

Multi-Threshold Based Data Gathering Algorithms for Wireless Sensor Networks

Jun-Zhao Sun

Academy of Finland

Department of Electrical and Information Engineering, University of Oulu

Email: junzhao.sun@ee.oulu.fi

Abstract— Sensor networks have recently attracted significant attention for many military and civil applications, such as environment monitoring, target tracking, and surveillance. A high-level abstraction of sensor networks forms the distributed database view, in which query is adopted to retrieve data from the network. Continuous query is an important approach that is commonly used for collecting real-time data. Sensor nodes have limited energy resources and their functionality continues until their energy drains. Therefore, query for sensor networks, and especially continuous query, should be wisely designed to extend the lifetime of sensors. This paper presents a query optimization method for continuous data gathering in sensor networks. The method is based on the flexible setting of thresholds and the analysis on the content to be sent over the network. Thresholds represent the accuracy requirement of the application issuing the queries. Thus at each single sensor node, instead of direct delivery of each result data, data communication is needed only when the current data is out of the range constrained by the thresholds. A series of static and dynamic algorithms are proposed for the flexible setting of threshold values. In particular, the paper studies deeply the situation where phenomena under monitoring appear a periodical feature. An initiative data model is first created, and then at each single sensor node, instead of direct delivery of each result data, data communication is needed only when the current data is out of the range constrained by the periodical data model. The performance of the proposed algorithms shows that the proposed methods can achieve better performance than without performing the optimization.

Index Terms—Wireless sensor networks, continuous query, aggregation, threshold.

I. INTRODUCTION

Sensor networks represent significant improvement over traditional sensors in many ways [1, 2]. However, sensor nodes have very limited supply of energy, and should be available in function for extremely long time (e.g. a couple of years) without being re-charged. Therefore, energy conservation needs to be one key consideration in the design of the system and applications. Extensive research work has been devoted to address the problem of energy conservation. Examples include energy efficient MAC protocol [3], clustering [4], localization [5], routing [6], data management [7], applications [8], etc.

A sensor field is like a database with dynamic,

distributed, and unreliable data across geographically dispersed nodes from the environment. These features render the database view [9-11] more challenging, particularly for applications with the low-latency, real-time, and high-reliability requirements. Under a database view, a wireless sensor network is treated as a virtual relational table, with one column per attribute and one row per data entry.

Sensor network applications use queries to retrieve data from the networks. The result is a logical sub-table of the whole virtual table of the network, with each data entry an associated timestamp to denote the time of measurement. The real data table of a node is different from the one in the virtual table in the sense that there will be only one attribute there.

Query processing is employed to retrieve sensor data from the network [12-14]. A general scenario of querying sensor network is, when user requires some information, he or she specifies queries through an interface at sink (also known as gateway, base station, etc.). Then, queries are parsed and query plans are made. After that, queries are injected into the network for dissemination. One query may eventually be distributed to only a small set of sensor nodes for processing. The end nodes then execute the query by sampling phenomena/object. When sensor node has the sampling data ready, results flow up out of the network to the sink. The data can then be stored for further analysis and/or visualized for end user. Query optimization can be made through out the process in all the stages.

Continuous query is commonly used for collecting real-time data from the network, which produces much data communications than other queries like snapshot or event based query. This paper presents a novel query optimization method for continuous query execution in wireless sensor networks, and in particular, for the last stage of query processing: query result collection. The key novelty of the method lies in the compromise consideration of the requirement on accuracy along with energy consumption in data communication. Accuracy as an application level QoS requirement is represented in our methods with various threshold settings. The real data sending is determined by analyzing the content of the result data. By taking advantage of the accuracy constraint (i.e. threshold items) specified with a query, the method can find the optimal sensing and transmission

of attribute readings to sink node in terms of energy consumption.

In particular, we consider a special category of tasks where the phenomena under monitoring appear a periodical feature. In this case, a periodical data model is first created, and then further collected data will be sent only the matching with the data model fails. In this way, energy consumption can be reduced to the best extent.

The remainder of this paper is organized as follows. Section II formalizes the problem. Section III overviews the main idea. Section IV and V present static- and dynamic-threshold processing algorithms respectively. Section VI the proposed coarse-grain query processing algorithms for periodical phenomena. Performance of the proposed method is evaluated in Section VII. Section VIII analyzes related work. Section IX concludes the paper.

II. PROBLEM ANALYSIS

Suppose a number of N sensor nodes, s_1, s_2, \dots, s_N , have been deployed into an interesting area to accomplish a monitoring task. All the sensor nodes are homogeneous, and equipped with a number of M sensors of different modals. One specific node s_l , known as the *sink* or *gateway node*, interacts with end applications for the purpose of injecting queries and collecting results. We assume a spanning tree has been constructed over the whole sensor networks for the purpose of routing: disseminating queries and gathering data.

A continuous query is injected into the sensor network. This query contains following elements:

1) A number of m ($m \leq M$) data attributes selected from the M modals of the sensors attached to one the sensor nodes, which are required by the applications.

2) A sensor selection predicate, which specifies the Boolean condition of which sensor nodes are selected to involve in the task. The condition can be for any data attributes or node information like location and node ID. The selection results in a number of n ($n \leq N$) sensor nodes selected from the total N nodes.

3) A series of temporal information, which specify the time to start sampling (t_0), sample interval (Δt_i), window size (length, T_w), and report interval (i.e. sliding increment, Δt_r). For one selected sensor node s_i ($i \in [1, n]$) at time $t_j = t_0 + j \cdot \Delta t$, the readings of the node is a m -dimensional vector:

$$\vec{v}_i(t_j) = (v_{i,1}, v_{i,2}, \dots, v_{i,m}).$$

We denote the number of samples covered by the window size with K , so that

$$T_w = K \cdot \Delta t.$$

Sliding window is defined for aggregation operations as explained below.

4) Defined operations on the collected data, i.e. the final results to be reported. Operations are those aggregation functions, including AVERAGE, SUM, MAX, MIN, MEDIAN, COUNT, etc. There are two sorts of aggregation functions, local aggregation and global aggregation. Local aggregation is performed on local sensor readings in one window, while global aggregation

is performed over all the data produced by all the selected sensor nodes. If for example the aggregation function is AVERAGE, then local aggregation for node s_i at report time t_j can be denoted as

$$\text{WINAVG} \left(\frac{1}{K} \sum_{k=0}^{K-1} \vec{v}_i(t_{j-k}) \right);$$

Global aggregation can be denoted by

$$\text{WINAVG} \left(\frac{1}{nK} \sum_{i=1}^n \sum_{k=0}^{K-1} \vec{v}_i(t_{j-k}) \right).$$

On the other hand, there are two sorts of continuous query aggregation functions, stepwise aggregation and direct aggregation. Stepwise aggregation includes for example MAX, MIN, SUM, COUNT, AVERAGE. One feature of this sort of aggregation is that, the aggregation result can be obtained gradually with partial data set. This means the aggregation can be performed with partial data, and does not have to wait for all the data available. On the contrary, direct aggregation is more complicated and different from stepwise aggregation in the sense that, direct aggregation cannot be performed until all the aggregation data is available. In other words, it cannot be executed above partial data.

