also be studied to determine the benefits of using in-network aggregation. This is done by assuming that each sensor node has a limited energy supply of and is deactivated when the available energy is used up. In addition to that, we will incorporate the energy wastage resulting from building the IRT (index routing tree).In our simulation, we are going to randomly spread about 500 sensor nodes in a 10000 x 10000 region in VB to investigate the change in temperature and humidity. We are going to query these nodes to get Max, Min, and Average values. We compare our approach (Energy Efficient Indexed Aggregation, EEIA) with the: 1- *Naïve algorithm* where all the query results from each node are sent to its parent until the results reach the root where they are aggregated and sent to the base station. 2- *Simple TinyDB* where the results are aggregated at each intermediate node until reaching the root. In our simulation, we assumed the energy wasted is 1μJ for sending a single bit and 0.5μJ for receiving a single bit. Initially, each node has 1 J of available energy. In our simulation, we also incorporated the energy consumption of building an index tree. As you can see in Figure 1, as the number of nodes increases, the energy consumption increases linearly since all nodes participate in building the query with same amount of energy. We concluded that the energy consumption of building the index tree is equivalent to initiating one query in the network. As for the maintenance of the index tree, we see from Figure 2 that the average energy consumed in the maintenance depends on the readings of the nodes in the network. If the network readings change significantly in a small amount of time, the energy consumed in maintaining the tree increases. On the other hand, if the network readings change slowly then the energy consumption of the index maintenance decreases. The energy consumption in the first 15 seconds is high since the index tree was being built. We issued a number of different queries on 500 nodes and compared the energy consumption, delay and number of instructions using our approach compared to the normal approach of broadcasting the query to the network. To simulate the same queries, we implemented in our simulator two approaches, the first approach queries the network by broadcasting the query to all the nodes and aggregating the results back to the base station. As for the second approach, we added our index querying approach. To make our simulation more realistic, we maintained the same condition on the network while using the different querying approaches. In Figure 3, we compared the energy consumed for the same queries using the 2 approaches. We issued 12 different queries and calculated the energy consumed by these queries to return the result. As can be seen from the graph, all queries using the indexed approach consume less energy than using the other approach. You can also note from the graph that some queries have energy consumption that is close to the normal approach while others have larger energy consumption. This difference depends on the conditions of the issued query since more selective queries tend to have a larger advantage using our approach (*indexing with more selectiveness decreases the number of messages sent and received in the network)*. In Figure 4, we compare the number of packets sent for the same query using the 2 approaches. As the figure shows, the number of packets sent decreases with the indexing approach since some nodes will not forward the packets to their children if their children don't satisfy the query conditions. As can be seen in Figure 5, the delay is decreased using our approach since as discussed before the number of packets sent decrease hence collisions decrease. In the delay simulation, we assumed a pack-

ets needs 0.01 sec to be resent. In the last simulation, we added the *cv* (*common value*) approach to our simulator and compared the lifetime of the network using indexing and *cv* querying to the normal approach. As can be seen from Figure 6, the first node dies after 83 queries in the normal approach while using our approach; the first node dies after 130 queries. This increase in the lifetime of the network is due to two factors. The first factor is that nodes are sending fewer packets using the indexed approach thus less energy consumption per node. The second factor is the decrease in packet size with *cv* approach where less energy is consumed in sending the packet.

