Histogram and Other Aggregate Queries in Wireless Sensor Networks

Khaled Ammar and Mario A. Nascimento
Department of Computing Science, University of Alberta, Edmonton, Canada
{ kammar,mn}@cs.ualberta.ca

Abstract—Wireless Sensor Networks (WSNs) are typically used to collect values of some phenomena in a monitored area. In many applications, users are interested in summaries of the observed data, e.g., a histogram reflecting the distribution of the collected values. However, processing a histogram query efficiently on top of WSN is a topic that has not received much attention in the literature. In this paper we propose two main contributions: (1) an efficient algorithm for answering histogram queries in a WSN, and (2) how to efficiently use the obtained histogram to process other types of aggregate queries approximately and exactly. Our experimental results using both synthetic and real data sets show that our proposed solutions are able to extend the lifespan of the WSN by up to one order of magnitude and at least double it.

1 Introduction

A typical Wireless Sensor Network (WSN) consists of sensor nodes distributed in an area and connected, via a tree-like topology, to a base station. Typically, WSN nodes have limited resources in terms of power, CPU and memory. Battery lifetime is considered the most important resource in WSN nodes. The required power for transmission is significantly higher than the required power for data processing in a WSN node. For example, sending one bit using the Berkeley Mica motes needs as much energy as processing 1000 CPU instructions [9]. For this reason, it is very important that all algorithms run on top of a WSN minimize the transmission. The base station is a full-fledged computer system with light limitations on memory, CPU, or bandwidth. Nodes are typically used to observe some phenomena about a monitored area and are becoming common in many applications. Examples include smart nursing, security monitoring, structural health monitoring, and monitor environmental variables [3], [7], [15].

Simple aggregate queries like *Max*, *Average* and *Sum* are sufficient for a large number of applications where a high-level (summary) view of the data suffices, e.g., when looking for abnormal behavior. However, complex aggregates like *Histogram* provides a broader picture and is mandatory for many applications. A *Histogram* query is an aggregate query, which provides a statistical summary of the available values. It consists of a set of bins representing numerical ranges and the answer is the count of how many values belong to each bin. These bins' ranges are adjacent, consecutive, non-overlapping and often are chosen to be of the same size. It is also useful for estimating the values distribution and then for computing approximate answers for other aggregate queries.

Many applications need the broader view provided by *Histograms*. For example, in the Electronic Nose project [1], any single value is not important by itself, but, the distribution of the sensor values is used as a chemical signature to classify the material as being safe or unsafe. In engineering, many applications use *Histogram* queries on WSN for different purposes. A civil engineer that needs to measure the pressure along a bridge can obtain a *Histogram* that finds pressure distribution in all bridge's areas [7]. Petrochemical engineers would use the *Histogram* function to understand the fluid directions inside a tube (Fluid direction gives an intuition about fluid pressure in the pumps and pumps network design).

We assume that a WSN has N nodes $s_j \in S$ $(1 \leq j \leq N)$ spread in a monitored area. Each node s_j in S periodically measures a value v_j. In fact, every value has an associated timestamp, however in order to lighten the notation we do not denote it unless necessary. Nodes are connected to the base station by a routing tree, where the base station is the root and, they can reach the base station by multi-hop routing through other nodes. Typically, this tree is constructed to minimize number of hops between any node and the base station with respect to the maximum range of each node. We assume that the connection between nodes are reliable (no link failure), and we focus on the data aggregation problem only.

Users are connected to the WSN through the base station where they can submit their queries. The base station forwards a query to the WSN, collects its result, and then sends its answer back to the user.

A *Histogram* query of B bins is formally defined as: $Q = (Lb, Ub, b_1, b_2, b_3, ...b_B, epoch)$, where $epoch$ is

Technical Report TR 11-03. February 2011. Dept. of Computing Science. University of Alberta. Canada.

Symbol	Meaning
S	A set of all sensors in the WSN
s_j	A single sensor in the WSN S
v_j	The value of a sensor s_j
T	The tree-like topology connecting all sensors
L	The hight of tree T
Q	A *Histogram* query
H	A *Histogram* answer
N	Number of sensors
B	Number of bins in the *Histogram* query
b_i	A bin range in the *Histogram* query
h_i	Number of values in bin b_i
I_j	Initial state in a node s_j
P_j	Partial state in a node s_j
U_j	update-message sent from a node s_j to its parent
M	Array of received messages

TABLE 1
Summary of the notation used in this paper

the time lapse between any two consecutive *Histogram* answers. The lower and upper bound values of the measured phenomena are Lb and Ub. Each b_i is a bin in the *Histogram* query and it is defined as an interval between Lb and Ub, where:

