A Layered Architecture for Delay Sensitive Sensor Networks

Dan Wang, Yan Long and Funda Ergun
School of Computer Science, Simon Fraser University,
Burnaby BC V5A 1S6, Canada,
{danw, ylong, funda}@cs.sfu.ca

***Abstract*— Sensor networks are powerful tools for performing monitoring and surveillance tasks over large areas. A sensor is a cheap, simple device with low power and limited capabilities. In a sensor network a large number of sensors are deployed to span the whole area to be monitored. Due to the simplicity and the large quantity of the sensors involved, collecting data from a sensor network can be time and energy inefficient.**

In this paper, we investigate making the data gathering task from a sensor network more efficient by using a randomized, layered architecture. The layers in our architecture are constructed in a distributed fashion, with each sensor deciding locally on what layers it will exist. The key property of our technique is that the information is collected from one layer of the architecture containing a small subset of the sensors, resulting in fewer hops and thus smaller data in data aggregation. We provide provably correct results for the delay incurred and the accuracy of the results.

In the context of our new techniques, we also explore ways to speed up the data gathering process even further, such as using history information. In addition, we consider how to optimize the structure of our system so that the energy consumption will be evenly distributed among each sensor, thus extending the overall lifetime of the entire network.

I. Introduction

Sensor networks provide a model in which sensors are deployed in large numbers where traditional wired or wireless networks are not available/appropriate. Their intended uses include terrain monitoring, surveillance, and discovery [11] with applications to geological tasks such as tsunami and earthquake detection, military surveillance, search and rescue operations, building safety surveillance (e.g. for fire detection), and biological systems.

The major difference between sensor networks and traditional networks is that unlike a host computer or a router, a sensor is typically a tightly-constrained device. Sensors not only lack long lifespans due to their limited battery power but also possess little computational power and memory storage [2]. As a result of the limited capabilities of individual sensors, one sensor usually can only collect a small amount of data from its environment and carry out a small number of computations. Therefore, a single sensor is generally expected to work in cooperation with other sensors in the network. As a result of this unique structure, a sensor network is typically data-centric and query-based [8]. When a query is made, the network is expected to distribute the query, gather values from individual sensors, and compute a final value. This final value typically represents key properties of the area where the network is deployed; examples of such values are MAXIMUM, MINIMUM, QUANTILE, AVERAGE, and SUM [18][19] over the individual parameters of the sensors, such as temperature, air or water composition, etc. As an example, consider a sensor network monitoring the average vibration level around a volcano. Each sensor lying in the crater area submits its own value representing the level of activity in a small area around it. Then the data values are relayed through the network; during this they are *aggregated* so that fewer messages need to be sent. Ultimately, the base station obtains the aggregated information about the area being monitored.

In addition to their distributed nature, most sensor networks are highly redundant to compensate for the low reliability of the sensors and the environmental conditions. Since data from a sensor network is the aggregation of the data from individual sensors, the number of sensors in a network has a direct impact on the delay incurred in answering a query. In addition, significant delay is introduced by in-network aggregation [14][16][18], since intermediate parent nodes have to wait for the data values collected from their children before they can aggregate them with their own data.

A long delay is highly undesirable for time-sensitive applications such as critical condition monitoring and security surveillance [5]. As a result, there is increasing interest in research dealing with the delay problem [1][5][24][25].

While most of the techniques for fast data gathering focus on delay-energy efficiencies, they lack provable guarantees for the accuracy of the result. In this paper we focus on a new approach to address the delay and accuracy challenges. We propose a simple distributed architecture which consists of layers, where each layer contains a subset of the sensor nodes. Each sensor randomly *promotes* itself into different layers, where large layers contain a superset of the sensors on smaller layers. The key difference between our layered architecture and hierarchical architectures is that each sensor in our network only represents itself and submits its own data for each query, without the need to act as a "head" of a cluster of sensors. Therefore, when queried, a sensor will always submit fresh data. In this model a query will be made to a particular layer, resulting in an aggregation tree with fewer hops, and thus smaller delay. Unfortunately, the reduction in delay comes with a price tag; since only a subset of the sensors submit their data, the accuracy of the answer to the query is compromised.

In this paper we study the tradeoff between the delay and the accuracy, proving bounds. We perform this study in the context of five key properties of the network, MAX, MIN, QUANTILE, AVERAGE and SUM. We analyze, given a user-defined accuracy level, what layer of the network should be queried for these properties. We show that different queries do show distinct characteristics which affect the delay/accuracy tradeoff. We also show that for certain types of queries such as AVERAGE and SUM, additional statistical information obtained from the history of the environment can help further reduce the number of sensors involved in answering a query. We then investigate the new tradeoffs given the additional information.

The algorithm that we propose for our architecture is fully distributed; there is no need for the sensors to keep information about other sensors. Using the fact that each sensor is independent of others, we show how to balance the power consumption at each node by reconstructing the layered structure periodically. This results in an increase in the life expectancy of the whole network.

A. Related Work

Some pioneering work in addressing the challenges of sensor networks can be found in [8]. A general overview of sensor networks can be found in [2]. For routing techniques and protocols, a survey is in [1].

Sensor networks use data-centric routing instead of address-based routing; SPIN [10] is the first data centric protocol which uses flooding; Directed Diffusion [13] is proposed to select more efficient paths. Several variations and related protocols with similar concepts can be found in [4][7][20].

As an alternative to flat routing, hierarchical architectures have been proposed for sensor networks; in LEACH [11], heads are selected for clusters of sensors; they periodically obtain data from their clusters. When a query is received, a head reports its most recent data value. An enhancement over LEACH can be found in [17]. In [24], energy is focused in a more refined way where a secondary parameter such as node proximity or node degree is included. Clustering techniques are studied in a different fashion in several papers, where [15] focuses on non-homogeneously dispersed nodes and [3] considers spanning tree structures.

In-network data aggregation is a widely used technique in sensor networks. Studies can be found in [18][19][23]. Ordered properties such as QUANTILE are studied in [9]. A recent result in [6] considers power-aware routing and aggregation query processing together, building energy-efficient routing trees explicitly for aggregation queries.

Delay issues in sensor networks are mentioned in [16][18] where aggregation introduces high delay since each intermediate node and the source have to wait for the data values from the leaves of the tree, as confirmed by [25]. In [14], where a modified direct diffusion is proposed, a timer is set up for intermediate nodes to flush data back to the source if the data from their children have not been received within a time threshold. In case of energy-delay tradeoffs, [25] formulates delay-constraint trees. A new protocol is proposed in [5] for delay critical applications where energy consumption is of secondary importance. In these algorithms, all of the sensors in the network are queried, resulting in $\Theta(N)$ processing time, where N denotes the number of sensors in the network, which incurs long delay. Embedding hierarchical architectures into the network where a small set of "head" sensors collect data periodically from their children/clusters and submit the results when queried [11][17][24] provides a very useful abstraction, where the length of the period is crucial for the tradeoff between the freshness of the data and the overhead.

B. Organization of the Paper

In section II, we present the notation that will be used in this paper. Section III is devoted to a detailed description of the proposed system architecture. Section IV contains the theoretical analysis of the tradeoff between the accuracy of query answers and the latency of the system. In section V, we address the energy consumption of our system. In Section VI, we evaluate the performance of our system using simulations. We conclude the paper and discuss possible future work in Section VII.