For instance, consider COUNT and MEDIAN as the examples of stepwise and direct aggregations respectively. The most crucial difference between sliding window based continuous MEDIAN query with COUNT query over stream data is that, COUNT query is a monotonic and summary aggregate which means its value can only get larger as more values are aggregated, while on the other hand allows partial aggregation with other count values. Instead, MEDIAN is an exemplary aggregate computing some property over the entire set of values, and therefore does not allow partial aggregation. In other words, the aggregation cannot be performed until all the concerned data is available.

For local aggregation, stepwise aggregation needs less memory than direct aggregation. For global aggregation, in-network processing is possible for stepwise aggregation only. This means in direct aggregation like MEDIAN, there will be more data transmission.

5) A query termination condition. This is to specify the termination of the query, which can be a specific time point or a number denoting the total loop times that data should be collected (i.e. the sensors perform the measurement). Query execution can also be stopped by explicitly disseminating a "stop query" message to the sensor network.

6) A threshold item, to denote the threshold strategy and parameters used in the query execution. It can be either static or dynamic threshold settings, which will be discussed in more detail in Section IV and V respectively.

To give an example of this query in the real world, consider a building with a couple of floors, on each floor there are several rooms, each room is deployed with a number of sensor nodes, and finally each sensor node is equipped with three sensors: temperature sensor, light sensor and sound sensor. Under this setup, one example

query could be that the application wants to know per 30 seconds the average temperature of all the rooms at the third floor for the past 2 minutes, with a sample interval of 10 seconds. In this example, the report interval is 30 seconds, window size is 2 minutes (12 samples), and operation is a global average aggregation function.

To simplify our discussions, without losing any generality, we assume in this paper that the query is aiming at only one data attribute. Therefore, the data produced at a sensor node at one time is a scalar value. We also assume there is no query stop condition, which means the query will keep on executing until an explicit command message is received.

Now consider the possible data flowing in a sensor network, i.e. data that a sensor node might receive and send for the purpose of data gathering. We don't consider query injection and other management messages. We also don't consider relayed data, i.e. data that simply forwarded by a sensor node without any change. There are basically three types of result data: a) local readings, b) local aggregation results, and c) in-network aggregation results. Table 1 shows all the combinations of local and global aggregations with corresponding result types of data produced.

The problem to be addressed in this paper can be described as follows. *For a node there is a streaming data as the results to be sent to the sink, the question is, 1) when to send the data, and 2) to send what exactly, so that the communications (number of messages) are minimum, while the results can be received on real-time with required quality.*

Table 1 Data types produces by aggregation operations.

Local Aggregation	Global Aggregation	Data Type
No	No	a
No	Stepwise	a, c
No	Direct	a
Stepwise / Direct	No	b
Stepwise / Direct	Stepwise	b, c
Stepwise / Direct	Direct	b

III. METHOD OVERVIEW

This section briefly introduces the key idea of our method: using threshold processing in the data collection of continuous query. Threshold processing is commonly used in event based querying. In event based querying, applications set the upper and/or lower boundaries for an attribute. When data value is out of the range of the thresholds, an event occurs. Upon the occurrence, the network may perform some actions like to collect some detailed data from a range of sensor nodes or create an alert message.

Continuous query is different from event based query in the sense that the former is interested in gathering real-time data from the network in a periodical fashion, not like event based query where data collection is triggered by special event. In general, event based query is vastly

preferable, as much less energy is spent than periodical query.

The idea of this paper is thus to explore the possibility of employing this threshold processing into the data gathering of continuous query, so that the energy consumption can be greatly reduced by decreasing the needs of communications.

In continuous query, even though the application is expecting that the network reports real-time data, in many cases however, the application does not really need to obtain the exact readings of each sensor. Instead, it is quite usual that an application just wants to learn the rough situation and trend of the phenomena under monitoring. In other words in terms of granularity, the results to be gathered can be in a coarse-grained style in a low resolution.

On the other hand, in the real world there are a lot of objects whose variance reflects a periodical nature. For example, the temperature of a forest, the sound volume of a road cross, and the light of an outdoor place periodically change over a time of one whole day. This comes to the idea that, to monitor such phenomena, the threshold can be "predicted" and pre-set according to the periodically changed history records, i.e. in a priori fashion.

In considering this fact, it is therefore feasible to collect results in a way like in event based querying, i.e. report results only when the values are out of the range of the thresholds. The difference from event based method is that, in event based query there are only up to two thresholds set as the maximum and minimum values, while in continuous query there must be a set of thresholds set across the whole possible value space.

We presents two categories of threshold processing for continues query, static- and dynamic-threshold processing methods, which are described in more detail in the following two sections.

There can be some other optimization methods employed under such settings, for example packet merge, compression, etc. Those are out of the scope of this paper.

IV. STATIC-THRESHOLD PROCESSING

In static-threshold processing, thresholds for continuous query are defined with the query and are fixed during the entire query execution. This is a simple way, and is often used in the case that there are some prior knowledge about the phenomena under monitoring. We describe the algorithm for the static-threshold processing based continuous querying method in this section.

Setting a static-threshold needs two parameters: *origin value* (V_o) and *threshold interval* (V_i). The choice of the two parameters largely depends on the prior knowledge. In this way the set of thresholds for continuous query is given by $V_k = V_o + k \cdot V_i$, where k is an integer (i.e. can be negative).

Suppose at a sensor node, there is a series of results, $r_1, r_2, \dots, r_i, r_{i+1}, \dots$, to be sent to the node's parent node. The results can be any of the three data types shown in

Table 1. For a specific result data r_i , the data is first rounded off to align with the set of thresholds as follows:

$$r_i' = \text{Round}(r_i) = V_k + V_i/2, \text{ IF } V_k \leq r_i < V_{k+1}.$$

This rounded value will be the one to be sent out from the sensor node. Whether this value will be really sent or not depends on whether it equals to a reference value R_i , i.e.

$$r_i' \text{ is sent, IIF } r_i' \neq R_i.$$

As there are sample interval and report interval clearly specified in the query, every sensor node involved in the data gathering is aware of the exact time of data receiving. Therefore if a node (the sink or a parent node) that is expecting r_i does not receive it in the specific time, it will simply use a value of R_i in their further calculations (i.e. global aggregation).

So far, the problem of 1) when to send, and 2) what to send have both been solved. Now the problem is how to choose a reference value R . Since the energy consumption of a sensor node results mostly from data transmission, and so the number of messages to be sent should be reduced to the biggest extent, therefore it is crucial to choose a suitable value R to be used as the reference that determines whether a data result is sent or not. We propose three methods of reference value calculation as follows.

Method 1 - Most Recent Value (MRV), that is, for any r_i' , $R_i = r_{i-1}'$. In this method, a result data will be sent only if it is different from the previous result data. This is based on the expectation that monitored phenomena won't change too quick. In this case, the sensor node needs to keep a local copy of the previous value.

Method 2 - Recent Average Value (RAV), that is, for any r_i' , $R_i = \text{Round}(\text{AVG}(r_1', r_2', \dots, r_{i-1}'))$. In this method, the reference value is calculated each time a new result data is ready to be sent, and then the average and counter i need to be stored for next calculation. This is to expect that most result data will be around the average of all the results till present. The average operation can be performed over all the history data, or to a set of most recent results.