- $b_i = [Lb_i, Ub_i) \ \ \forall 1 \leq i < B$ and $b_B = [Lb_B, Ub_B]$
- $Ub_i \leq Lb_j \ \forall i < j$
- $\bigcup_{1 \leq i \leq B} \{b_i\} = [Lb, Ub]$
- $Lb_1 = Lb$ and $Ub_B = Ub$

The answer for a *Histogram* query is $H = (h_1, h_2, ..., h_B)$, where $h_i = \mid \{(s_j, v_j) \mid Lb_i \leq v_j < Ub_i, s_j \in S\} \mid$. Naturally, a sensor's value v_j may change every epoch and so does the query answer. Table 1 summarizes the notation used in this paper.

This paper presents two main contributions. The first is an efficient distributed algorithm to answer *Histogram* queries. This algorithm require less than half amount of energy used by the classical TAG algorithm to construct a *Histogram* thus, it can at least double the network lifetime. Our second contribution is a set of algorithms to answer other aggregate queries including *Median*, which has not received its due share of attention in the related literature. The proposed algorithms find approximate as well as exact answers. Approximate answers can be obtained with no overhead, whereas exact answers require a very small overhead on the WSN.

The rest of this paper is organized as follows: Section 2 reviews the classical TAG approach answers a *Histogram* query and explains in details our proposed algorithm. How to compute approximate as well as exact answers for other aggregate queries using a *Histogram* as a starting point is discussed in Section 3. Section 4 presents our experiments and Section 5 shows the related work in the literature to the proposed algorithms. Finally, Section 6 concludes the paper and introduces a few future directions for further research.

2 In-network Algorithms for *Histogram* Queries

2.1 Background

The straightforward technique to build a *Histogram* is to periodically gather all values from all sensors at the base station and then build a histogram. The classical TAG algorithm decreases the number of required messages extensively comparing to the straightforward technique [8]. The authors define a model to answer aggregate queries using an in-network algorithm in the WSN context. Each node has initial and partial states. Nodes send their partial states to their parents[1]. Received partial states are merged together to construct the parents' partial states. The process can be visualized as a routing tree where the base station is the root and nodes send their partial states as messages up the tree towards that root. This process continues until constructing a partial state in the base station. Finally, the base station runs an evaluation function to compute the aggregate result from its partial state.

The authors classify aggregate queries based on their properties. One of these properties is the partial state size. Since what is the format of the initial and partial state in this case is not discussed in details in [8], we assume they have the following format:

- I_j is the initial state in a node s_j with a value v_j, represented by a pair $p_i = (i, h_i)$ where $v_j \in b_i$ and h_i is number of values in bin b_i.
- P_j is the partial state in a node s_j, and is an array of pairs p_i. It is constructed by merging partial states received from the children of s_j along with its own initial state.

Using the above assumptions, a TAG-base algorithm takes the *Histogram* query Q, the tree-like topology T, and starts a bottom-up merging of partial states. The pseudo-code for this algorithm is illustrated in Algorithm 1 where M is an array of all received messages from sensor s_j's children. In general, a message in M could be a single value or a partial state and occasionally include other information, but in the TAG-Histogram algorithm all messages in M have partial states only.

The MERGE function (illustrated in Algorithm 2) receives an array M of all received messages and an initial state I_j and returns a partial state. If M is empty then the sensor s_j is a leaf node and its partial state $P_j = I_j$. Otherwise, the MERGE function sums up all h_is relative to the same bin b_i from different messages, and adds a new pair p_i to the result array R.

When the collected partial states are merged together in the base station, level $l = 0$, the evaluation function computes the query result, H. The proposed evaluation function finds h_i in the final partial state P_j and copy it into the query answer H. If a count h_i $(1 \leq i \leq B)$ is not present in P_j, then its value is $zero$.

1. In case of leaf nodes, the partial state is identical to the initial state

Algorithm 1 TAG-Histogram(Histogram Query Q, Logical Routing Tree T)

1: $l \leftarrow L$ {L is number of levels in tree T}
2: **while** $l \geq 0$ **do** {Iterate on all levels using}
3: **for** each sensor s_j with a value v_j in level l **do**
4: $I_j \leftarrow (i,1) \mid i \leftarrow arg\{b_i \mid v_j \in b_i\}$ {Initial state for s_j}
5: $P_j \leftarrow$ MERGE(M,I_j) {The sensor s_j's partial state is based on I_j and all messages M from its children}
6: Send P_j to s_j's parent
7: $l \leftarrow l-1$ {move to the upper level}
8: **return** P_j in T's root