II. Preliminaries

We assume our network has N sensors denoted by $s_1, s_2, \ldots, s_N$ and deployed uniformly in a square area with side length D. We assume that a base station acts as an interface between the sensor network and the users, receiving queries which follow a Poisson distribution with mean interval length λ.

We embed a layered structure on our network, with L layers, numbered 0, 1, 2, ..., $L-1$. We use $r(l)$ to denote the transmission range used on layer l: during a transmission taking place on layer l, all sensors on layer l communicate using $r(l)$ and can reach one another, in one or multiple hops. Let $e(l)$ be the energy needed to transmit for layer l. The energy spent per sensor for a transmission is $e(l) = r(l)^\alpha$ where $2 \leq \alpha \leq 4$ [22]. Initially, each sensor is at energy level B, which decreases with each transmission. R denotes the maximum transmission range of the sensors.

III. System Architecture

In this section, we describe how the layered structure of our sensor network is constructed and maintained. We also discuss how the queries are distributed to the sensors and how the answers are gathered.

A. Network Construction

We would like to impose a layered structure on our sensor network where each sensor will belong to one or more layers. The properties of this structure are as follows.

a) The *base layer* contains all sensors $s_1, \ldots, s_N$.
b) The layers are numbered 0 through $L-1$, with the base layer labelled 0.
c) The sensors on layer l form a subset of those on layer $l-1$, for $1 \leq l \leq L-1$.
d) The expected number of sensors on each layer drops exponentially with the layer number.

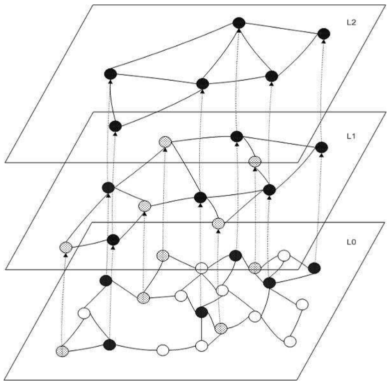

Fig. 1. A Layered Sensor Network; a link is presented whenever the sensor nodes in a certain layer are within transmission range

We now expound on how this structure is constructed. In our scheme, each sensor decides, without requiring any communication with the outside world, to which layer(s) it will belong. We assume that all the sensors have access to a value $0 < p < 1$ (this value may be hardwired into the sensors). Let us consider the decision process that a generic sensor s_i undergoes. All sensors, including s_i, exist in the base layer 0. Inductively, if s_i exists on some layer l, it will, with probability p, *promote* itself to layer $l+1$, which means that s_i will exist on layer $l+1$ *in addition to* all the lower layers $l, l-1, , \ldots, 0$. If on some layer l', s_i makes the decision not to promote itself to layer $l'+1$, s_i stops the randomized procedure and does not exist on any higher layers. If s_i promotes itself to the highest layer $L-1$, it stops the promotion procedure since no sensor is allowed to exist beyond layer $L-1$. Thus, any sensor will exist on layers $0, 1, \ldots, k$ for some $0 \leq k < L-1$. Figure 1 shows the architecture of a sensor network with three layers.

Since our construction does not assume the existence of any mechanism of synchronization, it is possible that some sensor may be late in completing its procedure for promoting itself up the layers. Since the construction scheme works in a distributed fashion, this is not a problem – the late sensor can simply promote itself using probability p and join its related layers in its own time.

Whenever the base station has a query, the query is sent to a specific Layer. Those and only those sensors existing on this layer are expected to take place in the communication. This can be achieved by reserving a small field (of $\log\log N$ bits) in the transmission packet for the layer number. Once l is specified by the base station (the method for which will be explained later), all of the sensors on layer l communicate using transmission range $r(l)$. The transmission range can be determined by the expected distance of two neighboring sensors on layer l, i.e. $r(l) = \frac{D}{\sqrt{N/2^l}}$, and can be enlarged a little further to ensure higher chances of connectivity.

B. Specifying the Structure of the Layers

Note that in the construction of the layers, the sensors do not promote themselves indefinitely; this is because if there are too few sensors on a layer, the inter-sensor distance will exceed the maximum transmission range R. Rather, we "cut off" the top of the layered structure, not allowing more than L layers where $L = \Theta\left(\log \frac{N}{(\frac{D}{R}+1)^2}\right)$.

In what follows, we assume that the promotion probability $p = \frac{1}{2}$; in the Appendix, we analyze the effect of varying p. We also study the effect of different values of p in our simulations.

C. Data Collection and Aggregation

Given a layered sensor network constructed as above, we now focus on how a query is injected into the network and an answer is returned. We simplify the situation by assuming the same as [21] that the base station is a special node where a query will be initiated. Thus the base station acts as an interface between the sensor network and the user.

When the base station has a query to make, it first determines which layer is to be used for this query. Let this layer be l. The base station then broadcasts the query using communication range $r(l)$ for this layer. In this message, the base station specifies the layer number l and the query type (in this paper, we study MAX, MIN, QUANTILE, AVERAGE and SUM). Any sensor on layer l that hears this message will relay information using communication range $r(l)$; those sensors not on layer l will simply ignore this message.

After the query is received by all the sensors on layer l, a routing tree rooted at the base station is formed. Each leaf node then collects its data and sends it to its parent, which then aggregates its own data with the data from its children, relaying it up to its parent. Once the root has the aggregated information, it can obtain the answer to the query.

Note that our schemes are independent of the routing and aggregation algorithms used in the network. Our goal is to specify the layer number l which will reduce the number of sensors, as well as the number of messages, used in responding to a query. Once l is determined, the distribution of the query and the collection of the data can be performed in a number of ways, such as that proposed in [6]. In fact, once the layer to be used for a particular query has been identified, the particular routing/aggregation algorithm to be used is transparent to our algorithm.

IV. Evaluation of the Accuracy and the Latency

In this section we explore how the accuracy of the answers to queries and the latency relate to the layer which is being queried.

In general, we would like to be able to obtain the answers to the queries with as little delay as possible. This delay is a function of the number of sensors whose data are being utilized for a particular query. Thus, the delay is reflected by the layer to which the query is sent. We would also like to get as accurate answers to our queries as possible. When a query utilizes data from all the sensors, the answer is accurate; however, when readings from only a subset of the sensors are used, errors are introduced. We now analyze how these concerns of delay and accuracy relate to the number of sensors queried, and thus to the layer used.

We measure accuracy in terms of the absolute deviation of the computed answer $\hat{a}$ to a query from the exact answer a^*. The accuracy requirement stipulates that this deviation not exceed ϵ in most cases. More precisely, we would like to have that $Pr[|\hat{a} - a^*| \geq \epsilon] \leq \delta$. Here we refer to ϵ as the *accuracy* parameter and δ as the *confidence* parameter.

To explore the relation between the accuracy of the answer to a query and the layer l to which the query has been sent, we recall that the current configuration of the layers has been reached by each sensor locally determining on how many layers it will exist. Due to the randomized nature of this process, the number of sensors on each layer is a random variable. In the next lemma, we investigate which layer must be queried if one would like to have input from at least k sensors.

Lemma 1: Let $l < \log N - \log\left(k + ln\frac{1}{\delta} + \sqrt{ln\frac{1}{\delta}(2k + ln\frac{1}{\delta})}\right)$, where $k \leq$ the expected number of sensors on layer l. Then, the probability that there are fewer than k sensors on layer l is less than δ.