Method 3 - Most Frequent Value (MFV), that is, the reference value is chosen to be the most frequently appeared rounded value till present. In this method, each node needs to maintain a table $\langle \text{value}, \text{counter} \rangle$, containing all the appeared values (rounded) and their corresponding times of appearance.

In case a big number of possible values exist, the table size will become larger and larger, and so not feasible for sensor nodes with quite limited capacity of memory. In this case, the counting may be terminated when 1) the MFV has been stable, meaning it has not changed for a relatively long run, or 2) a specific number of recent results have been counted, or 3) the table is already too big.

This method is essentially based on the histogram created during the data gathering. All the other methods, static or dynamic, are basically based on the assumption that the result data is uniformly distributed over the value space, i.e. the probability of any value is the same. In

practice, the probability distributions of many phenomena appear to be normal distribution. Therefore, setting the reference value according to the histogram of results may have better performance in terms of number of communications.

Section VII compares all the three methods above.

V. DYNAMIC-THRESHOLD PROCESSING

Static-threshold processing methods, as presented in previous section, are simple and easy to implement. However, they are less flexible in the sense that it does not take the changing conditions of the phenomena into consideration. Moreover, the prior knowledge about the phenomena under monitoring is not always available. We therefore propose three methods by employing dynamic-threshold processing, described as follows and next section. The main difference between the dynamic-threshold method and the static-threshold method lies in the fact that the former sets thresholds dynamically in run-time according to the situation at the moment.

Method 1 - Floating Origin Thresholds (FOT). The static methods proposed in the previous section have one weakness: when a set of readings vary exactly around the fixed points of thresholds, then sensor node will generate a series of unnecessary results to be sent, due to the ping-pong effect. In considering this, we propose an adaptive-average method. Comparing to the fixed-value method in which reported values are defined before the query execution, this is a run-time method in the sense that the thresholds are determined during the execution.

Setting an FOT based threshold needs only one parameters, the *threshold interval* (V_i). This gives the hint that there is no origin point in this method. At any time the node maintains two variables, r_{MIN} and r_{MAX} . Suppose at the next time point a new result data r_i is available, if $r_i \in [r_{\text{MIN}}, r_{\text{MAX}})$, then no data is sent. Otherwise, send the result as r_i , and replace r_{MAX} and r_{MIN} with r_i by: $r_{\text{MAX}} = r_i + V_i/2$ and $r_{\text{MIN}} = r_i - V_i/2$. The process then continues with next iteration.

Method 2 - Adaptive Interval Thresholds (AIT). In FOT method, the thresholds are floating, however the threshold interval is still fixed. Fixed threshold interval has such a problem that, especially when no prior knowledge about the phenomena is available, if the result data largely variant over time, the corresponding number of thresholds will be also very big, and so largely weaken the effect of our methods. On the contrary if the result data changes very small comparing to the threshold interval, it will be hard to reflect the change and thus making the results too coarse. Therefore, the variance of the result data needs to be considered in deciding the threshold interval, and this method represents a compromise between energy consumption (number of communications) and data quality (size of threshold interval).

Setting an AIT based threshold need only one parameter, number of thresholds (N_i). This number basically expresses the accuracy requirements for result reporting. Both the origin point and the threshold interval

are floating. Like in FOT method, at any time a sensor node maintains two reference values, r_{MAX} and r_{MIN} . However in this method, the two threshold values are calculated over a set of the historical data and changing each time a new result data is available, not like in FOT where the two values are simply set according to the latest new results reported.

At any time then, the number k pair of thresholds (T_{k-MIN} , T_{k-MAX}) can be represented by:

$$\left(r_{MIN} + \left(k - \frac{3}{2} \right) \cdot \frac{r_{MAX} - r_{MIN}}{N_t - 1}, \right. \\ \left. r_{MIN} + \left(k - \frac{1}{2} \right) \cdot \frac{r_{MAX} - r_{MIN}}{N_t - 1} \right),$$

where $k = 1, 2, \dots, N_t$. The corresponding reference value for this interval, $R_k = r_{MIN} + (k-1) \cdot \frac{r_{MAX} - r_{MIN}}{N_t}$. Now suppose a new result r_i is available, the operations are as followings.

$$\text{IF } (r_i < r_{MIN} \text{ OR } r_i > r_{MAX},) \\ r_{MAX} = \text{MAX } (r_{MAX}, r_i); \\ r_{MIN} = \text{MIN } (r_{MIN}, r_i); \\ \text{IF } (r_i \in [T_{k-MIN}, T_{k-MAX})) \\ \text{Send } R_k;$$

VI. PERIODICALLY VARIANT THRESHOLDS (PVT)

We describe the algorithm for the coarse-grain data gathering in continuous query processing for periodical phenomena in this section. The algorithm contains two stages, initiation stage and collection stage. In the initiation stage, sensor nodes that are relevant to a specific query collect and store data for the first round for exactly one period of time. In the collection stage, further continuous measurement results are investigated to decide if they are to be sent or not, and if being sent, what data exactly need to be sent.

Obviously, we are mostly aiming at the improvement in terms of energy consumption in the second stage, whilst in the first stage there is also chance to minimize the power consumption during the collection of the large amount of initial data.

A. Initiation stage

To explore the possibility of optimizing data transmission by utilizing the periodical nature of the phenomena under monitoring, initial data need to be collected and stored for the first round (i.e. for one period of time). Here we simply utilize a static-threshold based method. In static-threshold processing, thresholds for continuous query are defined with the query and are fixed during the entire query execution. This is a simple way, and is often used in the case that there are some prior knowledge about the phenomena under monitoring.

Setting a static-threshold needs two parameters: *origin value* (V_o) and *threshold interval* (V_i). The choice of the two parameters largely depends on the prior knowledge. In this way the set of thresholds for continuous query is

given by $V_k = V_o + k \cdot V_i$, where k is an integer (i.e. can be negative).

Suppose the query under study is specified with a period of T and a report interval (i.e. sliding increment, Δt_r), during this period of time T , it is easy to define a series of N time points t_1, t_2, \dots, t_N , so that for $i \in [1, N]$, $t_{i+1} - t_i = \Delta t_r$. Now suppose at a sensor node, there is a series of results, r_i ($i \in [1, N]$) obtained at time t_i , to be sent to the node's parent node. The results can be any of the three data types shown in Table 1. For a specific result data r_i , the data is first rounded off to align with the set of thresholds as follows:

$$r_i' = \text{Round}(r_i) = V_k + V_i/2, \text{ IF } V_k \leq r_i < V_{k+1}.$$

This rounded value will be the one to be sent out from the sensor node. The node first stores each rounded value with the index when it is obtained, then, whether this value will be really sent or not depends on whether it equals to a reference value R_i , i.e.

$$r_i' \text{ is sent, IIF } r_i' \neq R_i.$$

As there are sample interval and report interval clearly specified in the query, every sensor node involved in the data gathering is aware of the exact time of data receiving. Therefore if a node (the sink or a parent node) that is expecting r_i does not receive it in the specific time, it will simply use a value of R_i in their further calculations (i.e. global aggregation).