Algorithm 2 MERGE(Array of Messages M, Current Sensor's initial state I)

1: **if** M is empty **then**
2: **return** I
3: **else**
4: $R = \{\ \}$ {R will contain a set of pairs $p_i = (i, h_i)$}
5: $R \leftarrow I$
6: **for** each message m in M **do**
7: $P_m = \{\ \}$ {Partial state P_m will contain a set of pairs $p_i = (i, h_i)$}
8: **if** the message m is a value v_m **then**
9: $P_m = (i,1) \mid i \leftarrow arg\{b_i \mid v_m \in b_i\}$
10: **else** {the message m is a set of pairs}
11: $P_m \leftarrow$ all pairs in the message m
12: **for** each pair $p_i = (i, h_i) \in P_m$ **do**
13: **if** $\exists\ p_k = (k, h_k) \in R \mid k = i$ **then**
14: $h_k \leftarrow h_k + h_i$
15: **else**
16: Copy p_i from P_m to R
17: **return** R

In this approach, each sensor should send exactly one message on every epoch but the message sizes (in bits) are different depending on the node type. The size of a message from a leaf node is $size(P_j) = size(p_i) = \log_2 N + \log_2 B$, whereas the size of a message from a non-leaf node is $size(P_j) = size(p_i) \times B$. If the distribution of sensor values is wide and covers most of the *Histogram* bins, then the message format is not efficient because it includes the bin id (i) with each pair. In that case, if an array of all h_is, including the zero'ed ones, is sent without any bin id, the message size will be smaller ($\log_2 |S| \times N$).

TAG [8] is the current state-of-the-art approach to build a *Histogram* based on WSN nodes' values. In the next section we propose an algorithm that requires less and smaller messages sent in the network. The main idea is using in-node caching and sending incremental updates instead of actual values.

2.2 Histogram Incremental Updates (HIU) Algorithm

A sensor does not change a *Histogram* answer if its value changes within its current bin's lower and upper bounds. This *Histogram* property motivates us to look into more details of the *Histogram* construction process. Instead of sending its information every epoch, a sensor can build an update message based on the previous round's information. This idea was used in several algorithms in the literature. For example, in [5] the authors proposed algorithms to maintain materialized views incrementally. In our algorithm, sensors receive incremental *Histogram* updates, merge them together and then forward to their parents, and so forth. The process continues until the *Histogram* in the base station is updated.

In-node caching is an essential component in the HIU algorithm. Each node s_j caches its value and its partial state from the previous round in $\tilde{v_j}$ and $\tilde{P_j}$, respectively. In the first round, $\tilde{v_j}$ is undefined and assume that $\tilde{P_j}$ is $\{0, 0, 0, ..., 0\}$.

The HIU algorithm works as follows (Pseudo-code is shown in Algorithm 3). The initial and the partial states in HIU are both equivalent to the partial state in TAG. The initial state has two pairs if the current value v_j belongs to a different bin than the previous cached value $\tilde{v_j}$, and has one pair only if $\tilde{v_j}$ is undefined which means bin id = -1.

Nodes do not always send their partial states to their parents. A leaf node sends its partial state if the new value leads to a change of its bin. A non-leaf node may receive multiple values and update-messages from its children (array M). Update-messages have the same format as a partial state. If a message in M is a single value, MERGE converts it to the partial state format and continues. Merging all received messages in a sensor s_j with its initial state I_j yields its update-message U_j. The update-message U_j is applied to the cached partial state $\tilde{P_j}$ to keep it up-to-date. This step adds each h_i in U_j to h_k in $\tilde{P_j}$ iff $i = k$. Note that h_i values in U_j could be negative values. In fact, for all update-messages U_j, $\Sigma_{i=1}^{B} h_i = 0$.

Received update-messages in any non-leaf node may cancel each other in which case nothing is sent forward. For example, consider non-leaf node C that has two subtrees, A and B. Subtree A has x nodes where their value moved from bin b_k to b_l. On the other hand, the subtree B has $x-1$ nodes where their values moved from bin b_l to b_k. If these two updates are merged together, then subtree C has only one value moved from b_k to b_l. Moreover, if node C's value moved from b_l to b_k, then C should not send any update to its parent at all.