Proof: Define random variable Y_i for $i = 1, \ldots, N$ as follows.

$$Y_i = \begin{cases} 1 & \text{if } s_i \text{ is promoted to layer } l; \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $Y_1, \ldots, Y_N$ are independent. $Pr[Y_i = 1] = 1/2^l$, and $Pr[Y_i = 0] = 1 - 1/2^l$.

On layer l there are $Y = \sum_{i=1}^{N} Y_i$ sensors. $Pr[Y < k] = Pr[Y < \frac{k}{E[Y]}E[Y]] < e^{-(1-\frac{k}{E[Y]})^2 E[Y]/2}$ by Chernoff's inequality.

Since $E[Y] = N/2^l$, to have $e^{-(1-\frac{k}{E[Y]})^2 E[Y]/2} < \delta$, we must have $l < \log N - \log\left(k + ln\frac{1}{\delta} + \sqrt{ln\frac{1}{\delta}(2k + ln\frac{1}{\delta})}\right)$ ∎

In what follows, we analyze the accuracy and the latency in the context of certain types of queries.

A. MAX and MIN Queries

In general, exact answers to maximum or minimum queries cannot be obtained unless all sensors in the network contribute to the answer, since any missed sensor might contain an arbitrarily high or low data value. The following theorem is immediate.

Theorem 2: The queries for MAX and MIN must be sent to the base layer to avoid arbitrarily high error.

B. QUANTILE Queries

Due to similar reasons as the MAX and MIN queries, we cannot obtain an exact quantile by querying a proper subset of the sensors in the network. Thus, we introduce an approximate notion of quantile.

Definition 1: The *ϕ-quantile* ($\phi \in (0, 1]$) of an ordered sequence S is the element whose rank in S is $\phi|S|$.

Definition 2: An element of an ordered sequence S is the *ϵ-approximation ϕ-quantile* of S if its rank in S is between $(\phi - \epsilon)|S|$ and $(\phi + \epsilon)|S|$.

The following lemma shows that a large enough subset of S has similar quantiles to S.

Lemma 3: Let $Q \subseteq S$ be picked at random from the set of subsets of size k of S. Given error bound ϵ and confidence parameter δ, if $k \geq \frac{ln\frac{2}{\delta}}{2\epsilon^2}$, with probability at least $1 - \delta$, the *ϕ-quantile* of Q is an *ϵ-approximation ϕ-quantile* of S.

Proof: The element with rank $\phi|Q|$ in Q [1] does not have rank within $(\phi \pm \epsilon)|S|$ in S if and only if one of the following holds: a) More than $\phi|Q|$ elements in Q have rank less than $(\phi - \epsilon)|S|$ in S, or b) more than $(1 - \phi)|Q|$ elements in Q have rank greater than $(\phi + \epsilon)|S|$ in S.

Since $|Q| = k$, the distribution of elements in Q is identical to the distribution where k elements are picked uniformly at random without replacement from S. This is due to the fact that any element of S is as likely to be included in Q as any other element in either scheme, and both schemes include k elements in Q.

Since the two distributions mentioned above are identical, we can think of the construction of Q as k random draws without replacement from a *0-1 box* that contains $|S|$ items, of which those with rank less than $(\phi - \epsilon)|S|$ are labelled "1" and the rest are labelled "0". For $i = 1, \ldots, k$, let X_i be the random variable for the label of the ith element in Q. Then $X = \sum_{i=1}^{k} X_i$ is the number of elements in Q that have rank less than $(\phi - \epsilon)|S|$ in S. Clearly, $E[X] = (\phi - \epsilon)k$. Hence $Pr[X \geq \phi k] = Pr[X - E[X] \geq \phi k - (\phi - \epsilon)k] = Pr[X - E[X] \geq \epsilon k] = Pr[\frac{X}{k} - E[\frac{X}{k}] \geq \epsilon]$. This is at most $e^{-2\epsilon^2 k}$, by Hoeffding's Inequality. Note that Hoeffding's Inequality applies to random samples chosen without replacement from a finite population, as shown in Section 6 of Hoeffding's original paper [12], without the need for independence of the samples.

Similarly, it can be shown that the probability that more than $(1 - \phi)|Q|$ elements in Q have rank greater than $(\phi + \epsilon)|S|$ in S is also at most $e^{-2\epsilon^2 k}$. Setting $2e^{-2\epsilon^2 k} \leq \delta$, we have $k \geq \frac{ln\frac{2}{\delta}}{2\epsilon^2}$. ∎

We now show which layer we must use for given error and confidence bounds.

Theorem 4: If a ϕ-quantile query is sent to layer $l < \log N - \log\left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, then the answer will be the ϵ-approximation ϕ-quantile of the whole network with probability greater than $(1 - \delta)$.

Proof: By Lemma 1, the probability that layer $l < \log N - \log\left(k + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2k + ln\frac{2}{\delta})}\right)$ has fewer than k sensors is less than $\frac{\delta}{2}$. By Lemma 3, if the number of sensor nodes on layer l is at least $\frac{ln2(\frac{2}{\delta})}{2\epsilon^2} = \frac{ln\frac{4}{\delta}}{2\epsilon^2}$, the probability that the ϕ-quantile on layer l is ϵ-approximation ϕ-quantile of the sensor network is at least $1 - \frac{\delta}{2}$. Hence the answer returned by layer $l < \log N - \log\left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$ is ϵ-approximation ϕ-quantile of the sensor network with probability greater than $(1 - \delta)$. ∎

C. AVERAGE and SUM Queries

AVERAGE queries and SUM queries are correlated queries where the AVERAGE is just SUM$/N$. Since we know the

[1] Wherever rank in a set is mentioned, it should be understood that this rank is over a sequence obtained by sorting the elements of the set.

number of the sensors in advance, we just analyze the AVERAGE queries in this section and do not explicitly explain the SUM queries.

We now consider approximating the average data value over the whole sensor network by querying a particular layer. The below lemma indicates that the expectation of the average data value of an arbitrary layer is the same as the average of the base layer, which is the exact average of the sensor network.

Lemma 5: Let $a_1, a_2, \ldots, a_N$ be the data values collected by the nodes $s_1, s_2, \ldots, s_N$ of the sensor network. Let k be the number of sensors on layer l. Let $X_1, X_2, \ldots, X_k$ be the random variables describing the k data values on layer l. Let $\overline{X} = \frac{1}{k}\sum_{i=1}^{k} X_i$. Then $E[\overline{X}] = \frac{1}{N}\sum_{i=1}^{N} a_i$.

Proof: Since each sensor independently promotes itself to layer l with the same probability, $Pr[X_i = a_1] = Pr[X_i = a_2] = \ldots = Pr[X_i = a_N] = \frac{1}{N}$, for $i = 1, 2, \cdots, k$. Then $E[X_i] = \frac{1}{N}(a_1 + a_2 + \cdots + a_N)$. Hence $E[\overline{X}] = E[\frac{1}{k}\sum_{i=1}^{k} X_i] = \frac{1}{k}\sum_{i=1}^{k} E[X_i] = \frac{1}{k}\frac{k}{N}(a_1 + a_2 + \cdots + a_N) = \frac{1}{N}\sum_{i=1}^{N} a_i$. ■

We thus propose that the average returned by the queried layer be output as the average of the whole network. The next theorem shows that, given the appropriate layer, this constitutes an ϵ-approximation to the actual average with probability greater than $1 - \delta$.