So far, the problem of 1) when to send, and 2) what to send have both been solved. Now the problem is how to choose a reference value R . Since the energy consumption of a sensor node results mostly from data transmission, and so the number of messages to be sent should be reduced to the biggest extent, therefore it is crucial to choose a suitable value R to be used as the reference that determines whether a data result is sent or not.

Here we employ MRV (Most Recent Value), that is, for any r_i' , $R_i = r_{i-1}'$, $R_N = r_1'$. In this method, a result data will be sent only if it is different from the previous result data. This is based on the expectation that monitored phenomena won't change too fast.

B. Collection stage

After the initiation stage, a preliminary data model is created. In further data collection, we can utilize the rough model to reduce the number of data transmission and so decrease power consumptions.

Formally, suppose the phenomena under monitoring appears to be periodically change over a period of T , and there are a series of history result data r_1, r_2, \dots, r_N obtained (sampled or calculated) over the period T at time point t_1, t_2, \dots, t_N . In the initiation stage, all the rounded values $r_i' = \text{Round}(r_i)$ have been stored as the initial data model. Now suppose a new result r_{N+i} is available, if $\text{Round}(r_{N+i}) = \text{Round}(r_i)$, that is, the new result fits the prediction according to the periodical history, then no data is sent. Otherwise, the local stored $\text{Round}(r_i)$ is replaced by $\text{Round}(r_{N+i})$ to update the data model, and in this case, simply send the updated value $\text{Round}(r_{N+i})$.

This method looks simple so far, but there is a problem when a global aggregation is defined in the query. Suppose a node A has a couple of direct offspring, and is going to perform a global aggregation over the data received from its offspring. During the collection stage, if in a allowed report interval for time t_{N+i} the node A does not receive a result from one of its offspring B , which means node B has a result that is equal to r_i obtained at time t_i , then node A should use node B 's history data r_i in its aggregation. In other words, the aggregation node A must also store all the history data (in one period time) of all its direct offspring. This might be a serious problem, especially when a sensor node has a big number of offspring nodes, the node needs to store a large amount of data. This makes the algorithm not feasible at all, in considering the very limited size of memory of a sensor node in practice.

To solve this problem, we thus improve the algorithm by considering the following three cases for different sorts of aggregations.

Case 1 - MEDIAN aggregation. As analyzed in Section II, MEDIAN is a direct aggregation, so all the data must be sent to the sink before any aggregation could be performed. In other words in the collection stage, no median node perform any aggregation, but instead simply rely the original data. Therefore in this case, local node needs only to maintain the history data of itself.

Case 2 - MAX/MIN aggregations. We simply take MAX aggregation for explanation, as for MIN the idea is exactly the same.

If at time t_{N+i} an aggregation node A receives from all its offspring nodes new results, then node A simply aggregates (i.e. MAX or MIN) a result r_{N+i} by using all the new results. On the contrary, if no offspring node sends any data to node A , then A simply sends to its parent the previous result r_i .

Otherwise if node A receives some updates from part of its offspring, A first calculates a preliminary result r_{N+i} from all the newly received data, and then compares this result with the previous result r_i . If $r_{N+i} > r_i$, then r_{N+i} will be the aggregation result to be sent. If $r_{N+i} < r_i$, then node A broadcasts r_{N+i} to all its offspring nodes. Each node compare this preliminary aggregation result with its local result r . If $r_{N+i} < r$, the offspring node will have to send r to A .

This solution works, but not so good because it requires cluster head to broadcast a middle result to all the offspring nodes. We will show in the experiment section that we actually have better solution than this.

Case 3 - AVERAGE/SUM/COUNT aggregations. We start from the most complex case, AVERAGE aggregation. In this case when new result $r_{N+i} \diamond r_i$, instead of sending r_{N+i} , the node will send a drift value $r_{N+i} - r_i$, as well as a multiplier. Suppose node A has a old average result r_i , and has M offspring nodes among which K of them send drift values r^k , together with the

multipliers m^k . New average aggregation can be calculated by

$$r_{N+i} \in \frac{1}{M} \left(r_i + \sum_{k \in I} r^k \cdot m^k \right).$$

In this case, node A does not have to store any history data of its offspring.

As for the case of SUM or COUNT aggregation, the algorithm is similar: node simply sends a drift value to its parent node where new aggregation value can be calculated by adding the old value with the drift value.

VII. EXPERIMENTS

A. Static vs. dynamic methods.

In this section, we compare the performance of queries by using the proposed methods with queries without any optimization. We also compare the applicability of different threshold setting algorithms to different source of data and aggregations. Four source data are used in the simulation experiments, including A) uniform random data, B) variant random data, C) normal random data, and D) periodical data (period is 20 samples). Each data series contains 100 samples which values change between 0 and 1. Three operations are considered, including 1) no aggregation (No Agg), 2) MAX aggregation, and 3) AVERAGE aggregation. The MAX and AVERAGE aggregations are all performed over the past 10 samples. Sample and report interval are set to be the same. Figure 1 illustrates the source data and the respective operations over them.

Figure 2 to 4 illustrate the simulation results by using static method, MOT method, and AIT method over the four data sources. Figure 5 shows the results by using PVT method over Data D. In the methods, threshold origin is set to 0, threshold interval is set to 0.1, and number of thresholds is set to 10. Finally, the PVT method uses MOT for setting up thresholds in its initial period.

Figure 6 shows the numerical results of energy consumption (in terms of number of communications) for different data series and methods. From these figures, we can clearly see that the proposed methods work quite well. In particular, some observations include that:

1) Our methods achieve better performance for aggregation data type b and c (see Table 1). With respect to different aggregations, the methods work extremely better for MAX than AVERAGE operations. For data type a, i.e. raw readings, only when data source appears to follow a normal distribution do our methods significantly improve the performance, comparing to the situation when no optimization is conducted.

2) Among the three static-threshold processing methods, MRV performs better than method RAV and MFV. MFV is slightly better than MRV only for normal random data (Data C). RAV performs the worst for almost all the situations.