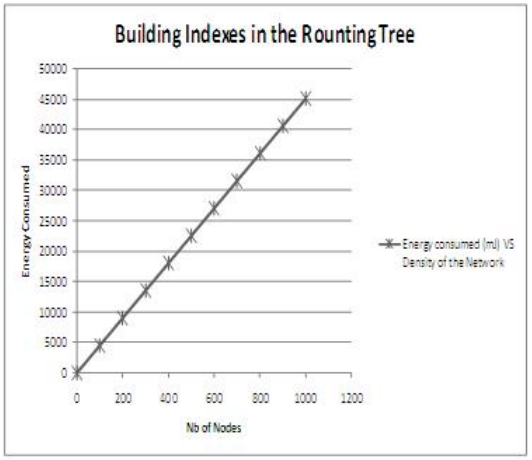

Figure 1: Building the routing tree

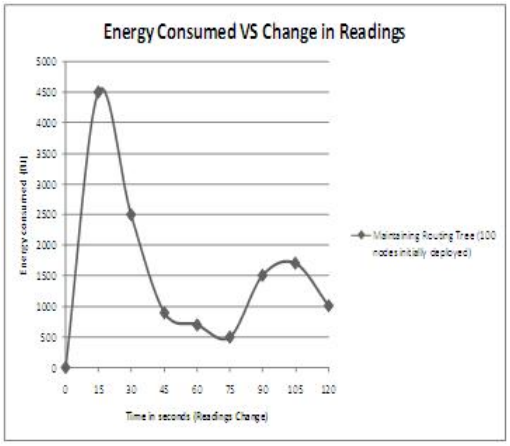

Figure 2: Maintaining the routing tree

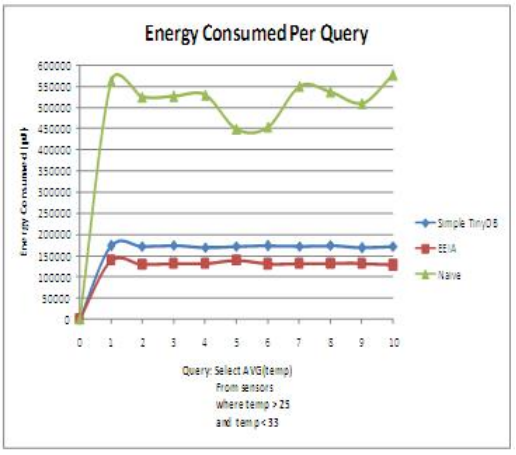

Figure 3: Energy consumption per query

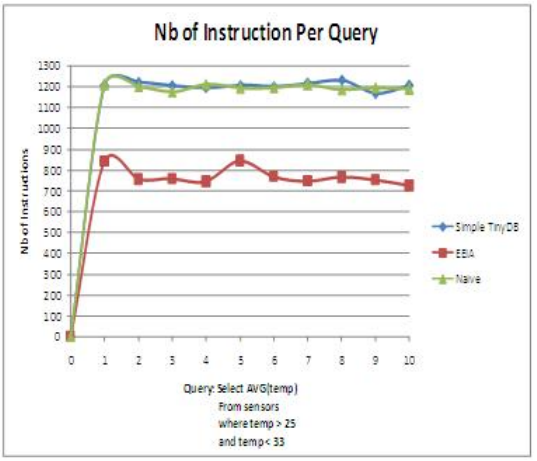

Figure 4: Number of instructions per query

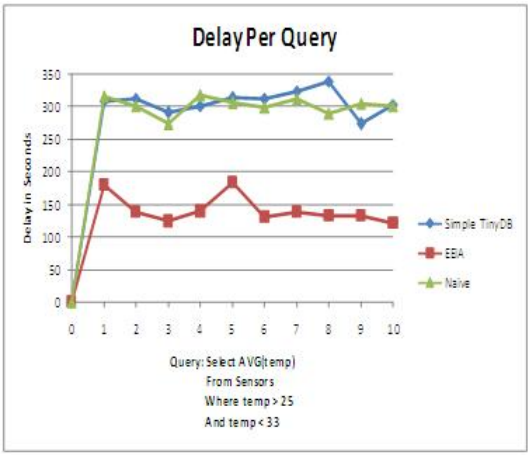

Figure 5: Time delay per query

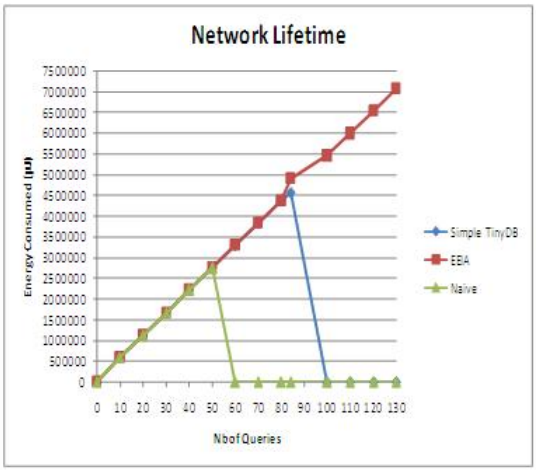

Figure 6: Network lifetime

5 Conclusions

In summary, we have showed how aggregate queries are efficiently executed over wireless sensor networks in a distributed manner. We have proved that our in-network distributed approach performed better in terms of energy reduction and network lifetime than the naïve and simple TinyDB approaches. Furthermore as for future work, our approach should confront with the difficulties of topology construction, data routing, loss tolerance by including several optimization techniques that further decrease message costs and improve tolerance to failure and loss. In addition to implementing these techniques, we need to rethink some of these techniques to present more efficiency to network changes and external factors which could affect our approach such as node mobility, obstacles and other issues. We could also extend our simulator to incorporate a 3D tree construction technique plus other methodologies mentioned above.

6 Acknowledgment

The authors would like to gratefully thank Miss Farah Abou Shahla for her help in submitting the final version of this paper.

References

[1] Bonnet, P., Gehrke, J., and Seshadri, P. Towards sensor database systems. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 3–14, London, UK, 2001. Springer-Verlag.

[2] Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. Building efficient wireless sensor networks with low-level naming. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 146–159, New York, NY, USA, 2001. ACM.

[3] Intanagonwiwat, C., Estrin, D., Govindan, R., and Heidemann, J. Impact of network density on data aggregation in wireless sensor networks. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 457, Washington, DC, USA, 2002. IEEE Computer Society.

[4] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[5] Madden, S., Franklin, M. J., Hellerstein, J. M., and Hong, W. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM.

[6] Madden, S., Szewczyk, R., Franklin, M. J., and Culler, D. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 49, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Madden, S. R., Franklin, M. J., Hellerstein, J. M., and Hong, W. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[8] Report, N. W.-I.-P., Demers, A., Gehrke, J., Rajaraman, R., Trigoni, N., and Yao, Y. Energy-efficient data management for sensor. In *2nd IEEE Upstate New York Workshop on Sensor Networks, comlab.ecs.syr/edu/workshop*, 2003.

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

[10] Yao, Y. and Gehrke, J. Query processing for sensor networks, 2003.