Theorem 6: Let the data value at each sensor come from the interval $[a, b]$, and let l be such that $l < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$. Then the probability that the average of the data values on layer l deviates from the exact average by more than ϵ is less than δ.

Proof: Let k be the number of sensors on layer l. As we have explained in Lemma 3, these k sensors can be considered to be random samples without replacement from all of the N sensors. Let $X_1, X_2, \ldots, X_k$ be the random variables describing the k sensor values on layer l, as in Lemma 5. Then $a \leq X_i \leq b$ for $i = 1, 2, \cdots, k$. Let $\overline{X} = \frac{1}{k}\sum_{i=1}^{k} X_i$. By Lemma 5, $E[\overline{X}]$ is the exact average of the sensor network. For any $\epsilon > 0$, $Pr[|\overline{X} - E[\overline{X}]| \geq \epsilon] \leq 2e^{\frac{-2k\epsilon^2}{(b-a)^2}}$, by Hoeffding's Inequality. Setting $2e^{\frac{-2k\epsilon^2}{(b-a)^2}} \leq \delta/2$, we have $k \geq \frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2}$. By Lemma 1, the probability that layer $l < \log N - \log\left(k + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2k + ln\frac{2}{\delta})}\right)$ has fewer than k sensors is less than $\frac{\delta}{2}$. Thus, if we send an AVERAGE Query to layer $l < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})}\right)$, the probability that the estimated average deviates from the exact average more than ϵ is less than $\frac{\delta}{2} + \frac{\delta}{2} = \delta$. ■

1) Utilizing Statistical Information about the Behavior of Data: If we have access to additional information regarding the characteristics of the objects that the sensor network is monitoring, we can reduce the latency even further. In what follows, we show that the knowledge that the change in data values over time respects a certain distribution (such as the normal distribution) can be used to improve the quality of our estimates.

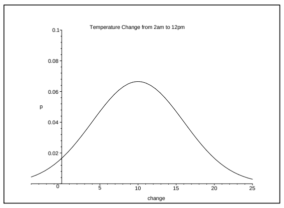

Fig. 2. Temperature changes from 2am to 12pm

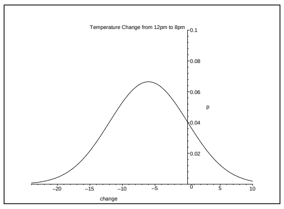

Fig. 3. Temperature changes from 12pm to 8pm

Assume the change of the data value in one time unit for each single sensor follows a normal distribution with mean μ. For instance, we might know that the temperature is likely to rise around 10 degrees from 2 am to 12 pm, and fall around 6 degrees from 12 pm to 8 pm. Small variations might happen but substantial changes are less likely. (See Figures 2 and 3). The change in the average value also follows a normal distribution since the sum of normal distributions is still a normal distribution with mean and variance equal to the sum of the individual means and variances.

To make use of the statistical information regarding the change in the value, we adopt a history-based approach, where we assume that we know the distribution of the change of the environment to be a normal distribution with mean μ.

The intuition behind our strategy is as follows. First we obtain an initial estimate avg of the average data value in the network, which, by our computations above, is likely to be close to the true average. After one unit of time, the true average is likely to have changed by some value close to μ. Thus, $avg + \mu$ is likely to be a good estimate for the average for that point in time. However, errors have been introduced into our estimate. One cause for possible error is the fact that only a subset of the sensors have been queried. The other contribution to the error comes from our inability to know the exact change in the data value; we only know from the normal distribution that the change is "likely" to be "around" μ. Since the quantity of the error, as well as its likelihood increases with each step of this procedure, we need to make sure that our error and confidence bounds remain at acceptable levels.

To ensure low error, we adopt a multi-stage approach to our estimation of the average. In the first stage, which we call *Query Average*, we query a relatively large subset of the sensors – more precisely, we query a low enough layer to obtain an error of $\epsilon_1 < \epsilon$ with confidence level $\delta_1 < \delta$. This high guarantee will leave some room for extra error to be incurred in the following stages.

In the following stage (after one time unit has elapsed), which we call *Test Average*, and subsequent ones, we will query higher layers, thus involving a smaller number of sensors, to see whether the expected change pattern is followed. The result of doing this is that either (a) we will boost the confidence to an acceptable level or (b) we will observe an "anomaly", that is, a deviation from expected behavior, which we will attempt to resolve by querying a lower layer with a larger number of sensors. In case of (a), we will have obtained a fast and acceptable answer by querying only a very small number of sensors. Case (b) on the other hand is, by definition of the normal distribution, an anomaly that will not happen often. In the unlikely event of an "accident" near one of the nodes, in the form of an atypical value, our system will experience a longer query time for the sake of accuracy. In the long run, we will see more "expected" cases and will observe a lower average query time.

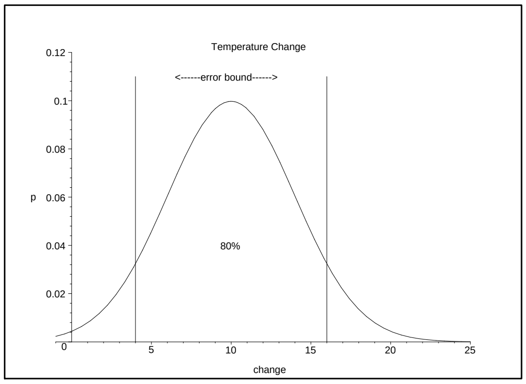

Fig. 4. The possible change for Temperature after a time unit follows a normal distribution with $\mu = 10$ and $\sigma = 4$. To ensure the ultimate error bound of $\epsilon = 8$, the error bound $\epsilon_n = 6$

Before we go into the specifics of the algorithm, we present an example (Figure 4). Suppose $\epsilon = 8$ and $\delta = 20\% (i.e., 0.2)$. In the first stage, we see that we get the average data value $avg_1 = 60$°F by using error bound, say $\epsilon_1 = 2$ and a confidence level, say 90%, (i.e. error probability $\delta_1 = 10\%$). After 10 hours, we expect that the temperature changes to $avg_1 + \mu = 70$°F. However, to ensure an error within the user specified bound of 8, the error (one-sided confident interval) for the normal distribution must be less than 6 (i.e. $\epsilon_n = 6$), as shown in Figure 4, with probability 80%. Therefore, after one time unit, the confidence level is $80\% \times 90\% = 72\%$, i.e., the error probability is larger than the 20% specified as acceptable. To boost the confidence level to 80%, we need to query a few more sensors with an error bound of $\epsilon_2 = 8$ and confidence level of only $\delta_2 = 28\%$. If the returned value of Test Average is 70, we will return this value. Otherwise, if the returned value falls outside of 70 ± 8, this indicates that an anomaly might be present, [2] in which case we perform Query Average to determine the new temperature value. If the returned value falls within 70 ± 8, to ensure the error bound, we perform Test Average with a more stringent error bound $\hat{\epsilon}$ until an anomaly is found or the new average value is confirmed.

Below we explain our algorithm in higher detail and analyze its properties mathematically. Figure 5 shows Algorithm Query Average. It takes as input the error and confidence parameters ϵ, δ. We assume that $Query(l)$ returns the average data value for sensors on layer l.