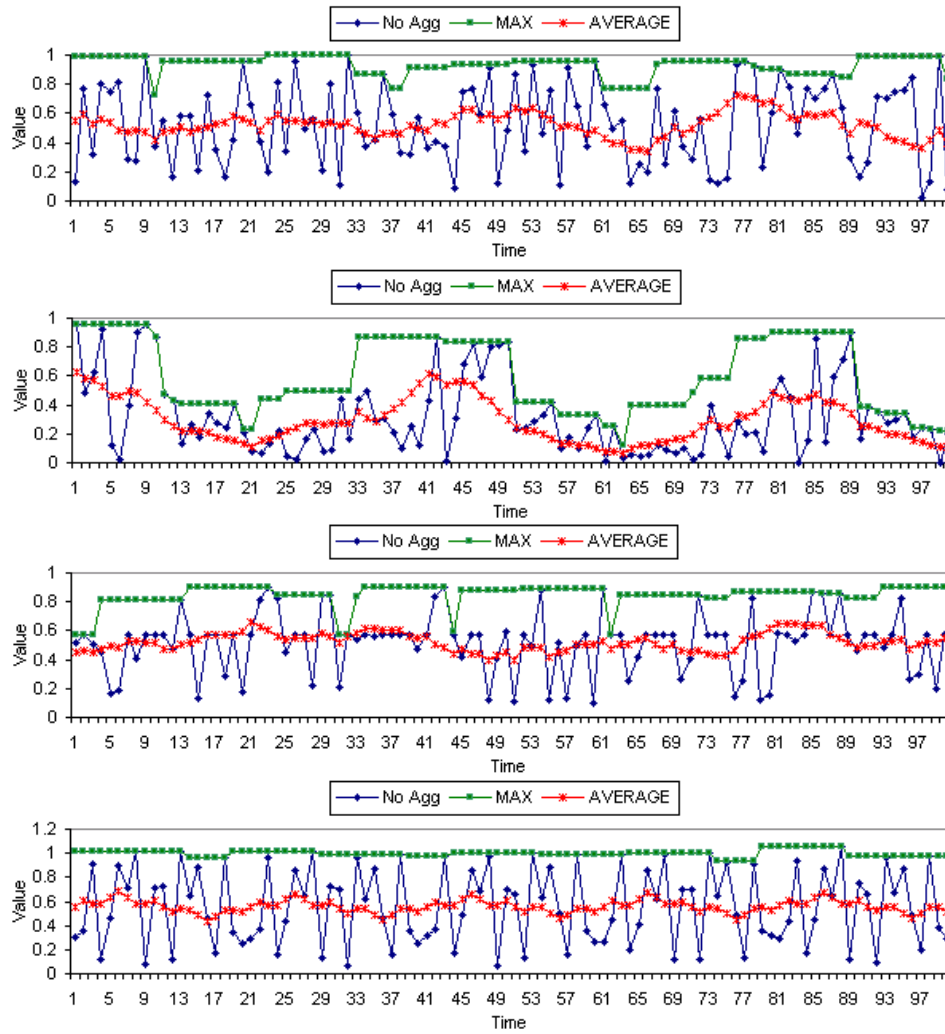
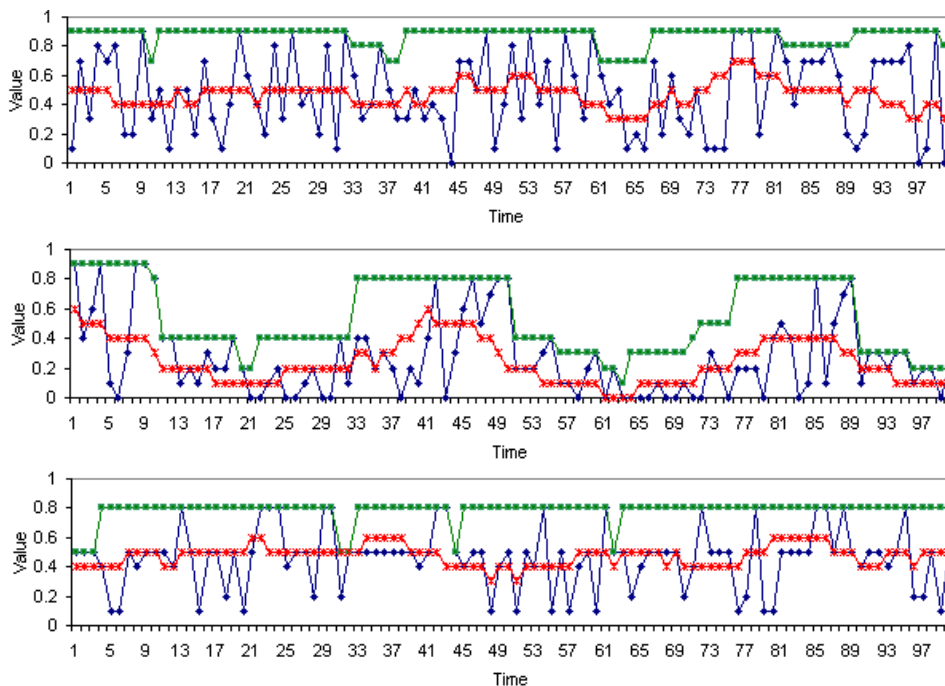


Figure 1. Result data source A to D.



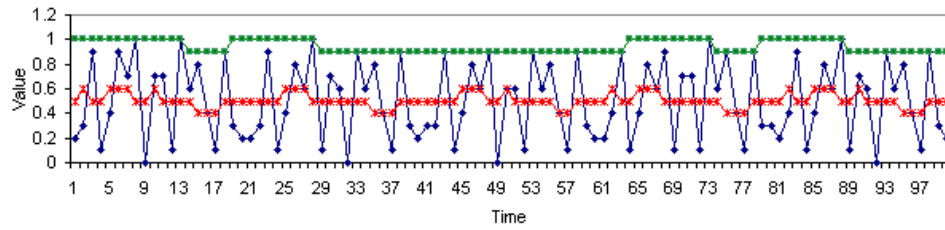


Figure 2. Results by using static methods.