As discussed earlier, the Encoding function (in line 21) decides whether to send U_j as a set of pairs (i, h_i) or send all h_is without the need to identify them with bin ids i then attach E,if an exact answer is required, with the message. The smaller representation, based on number of bits, is chosen. A more complex compression can be implemented for this function, e.g., [12], [16]. However,

Query	Approximate Answer Equation	Error Margin
Max	$\frac{Ub_m+Lb_m}{2} \mid h_m \neq 0, \forall i > m, h_i = 0$	$\frac{Ub_m-Lb_m}{2}$
Min	$\frac{Ub_m+Lb_m}{2} \mid h_m \neq 0, \forall i < m, h_i = 0$	
Median	$\frac{Ub_m+Lb_m}{2} \mid \Sigma_{i=1}^{m}\{h_i\} \geq \frac{Count}{2} \wedge \Sigma_{i=m}^{B}\{h_i\} \geq \frac{Count}{2}$	
Count	$\Sigma_{i=1}^{B}\{h_i\}$	0
Sum	$\Sigma_{i=1}^{B}\{h_i \times \frac{Ub_i+Lb_i}{2}\}$	$\Sigma_{i=1}^{B}\{\frac{Ub_i-Lb_i}{2} \times h_i\}$
Average	$\frac{Sum}{Count}$	$\frac{1}{Count}\Sigma_{i=1}^{B}\{\frac{Ub_i-Lb_i}{2} \times h_i\}$

TABLE 2
A summary for supported aggregate queries

a detailed discussion about compression algorithms in WSN is beyond the scope of this paper.

In the next section we show how a *Histogram* could be used to compute approximate answers for aggregate queries. We also show how the ExtraInformation function (in line 20) works to collect necessary values and send them through in-network to facilitate computing exact answers for other aggregate queries in the base station.

3 Other Aggregate Queries

A *Histogram* provides a broad picture for values in the WSN and is a starting point for more statistical analysis. Occasionally, a user might like to know more specific information (e.g. *Max* or *Average*) about those values represented by the histogram. In this section, we present algorithms to compute approximate and exact answers for several aggregate queries using a previously obtained *Histogram* in the base station. The approximate solutions have bounded accuracy levels and the exact solutions can be computed with very low extra overhead on the WSN.

Recall that we consider a *Histogram* query is defined as: $Q = (Lb, Ub, b_1, b_2, b_3, ...b_B, epoch)$ and its answer is: $H = (h_1, h_2, h_3, ...h_B)$, where $h_i = \mid \{(s_j, v_j) \mid Lb_i \leq v_j < Ub_i, s_j \in S\} \mid$. Table 2 shows how to compute approximate answers for some aggregate queries in the base station using a *Histogram* result. Because, all computations are made on the base station, there is absolutely no overhead on the WSN.

Table 2 also shows the error margin limit for each approximate aggregate answer. Trivially, a *Histogram* query can provide an exact answer for *Count* aggregate query. All margin limits depends on the bin size ($Ub_i - Lb_i$). Decreasing the bin size in the *Histogram* query will lead to answers with higher accuracy. However, this will increase the overall cost of the *Histogram* result because it increases number of sent bytes.

Next we propose algorithms to compute exact answers for different types of aggregate queries. We can obtain exact answers using two strategies: (1) adding some overhead to the HIU messages but not extra messages, or (2) sending a refinement query, thus extra messages, with no overhead on HIU *Histogram* messages.

3.0.1 Exact answers using per Message overhead

Communication devices in some WSN mandate the sensor to send messages of fixed size only [11]. In this case, sending less information will not decrease the energy consumption because all buckets should have the same size. These idles bytes can be used to send the extra information, with no extra cost, to facilitate computing the exact answer. We use this strategy to compute exact answers for *Max*, *Min*, *Sum*, and *Average* queries.

While obtaining a *Histogram* answer, the ExtraInformation function (line 20 in Algorithm 3) collects required information and aggregate them to facilitate computing the exact answer in the base station. If an exact answer is required, then leaf nodes values are needed even if the value's bin did not change.

The behavior of ExtraInformation function (Algorithm 4) depends on the required aggregate query. *Max* and *Min* queries are handled from line 2 to line 12, while *Sum* and *Average* are starting on line 14 to line 22. For simplicity, Algorithm 4 handles only one aggregate function at a time, but it can be easily extended to support multiple aggregate queries.

In order to find the exact answer for *Max* (or *Min*) queries, all intermediate nodes who construct a partial status should report information about the maximum (or minimum). The code between lines 2- 12 in Algorithm 4 will compute the maximum for each subtree. The pseudo code assumes that each message sent from an intermediate node s_j to its parent includes E_j, that includes the maximum value over the node's subtree.

Because the base station (and all intermediate nodes) already has an exact answer for *Count*, they can compute the exact result for *Average* if the exact *Sum* is available. The exact answer for *Sum* can be computed if each intermediate node sends the total sum of its subtree while leaf nodes send their own values only. The code between lines 14 - 22 in Algorithm 4 computes the sum of all values in a subtree rooted by each intermediate node.