Our next algorithm (Figure 7) shows how to perform Test Average given Query Average. It takes as input the error and confidence parameters ϵ, δ, as well as the mean μ and standard deviation σ of the distribution of the change of the data value. It also takes avg_1 which is the average obtained from Query Average and the round number i. Other parameters used in the algorithm are listed in the Figure 6.

[2]Here we use the word anomaly to indicate a situation whose likelihood is small according to the given normal distribution.

Algorithm QueryAvg (ϵ, δ)
Select $\epsilon_1 < \epsilon$ and $\delta_1 < \delta$.
$l_1 = \log N -$
$$\log\left(\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1} + \sqrt{ln\frac{2}{\delta_1}(2\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1})}\right)$$
$avg_1 = Query(l_1)$.
return avg_1

Fig. 5. Algorithm Query Average

q_i	confidence level of the normal distribution
ϵ_n	one-sided confidence interval of the normal distribution
s	factor to make the confidence interval stringent

Fig. 6. parameter table

Algorithm TestAvg $(i, \epsilon, \delta, \mu, \sigma, avg_1)$
1 $\epsilon_n = \epsilon - \epsilon_1$.
2 Calculate $q_i = Pr(\mu - \epsilon_n < X < \mu + \epsilon_n)$ by Normal Distribution.
3 **if** $1 - q_i \times (1 - \delta_1) < \delta$,
4 $avg_i = avg_1 + \mu$, **return** avg_i
5 **else**
6 **repeat**
7 $s = 0$, $\epsilon_i = \epsilon - s$, $\delta_i = \frac{\delta}{1 - q_i \times (1-\delta_1)}$
8 $l_i = \log N -$
$$\log\left(\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i} + \sqrt{ln\frac{2}{\delta_i}(2\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i})}\right)$$
9 **if** $(Query(l_i) \le avg_1 + \mu + s)$ **and** $(Query(l_i) \ge avg_1 + \mu - s)$
10 **then**
11 $avg_i = avg_1 + \mu$, **return** avg_i
12 **else**
13 increase s
14 **if** $s \ge \epsilon$, Goto QueryAvg

Fig. 7. Algorithm Test Average

In Line 2 of Algorithm Test Average, we calculate the probability that the change will fall within interval ϵ_n. In Lines 1-5, if the Query Average has already guaranteed the error probability, we do not perform any further queries. This might occur when the number of sensors queried in Query Average is large enough. Line 14 displays the threshold where we should perform the query again.

Theorem 7: Assume the data value collected by each sensor is bounded by $[a, b]$ and the change in the average of the values at all sensors follows a normal distribution with mean μ and standard deviation σ. The probability that algorithm QueryAvg and TestAvg will deviate from the exact average by more than ϵ is less than δ.

Proof: Let us first consider QueryAvg. Choose any ϵ_1, δ_1 such that $\epsilon_1 < \epsilon$ and $\delta_1 < \delta$. By Theorem 6, we can obtain the desired accuracy by sending queries to any layer l_1 where $l_1 < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1} + \sqrt{ln\frac{2}{\delta_1}(2\frac{(b-a)^2 ln\frac{4}{\delta_1}}{2\epsilon_1^2} + ln\frac{2}{\delta_1})}\right)$.

For TestAvg, let $\overline{Y}_i$ denote the average value over all the sensors at round i. Define $\Delta_i = \overline{Y}_i - \overline{Y}_1$. We know *a priori* that the probability distribution for each Δ_i is a normal distribution with mean μ_i and variance σ_i^2.

In round i, $\epsilon_n = \epsilon - \epsilon_1$ is the confidence interval for normal distribution, hence q_i is probability that the change in the value of the average will not exceed ϵ_n.

Therefore with probability $1 - q_i \times (1-\delta_1)$, we can guarantee an error bound of $\epsilon_1 + \epsilon_n < \epsilon$. If $1 - q_i \times (1-\delta_1) < \delta$ then our query satisfies both bounds ϵ and δ, and we can compute the value to be returned from the value in the previous round and the expected change. Otherwise, we choose $\delta_i = \frac{\delta}{1-q_i\times(1-\delta_1)}$ which ensures that the confidence error δ will be bounded in the ith round. For error bound ϵ_i in the ith round, since we don't know the returned value, we can use all ϵ_i from ϵ to 0 as long as the returned value $avg_i \pm (\epsilon_i)$ will be bounded by $(avg_1 + \mu) \pm \epsilon$. If that happens, the change is confirmed. Otherwise, to bound ϵ and δ, a new QueryAvg must be performed. In our algorithm, we reduce ϵ_i iteratively from ϵ to 0, and use ϵ_i and δ_i to query layer $l_i < \log N - \log\left(\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i} + \sqrt{ln\frac{2}{\delta_i}(2\frac{(b-a)^2 ln\frac{4}{\delta_i}}{2\epsilon_i^2} + ln\frac{2}{\delta_i})}\right)$ so that the number of sensors in TestAvg will increase little by little and stop as early as possible. ■

V. Energy Consumption

It can be readily observed that in our system higher layer sensors will be transmitting at longer ranges than their lower layer counterparts. Given that any high layer sensor is also present in all the lower layers, if nothing is done to balance out the energy consumption, the higher layer sensors may get depleted much faster than the lower layer ones. To balance out the energy consumption, our system reconstructs the layered network periodically by deciding each layer from scratch, so that the top layer sensors change over time. An appropriate timing scheme for the reconstruction will lead to relatively uniform energy consumption across the sensors in the network. Note that the frequency of reconstructions has no expected effect on accuracy, since we are as likely to be stuck with a "good" sample of sensors (in which case reconstruction is likely to give us a worse sample) as with a "bad" one. Given the above and the overhead of building a new aggregation tree for each new construction, we are interested in infrequently repeating this procedure for making the energy consumption more even across sensors.

Let the lifetime of the network be the time between its initial construction and the first time that a sensor runs out of power [24]. We investigate the relationship between the timing of the reconstructions and the expected lifetime of our system in our simulations. In this section, we analyze our system assuming that each sensor has sufficient power to let it undergo several reconstructions, and that we run reconstructions enough times. Ideally, we have a totally symmetric scenario where the service that each sensor has performed on each layer is identical across sensors. Since the layers are chosen in a randomized fashion, given a large enough number of reconstructions what one expects to really see is most sensors having served on most layers.

The energy spent by each sensor for a query directly depends on the distance between the sensor and its neighbors. Recall that since there are an expected $(N/2^l)$ nodes on layer l, the transmission range is set to be $r(l) = \frac{D}{\sqrt{N/2^l}}$. Therefore, the energy spent by each sensor for each query on layer l is $e(l) = \left(\frac{D}{\sqrt{N/2^l}}\right)^\alpha$, which is what we will use below to estimate the overall system lifetime.

A. Overall Lifetime of the System

In this section, we assume that the queries are uniformly distributed across different layers due to the error bounds and confidence levels coming independently from the users.

We now present a theorem which estimates the expected lifetime of our system depending on the network parameters.