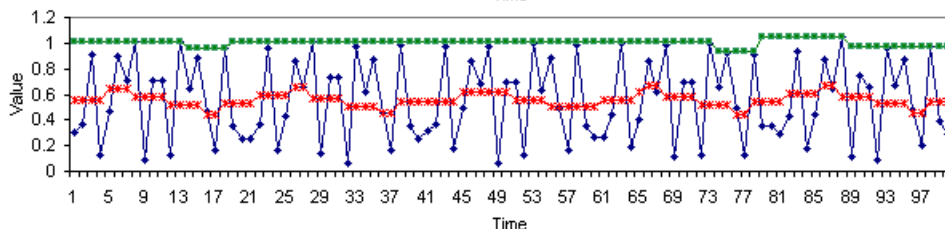
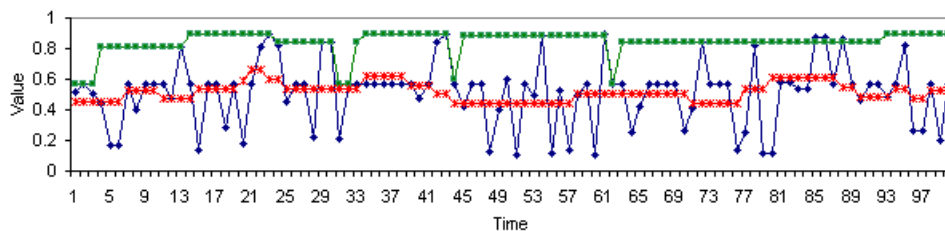
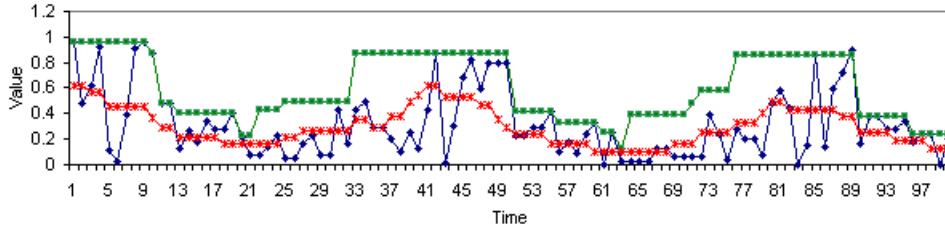
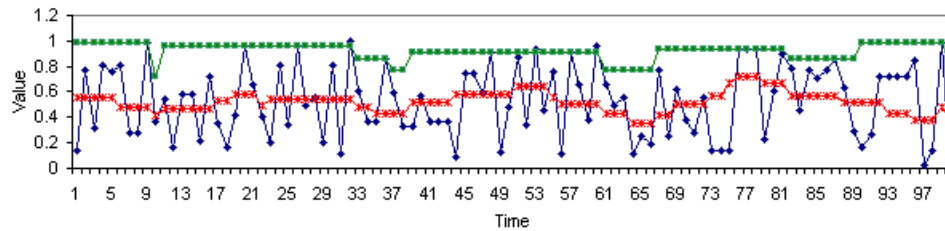
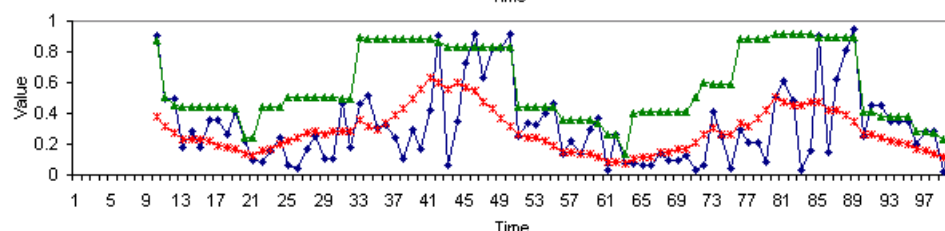
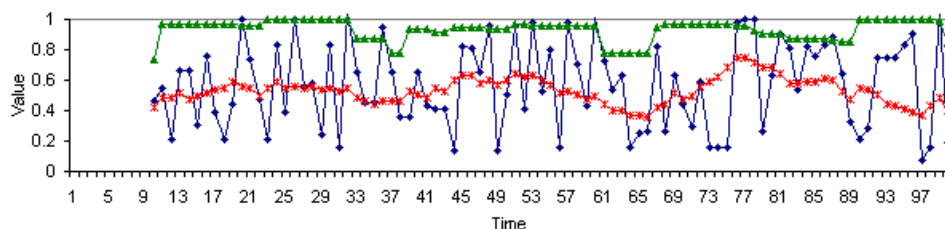


Figure 3. Results by using MOT method.



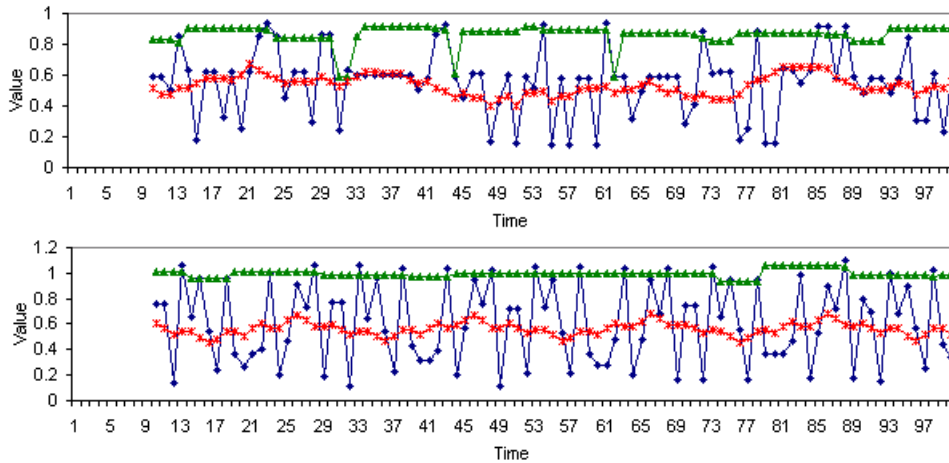


Figure 4. Results by using AIT method.

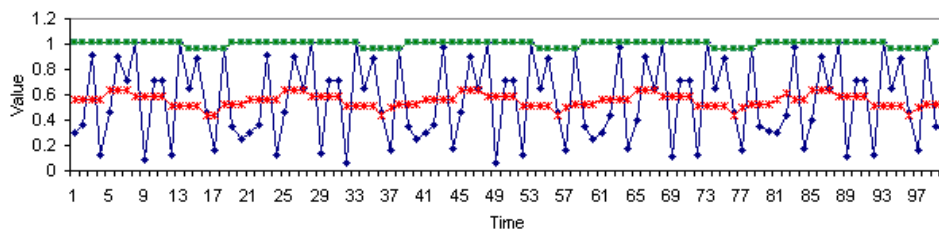


Figure 5. Results of data D by using PVT methods.

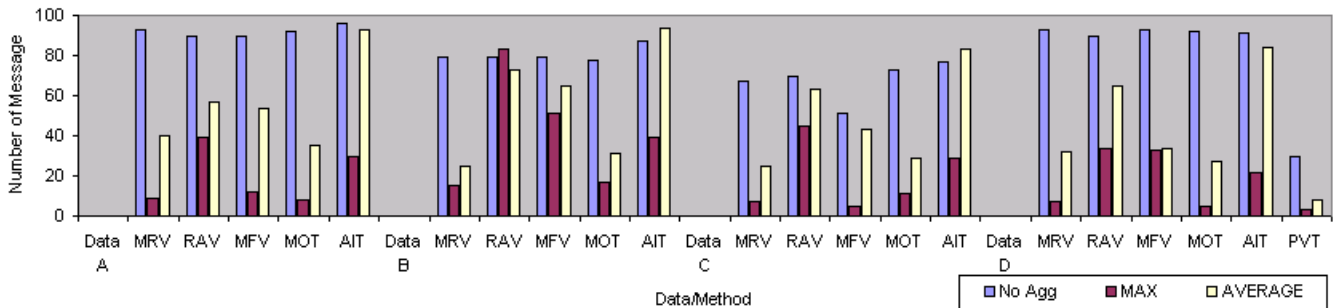


Figure 6. Number of messages for different data series and methods

3) As to dynamic methods, MOT performs much better than AIT, and never worse than MRV (the best of static methods). AIT works poor even for AVERAGE aggregation data. However, AIT can preserve the signal at the best and gives the best restore of the original data, meaning that the results are of the most accurate. This is particularly true when considering variant random data source (Data B).

4) Finally, as shown in Figure 5, PVT returns significantly good results comparing to any other methods, and thus particularly suitable for periodical data.

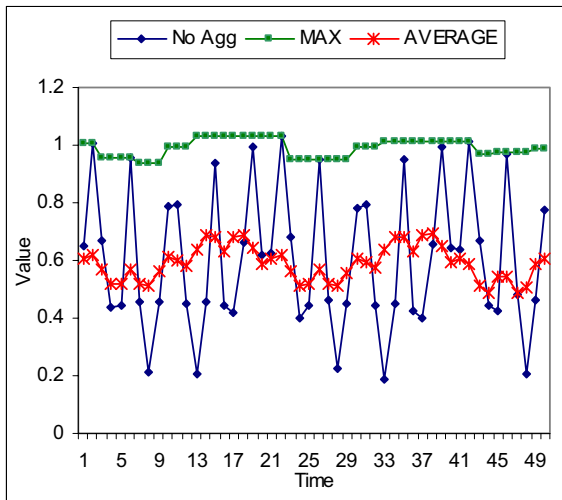
B. Periodically variant thresholds

In this section, we study the performance of queries by using the PVT methods. We also compare the applicability of the PVT algorithms to different source of data and aggregations. Finally, we compare the PVT algorithm with the case of utilizing the MRV method used in the initiation stage.