Theorem 8: In a setting where each level is equally likely to be queried, the expected lifetime of our system is $E(t) = \frac{BL(\sqrt{N})^\alpha(1-(\sqrt{2})^{\alpha-2})}{\lambda D^\alpha(1-(\sqrt{2})^{L(\alpha-2)})}$

Proof: We assume that each layer has the same probability $\frac{1}{L}$ of being queried. The probability that a sensor exists on layer l is $\frac{1}{2^l}$, therefore, the energy consumption for this sensor is $\sum_{l=0}^{L-1}\frac{1}{2^l}e(l)\frac{1}{L}$. Let the life expectancy of this sensor be t. Recall that B is the battery power, λ is the incoming query interval following (assumed) Poisson distribution, and $e(l) = \left(\frac{D}{\sqrt{N/2^l}}\right)^\alpha$. We have $\sum_{l=0}^{L-1}\frac{1}{2^l}e(l)\frac{1}{L}\lambda t = B$. Therefore, the expected lifetime of the system is $E(t) = \frac{BL(\sqrt{N})^\alpha(1-(\sqrt{2})^{\alpha-2})}{\lambda D^\alpha(1-(\sqrt{2})^{L(\alpha-2)})}$ ■

VI. Simulations

We use simulations to test the performance of our system, as well as to observe the effects of the parameters of the algorithm and the re-election time on the performance.

A. The Relationship Between Layer and Accuracy

We first evaluate the relationship between the layer answering a query and parameters relating to the quality of the answer to the query. In our simulations we set the promotion probability $p = 1/2$ (we explore the effects of varying p later) and $N = 10000$, and focus on QUANTILE and AVERAGE queries.

1) QUANTILE Queries: We first present a general picture of the relationship among ϵ and δ in Figure 8. It can be seen clearly that, as ϵ and δ increase, the layer that the query should be sent to also increases, as confirmed by our computations. As expected, we also observe this trend for AVERAGE.

Figures 9 and 10 show the relationship between δ and the layer number and the relationship between ϵ and the layer number. Here it can be observed that even though the layer number monotonically increases with both parameters, ϵ has more impact on it than δ. This is because the confidence parameter δ can easily be improved using standard

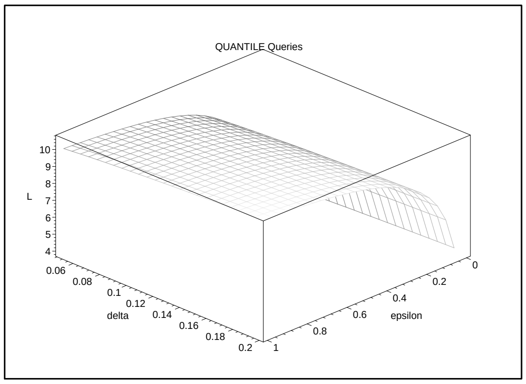

Fig. 8. QUANTILE Queries, the impact of ϵ, δ with Layer number L, N = 10000 and $p = 0.5$

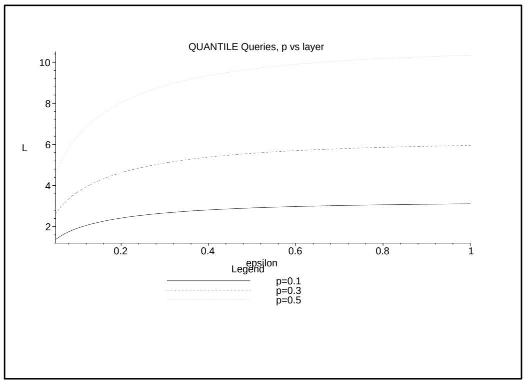

Fig. 11. QUANTILE Queries, the effect of promotion probability p

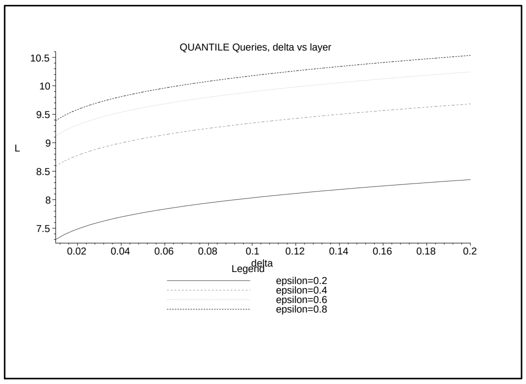

Fig. 9. QUANTILE Queries, confident parameter δ vs. Layer L

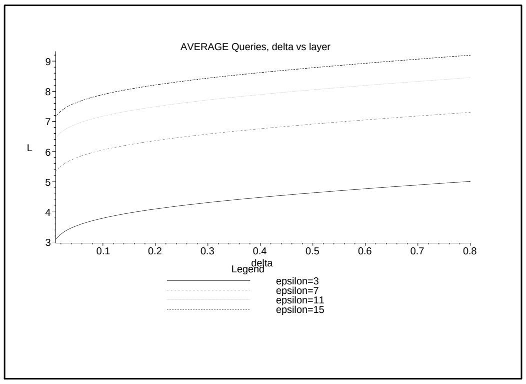

Fig. 12. AVERAGE Queries, confident parameter δ vs. Layer L

boosting techniques from probability theory and randomized algorithms. In fact, repeating the algorithm $O(\log k)$ times and returning the median answer will improve δ to δ/k, since the probability of getting an incorrect answer $(\log k)/2$ times is at most $\delta^{\log k}$, which is $O(\delta/k)$. On the other hand, to reduce ϵ by a constant factor k, $O(k)$ repetitions of the experiment are needed.

Figure 11 shows the relationship between p and the layer number. As p increases, the variation in the layer number for the same query is more obvious. This is because there are fewer layers for smaller p and the choice of layer is more coarse-grained than for larger p.

2) AVERAGE Queries: Figures 12 and 13 show the relationship between δ and the layer number and the relationship between ϵ and the layer number. We observe here the same effect as with the QUANTILE queries, i.e., ϵ has a larger impact than δ, for the same reason. This gives us hints for building test queries as we have additional statistical information.

To investigate the question of how to choose the parameters introduced in our algorithms, QueryAvg and TestAvg, we fix the following parameters and vary the others. The upper bound and lower bound of the data value are $a = 20$ and $b = 100$. p and N are set to be the same as described in the above section. We set the user-defined error bound and confidence parameter to be $\epsilon = 8.1$ and $\delta = 25\%$ respectively. The mean and standard deviation for the normal distribution are set to $\mu = 10$ and $\sigma = 2$.

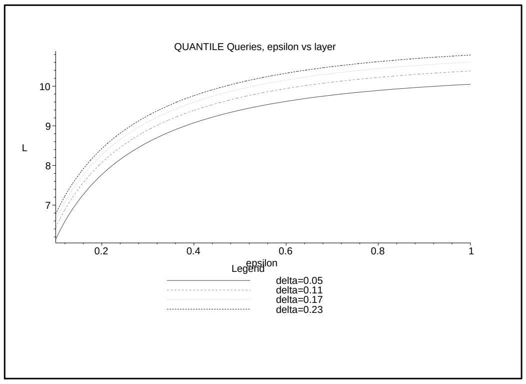

Fig. 10. QUANTILE Queries, error bound ϵ vs. Layer L

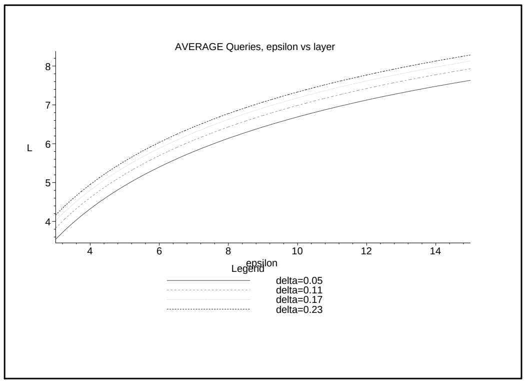

Fig. 13. AVERAGE Queries, error bound ϵ vs. Layer L

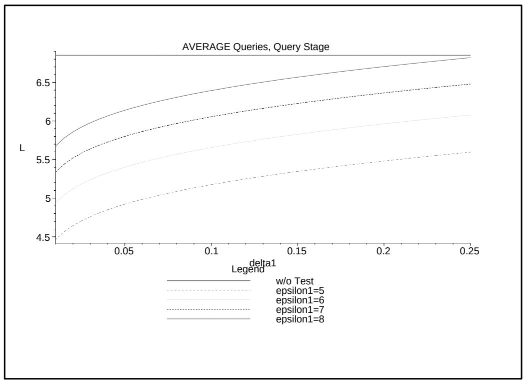

Fig. 14. AVERAGE Queries, Algorithm QueryAvg

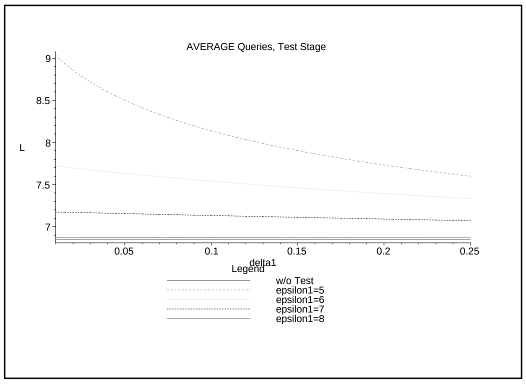

Fig. 15. AVERAGE Queries, Algorithm TestAvg

Figure 14 shows the effect of ϵ_1 and δ_1 for QueryAvg. Figure 15 shows the effect of ϵ_1 and δ_1 for TestAvg.

In our simulations we see that regardless of the choice of ϵ_1 and δ_1 the QueryAvg procedure will query a larger number of sensors than the TestAvg procedure. However, in QueryAvg, varying ϵ_1 has relatively less effect whereas in TestAvg the performance changes drastically when ϵ_1 is decreased. As a result we suggest choosing a smaller ϵ_1 for QueryAvg.

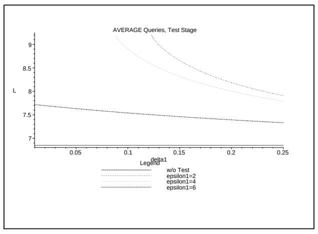

Fig. 16. $\sigma = 2$

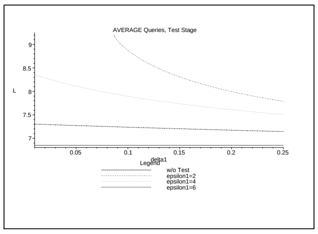

Fig. 17. $\sigma = 3$

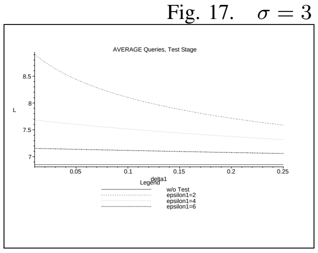

Fig. 18. $\sigma = 4$

Next, we study the effect of σ, which represents the rate of change in the data. One can see in Figures 16, 17, and 18 that σ has a big influence on efficiency. The discontinuity of the lines in Figures 16 and 17 indicates that QueryAvg has tested more sensors than required, making some of the following rounds of TestAvg unnecessary.

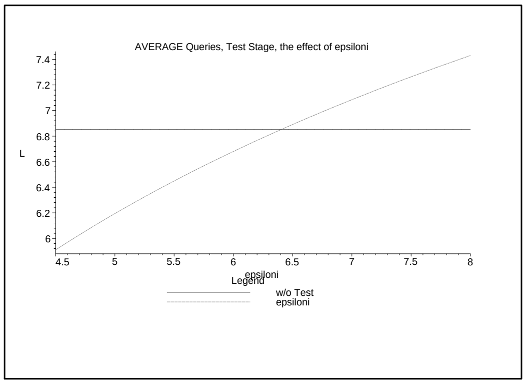

Fig. 19. AVERAGE Queries, the effect of the readings from TestAvg

Note that, when $p = 0.5$, the expected number of sensors in each layer increases by 2 as we go down each layer. To reduce energy consumption, every query performed on some layer $i-1$ rather than layer i must be compensated by 2 or more runs of TestAvg performed on layer $i+1$ rather than layer i, or 1 or more on $i+2$ rather than $i+1$, etc. Thus, the combination of QueryAvg and TestAvg is more profitable when the change in the data is highly predictable. Thus, QueryAvg and TestAvg can be used for emergency monitoring applications in stable environments whereas QueryAvg alone can be used in applications of data acquisition in changing environments.

We then investigate the effect of using TestAvg. Let r denote the output of TestAvg. In our algorithm, if the answer to a query $r \pm \epsilon_i$ is out of the range of $avg_1 + \mu \pm \epsilon$ then we set ϵ_i more stringently. This, however, leads to possibly involving a larger number of sensors for this query. In theory, we stop at $\epsilon_i = 0$, however, in practice, we can stop earlier and move to QueryAvg for efficiency reasons. We observe this effect in Figure 19 with the same data values as in Figure 14 and Figure 15, setting $\epsilon_1 = 6$ and $\delta_1 = 15\%$, which are reasonable choices.

In Figure 19 we see that when the results from TestAvg fall within $avg_1 + \mu \pm 1.5$, fewer sensors are queried, confirming that ϵ has a greater impact than δ for the delay.

B. Energy Consumption Evaluation

In this section we consider the effect of our algorithms on the energy consumption. We assume that our sensor network occupies an area of 100×100, with uniformly distributed positions, and each sensor has 5000 units of energy. We also assume that the queries are generated according to Poisson distribution with mean value $\lambda = 20$. The data queries are generated uniformly from the set {MAX, MIN, QUANTILE, AVERAGE}. The query parameters are assumed to be uniformly generated within the bounds $0 < \delta < 0.5$, and $0 < \epsilon < 0.5$ for QUANTILE. For AVERAGE, ϵ is proportional to the bounds a, b which we set to be 20 and 100. In our energy consumption calculations we use $\alpha = 2$. The promotion probability is set to be $p = 1/2$.

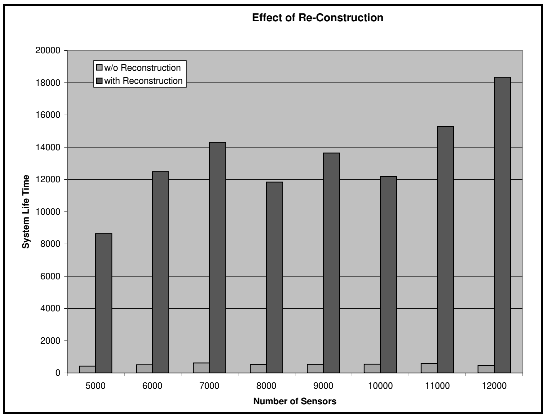

Fig. 20. with Reconstruction vs. without Reconstruction

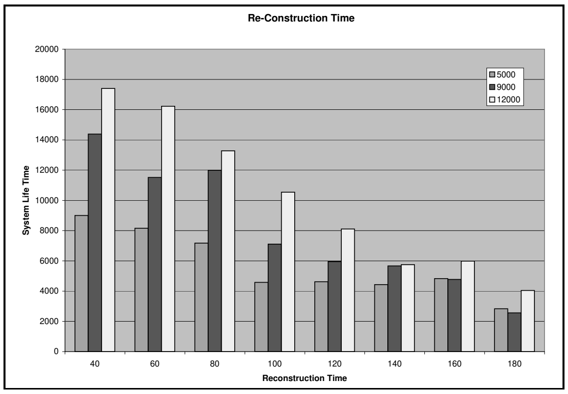

Fig. 21. the Effect of Reconstruction Time

In our experiments, we use the simplest data gathering technique, flooding, where the source and all the intermediate sensors in the same layer just broadcast the query within their transmission range and collect the answers in a reverse fashion. Every data point presented in the figures is the average of 100 random experiments.