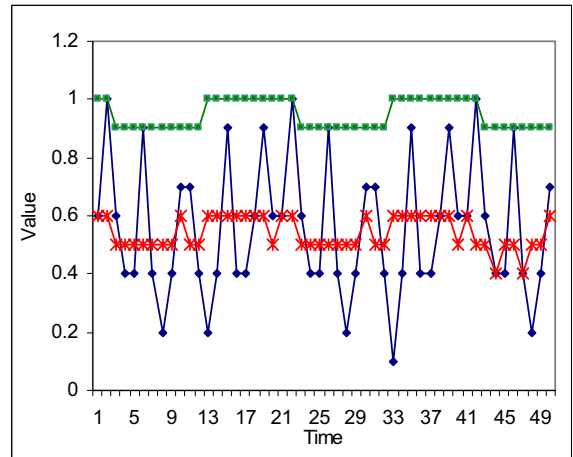
Three sources of periodical data (period is 20 samples) are used in the simulation experiments, including A) variance is on the order of 1/20 of the value range, B) variance is on the order of 1/10, C) variance is on the order of the value range. Each data series contains 50 samples which values change between 0 and 1.

Three operations are considered, including 1) no aggregation (No Agg), 2) MAX aggregation, and 3) AVERAGE aggregation. The MAX and AVERAGE aggregations are all performed over the past 10 samples. Sample and report interval are set to be the same.

Figure 7 to 9 (a) illustrates the source data and the respective operations over them. Figure 7 to 9 (b) illustrate the simulation results. Table 2 shows the numerical results of energy consumption (in terms of number of communications) for different data series and methods. From the figures and the table, we can clearly see that the proposed methods work quite well. In particular, some observations include that:

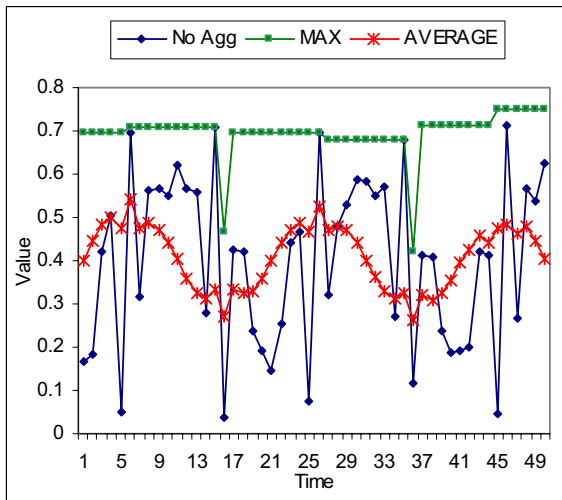


(a)

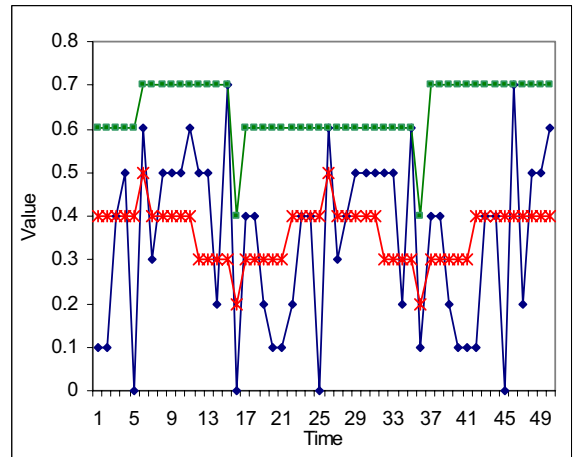


(b)

Figure 7. Data source A and results.

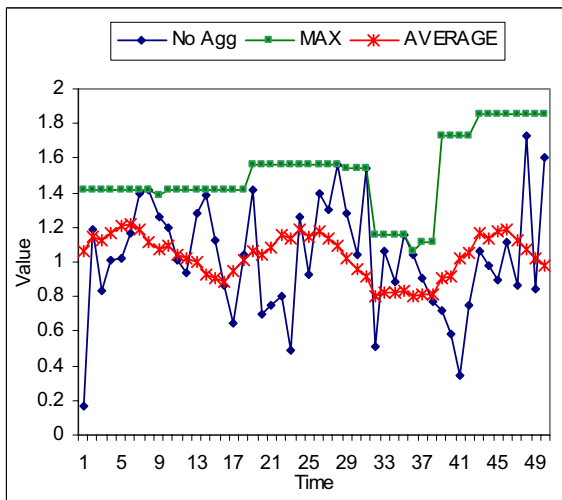


(a)

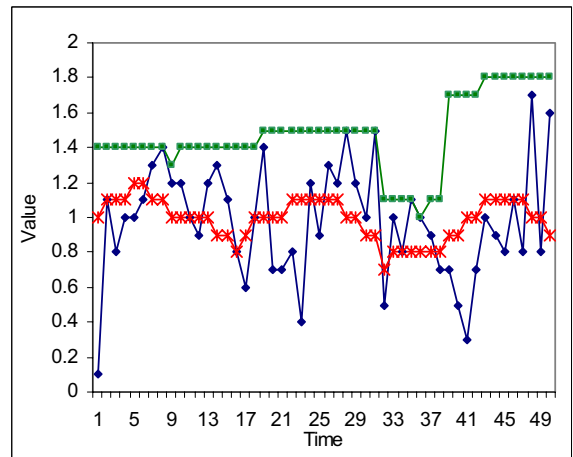


(b)

Figure 8. Data source B and results.



(a)



(b)

Figure 9. Data source C and results.

Table 2 Numerical results.

Data	Method	Number of Msg		
		No Agg	MA X	AVG
A	Proposed	18	3	9
	MRV	41	6	19
B	Proposed	26	28	8
	MRV	34	6	13
C	Proposed	46	34	23
	MRV	46	9	19

1) According to Table 2, in any case, the methods improve the performance than without any optimization performed. Note that when nothing is done, the number of messages to be sent will always be 50 for one single node.

2) The proposed method works much better on data source A than B and C. In other words, when variance is little and so the periodical feature well keeps, the proposed algorithm can significantly improve the performance.

3) The proposed method work fine for no aggregation data and average results, better than MRV method. However when variance becomes larger, the method works much worse to MAX result data than MRV method. Back to the question in the previous section, in this case we can say the MRV works much better than the proposed method if the data are MAX aggregation results.

4) There is no big difference between the results obtained by using MRV method for difference data sources.

VIII. RELATED WORK

There have been a large number of research carried out aiming at query processing for sensor networks, based on a database view of sensor network. In [9] the authors provide a well-understood non-procedural programming interface suitable to data management, allowing the community to realize sensornet applications rapidly, by architecting sensor networks as virtual databases. They also argue that in order to achieve an energy-efficient and usefull implementation, query processing operators should be implemented within the sensor network, and that approximate query results will play a key role.

In [10] the authors define new concept and model of sensor database system, queries dictate with data is extracted from the sensors. Stored data are represented as relations while sensor data are represented as time series. Each long-running query formulated over a sensor database defines a persistent view, which is maintained during a given time interval. The paper also describe the design and implementation of COUGAR sensor database system.

A detailed description of the query processing

mechanism is presented in [14], in which the authors evaluate the design of a query layer that accepts queries in a declarative language that are then optimized to generate efficient query execution plans with in-network processing which can significantly reduce resource requirements.

In [11 and 12] The authors are concerned with query processing in sensor networks and describes in detail the design and implementation of TinyDB, an acquisitional query processing system for sensor networks. Acquisitional issues are those that pertain to where, when, and how often data is physically acquired (sampled) and delivered to query processing operators. The designed system can significantly reduce power consumption over traditional passive systems that assume the a priori existence of data, by focusing on the locations and costs of acquiring data.

Berkely and Cornell have built two prototype sensor network query processors (SNQPs) – TinyDB and Cougar – that run on a variety of sensor platforms. Paper [13] report the architecture and methods, as well as query processing optimization methods of the prototypes.

Some other works include, in [17] the authors proposes analytical models to evaluate the performance of three methods for processing historical spatial-temporal queries in sensor networks. In [18] the authors presents TiNA, an in-network aggregation scheme that maintains the user-specified quality of data requirement while significantly reducing the overall energy consumption. Paper [19] presents the author's progress to date in building TeleTiny, with a particular focus on two components and interfaces between them: server side and sensor side. In [20] the authors consider multi-query optimization for aggregate queries on sensor networks by developing a set of distributed algorithms.

Comparing to these related works, the unique novelty of our proposal lies in the adoption of threshold based methods in continuous aggregation or non-aggregation queries. By taking advantage of the application accuracy constraints, an optimal strategy can be figure out, in which both the accuracy requirements are fulfilled and the performance (with respect to energy consumption) is improved to the best extent. Analysis on the content of the data to be delivered plays very important role in the proposed methods.

IX. CONCLUSION AND FUTURE WORK

This paper presents a set of query optimization methods for continuous data gathering in sensor networks. The method is based on the flexible setting of thresholds and the analysis on the content to be sent over the network. A series of static and dynamic algorithms are proposed for the flexible setting of threshold values. Experiments are conducted to validate the method. Results show that the proposed method can achieve the goal of query optimization.

As shown in the paper, having a priori knowledge about the result data helps very much on choosing the best methods. When the distribution of the sensor

readings is known, e.g. normal distribution rather than uniform distribution, the proposed methods can be further developed to take this knowledge into account to achieve better performance. Moreover, different methods are suitable for data generated from different aggregation operations. This paper only investigates some commonly used aggregations. There are many others that are commonly used in practice. These two issues are the problems for future investigation.

ACKNOWLEDGMENT

The financial support by the Academy of Finland (Project No. 209570) is gratefully acknowledged.

REFERENCES

- [1] Johannes Gehrke and Ling Liu, "Sensor-network applications," *IEEE Internet Computing*, vol. 10, no. 2, 2006, pp. 16-17.
- [2] H. Gharavi and S.P. Kumar, "Special Issue on Sensor Networks and Applications," *Proceedings of the IEEE*, vol. 91, no. 8, Aug. 2003.
- [3] Matthew J. Miller and Nitin H. Vaidya, "A MAC Protocol to Reduce Sensor Network Energy Consumption Using a Wakeup Radio," *IEEE TRANSACTIONS ON MOBILE COMPUTING*, VOL. 4, NO. 3, 228-242, MAY/JUNE 2005.
- [4] Y. Fukushima, H. Harai, S. Arakawa, and M. Murata, "Distributed clustering method for large-scaled wavelength routed networks," *Proc. Workshop on High Performance Switching and Routing*, 416-420, May 2005.
- [5] Lingxuan Hu and David Evans, "Localization for Mobile Sensor Networks," *Proc. Tenth Annual International Conference on Mobile Computing and Networking (MobiCom 2004)*, Philadelphia, 45-57, Sept.-Oct. 2004.
- [6] J.N. Al-Karaki and A.E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, 6-28, Dec. 2004.
- [7] Alan Demers, Johannes Gehrke, Raimohan Rajaraman, Niki Trigoni, and Yong Yao. Energy-Efficient Data Management for Sensor Networks: A Work-In-Progress Report. 2nd IEEE Upstate New York Workshop on Sensor Networks. Syracuse, NY, October 2003.
- [8] Yi Zou and Krishnendu Chakrabarty, "Energy-Aware Target Localization in Wireless Sensor Networks," *Proc. 1st IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, 60-67, Dallas-Fort Worth, Texas, USA, Mar. 2003.
- [9] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, and Scott Shenker, "The sensor network as a database," USC Technical Report No. 02-771, September 2002.
- [10] Philippe Bonnet, J. E. Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. In *Proceedings of the Second International Conference on Mobile Data Management*. Hong Kong, January 2001.
- [11] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Transactions on Database Systems*, vol.30, no.1, 122-173, Mar. 2005.
- [12] S. Madden, M. J. Franklin, J. M. Hellerstien, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings ACM SIGMOD*, pp. 491-502, June 2003, San Diego, CA, USA.
- [13] J. Gehrke and S. Madden, "Query processing in sensor networks," *IEEE Pervasive Computing*, vol. 3, no. 11, pp. 46-55, 2004.
- [14] Y. Yao and J. Gehrke, "Query processing for sensor networks," In *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, California, January 2003.
- [15] W. Rabiner Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocols for Wireless Microsensor Networks," In *Proc. Hawaii International Conference on System Sciences (HICSS '00)*, January 2000.
- [16] W. Heinzelman, A. Sinha, A. Wang, and A. Chandrakasan, "Energy-Scalable Algorithms and Protocols for Wireless Microsensor Networks," In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP '00)*, June 2000.
- [17] Alexandru Coman, Jorg Sander, and Mario Nascimento, "An Analysis of Spatio-Temporal Query Processing in Sensor Networks", In *Proceedings of the 21st International Conference on Data Engineering Workshops*, April 2005, 1190- 1190.
- [18] Jonathan Beaver, Mohamed A. Sharaf, Alexandros Labrinidis, and Panos K. Chrysanthis, "Power-Aware In-Network Query Processing for Sensor data," In *Proceedings of the 2nd Hellenic Data Management Symposium (HDMS'03)*, Athens, Greece, September 2003
- [19] Samuel Madden, "Query Processing for Streaming Sensor Data," Ph.D. Qualifying Exam Proposal, http://db.lcs.mit.edu/madden/html/madden_qual.pdf
- [20] Niki Trigoni, Yong Yao, Alan Demers, Johannes Gehrke, and Rajmohan Rajaraman, "Multi-query Optimization for Sensor Networks," In *Proceedings International Conference on Distributed Computing in Sensor Systems*, June 2005.