We first consider the effect of the reconstruction of the layers in Figure 20. Clearly, without reconstruction, the sensor network depletes much faster. As the number of sensor nodes increases, the lifetime of the system also increases, since the distances between sensors are smaller, leading to less energy consumption.

We see the effects of the reconstruction time in Figure 21; if we reconstruct the sensor network more frequently, the lifetime of the system increases. The effect is more obvious when the number of sensors is large.

VII. Conclusion and Future Work

In this paper we focus on building a layered architecture for a sensor network where queries can be sent to one layer instead of the entire network, to reduce the latency of the queries. We analyze our architecture in terms of accuracy and latency (expressed in terms of the number of sensors involved in answering a query) and prove bounds on those parameters. We present results from a substantial number of experiments to show the relationships between different parameters in our system.

The following research directions are left for future work. 1) There are many other additional types of queries which are of interest, e.g. majority; we would like to explore methods for improving those. 2) Although we explore the effect of ϵ_1 and δ_1 for our query average, it is unknown whether they are optimal; we would like to mathematically search for the optimal values.

Acknowledgements

We thank the anonymous referee for pointing out that more clarification is needed as to why we can use Hoeffding's bounds for Lemma 3. We have included pointers to Hoeffding's paper to make this issue clear.

References

[1] J. Al-Karaki, and A. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey" *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6-28, Dec. 2004.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks" *IEEE Communications Magazine*, vol. 40, no. 8, pp.102-114, Aug. 2002.

[3] S. Banerjee and S. Khuller, "A Clustering Scheme for Hierarchical Control in Multi-hop Wireless Networks" in *Proc. IEEE INFOCOM'01*, Anchorage, AK, Apr. 2001.

[4] D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks" in *Proc. WSNA'02*, Atlanta, GA, Sept. 2002.

[5] A. Boukerche, R. Pazzi, and R. Araujo, "A Fast and Reliable Protocol for Wireless Sensor Networks in Critical Conditions Monitoring Applications" in *Proc. ACM MSWiM'04*, Venice, Italy, Oct. 2004.

[6] C. Buragohain, D. Agrawal, and S. Suri, "Power Aware Routing for Sensor Databases" in *Proc. IEEE INFOCOM'05*, Miami, FL, Mar. 2005.

[7] M. Chu, H. Haussecker and F. Zhao, "Scalable Information-Driven Sensor Querying and Routing for Ad Hoc Heterogeneous Sensor Networks" *International Journal of High Performance Computing Applications*, vol. 16, no. 3, pp. 293-313, Aug. 2002.

[8] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks" in *Proc. ACM MOBICOM'99*, Seattle, WA, Aug. 1999.

[9] M. Greenwald and S. Khanna, "Power-Conserving Computation of Order-Statistics over Sensor Networks" in *Proc. ACM PODS'04*, Paris, France, Jun. 2004.

[10] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks" in *Proc. ACM MOBICOM'99*, Seattle, WA, Aug. 1999.

[11] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks" in *Proc. HICSS'00*, Wailea Maui, HI, Jan. 2000.

[12] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables" *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13-30, Mar. 1963.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communcation Paradigm for Sensor Networks" in *Proc. ACM MOBICOM'00*, Boston, MA, Aug. 2000.

[14] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of Network Density on Data Aggregation in Wireless Sensor Networks" in *Proc. ICDCS'02*, Vienna, Austria, Jul. 2002.

[15] V. Kawadia, and P. Kumar, "the Power Control and Clustering in Ad-hoc Networks" in *Proc. IEEE INFOCOM'03*, San Francisco, CA, Mar. 2003.

[16] B. Krishnamachari, D. Estrin, and S. Wicker, "the Impact of Data Aggregation in Wireless Sensor Networks" *ICDCS Workshop on Distributed Event-based System (DEBS'02)*, Vienna, Austria, Jul. 2002.

[17] S. Lindsey, and C. Raghavendra, "PEGASIS: Power-Efficient Gathering in Sensor Information Systems" in *IEEE Aerospace Conference Proceedings*, vol. 3, 9-16, pp. 1125-1130, 2002.

[18] S. Madden, R. Szewczyk, M. Franklin, and W. Hong, "Supporting Aggregate Queries over Ad-Hoc Wireless Sensor Networks" in *Proc. IEEE International Workshop on Mobile Computing Systems and Application (WMCSA'02)*, Callicon, NY, Jun. 2002.

[19] S. Madden, M. Franklin, J. Hellerstein and W. Hong, "TAG: a tiny Aggregation Service for Ad-Hoc Sensor Networks" in *Proc. OSDI'02*, Boston, MA, Dec. 2002.

[20] N. Sadagopan et al, "the ACQUIRE Mechanism Mechanism for Efficient Querying in Sensor Networks" in *Proc. the First Workshop on Sensor Network Protocol and Applications*, pp. 149-155, May. 2003.

[21] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and Beyond: New Aggregation Techniques for Sensor Networks" in *Proc. ACM SENSYS'04*, Baltimore, MD, Nov. 2004.

[22] J. Wieselthier, G. Nguyen, and A. Ephremides, "On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks" in *Proc. IEEE INFOCOM'00*, Tel-Aviv, Israel, Mar. 2000.

[23] Y. Yao, and J. Gehrke, "Query Processing for Sensor Networks" in *Proc. CIDR'03*, Asilomar, CA, Jan. 2003.

[24] O. Younis, and S. Fahmy "Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach" in *Proc. IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004.

[25] Y. Yu, B. Krishnamachari, and V. Prasanna, "Energy-Latency Tradeoffs for Data Gathering in Wireless Sensor Networks" in *Proc. IEEE INFOCOM'04*, Hong Kong, China, Mar. 2004.

APPENDIX

In this section, we show the effect of the promotion probability p.

Theorem 9: (Theorem 4) To attain the ϕ-quantile of the sensor readings with error bound ϵ and confidence level δ, the query must be sent to layer $l < \log_{\frac{1}{p}} N - \log_{\frac{1}{p}} \left(\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})} \right)$.

Theorem 10: (Theorem 6) Let the data value at each sensor come from the interval $[a, b]$, and let l be such that $l < \log_{\frac{1}{p}} N - \log_{\frac{1}{p}} \left(\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta} + \sqrt{ln\frac{2}{\delta}(2\frac{(b-a)^2 ln\frac{4}{\delta}}{2\epsilon^2} + ln\frac{2}{\delta})} \right)$. Then the probability that the average of the data values on layer l deviates from the exact average by more than ϵ is less than δ.