If the used communication device allows variable message size, then every bit is counted when calculating the energy consumption. We can decrease the size of the Max(Min) and Sum(Average) overheads using extracted information from the *Histogram* because each intermediate node will send an update for its partial state (histogram) to its parent.

Algorithm 3 HIU(Histogram Query Q, Logical Routing Tree T, Aggregate Query Agg)

$l \leftarrow L$ {L is number of levels in tree T}
while $l \geq 0$ **do** {Iterate on all levels using bottom-up }
  for each sensor s_j in level l **do**
    $b \leftarrow arg\{b_i \mid v_j \in b_i\}$ {b is the bin id for the current value v_j}
    if the cached value $\tilde{v_j}$ of sensor s_j is undefined **then**
      $\tilde{b} \leftarrow -1$ {-1 is an alias for an undefined bin}
    else
      $\tilde{b} \leftarrow arg\{b_i \mid \tilde{v_j} \in b_i\}$
    {$\tilde{b}$ is the bin id for the cached value $\tilde{v_j}$}
    if a sensor s_j is a leaf-node **then**
      if $b \neq \tilde{b}$ OR $Agg \neq$ NULL **then**
        Send v_j to s_j's parent
      {No data sent if current and cached bins are equal}
    else
      {Construct Initial state (I_j)}
      if $\tilde{b} = -1$ **then**
        $I_j \leftarrow (b, 1)$
      else
        $I_j \leftarrow$[(*b*,1), ($\tilde{b}$,-1)] {increase the counter of the current bin by 1 and decrease the cached by 1}
      $U_j \leftarrow$ MERGE (M , I_j) {Build sensor's update message U_j using sensor's I_j and messages in M}
      if Agg= NULL **then**
        $E \leftarrow$ ExtraInformation (s_j, Agg) {It returns the necessary value to allow computing the exact answer for an aggregate query Agg.}
        Send Encode(U_j,E) to s_j's parent
      else
        Send Encode(U_j) to s_j's parent
      {Update the cached partial state $\tilde{P_j}$ from the previous round}
      for each pair $p_i = (i, h_i) \in U_j$ **do**
        if $\exists\ p_k = (k, h_k) \in \tilde{P_j} \mid k = i$ **then**
          $h_k \leftarrow h_k + h_i$ {Recall that in U_j, h_i could be positive or negative}
        else
          Copy p_i from U_j to $\tilde{P_j}$
    $\tilde{v_j} \leftarrow v_j$
  $l \leftarrow l - 1$ {Go to the upper level}
return $\tilde{P_j}$ in T's root

Instead of reporting the real value of the Maximum (Minimum), nodes select a bin id (m) that includes the maximum value and send the difference between bin's lower bound (LB_m) and this value ($MAX - Lb_m$). In the worst case, the overhead will be number of bits required to represent the bin size ($Ub_m - Lb_m$) which is $log_2(Ub_m - Lb_m)$ bits per node, every epoch.

Following the same idea, instead of sending the real Summation of all values in a node's subtree which might need large space, each the node will send difference = $SUM - \Sigma_{i=1}^{B} \frac{Ub_i + Lb_i}{2} \times h_i$. The maximum possible value of SUM in any node is $\Sigma_{i=1}^{B} Ub_i \times h_i$, if all values in all bins equal the upper bound of this bin. The maximum possible overhead per node per round is $log_2(\Sigma_{i=1}^{B} \frac{Ub_i - Lb_i}{2} \times h_i)$.

Algorithm 4 ExtraInformation(Sensor Node s_j, Aggregate Query Agg)

$E = 0$ {E is the returned value to allow computing the exact answer for an aggregate query Agg}
if $Agg = Max$ **then**
  $MAX = v_j$ {v_j is the current value of sensor s_j}
  for each message m sent to a sensor s_j **do**
    if m is a value v_m **then** {m was sent by a leaf node}
      $temp \leftarrow v_m$
    else {m was sent by an intermediate node}
      $temp \leftarrow E_m$
      {E_m is a value in m represents the max of sender's sub tree.}
    if $MAX < temp$ **then**
      $MAX \leftarrow temp$
  $E \leftarrow MAX$
{The code for *Min* is very similar to *Max* and omited from this code}
if $Agg =$*Sum* or $Agg =$*Average* **then**
  $SUM = v_j$
  for each message m sent to a sensor s_j **do**
    if m is a value v_m **then** {m was sent by a leaf node}
      $SUM \leftarrow SUM + v_m$
    else {m was sent by an intermediate node}
      $SUM \leftarrow SUM + E_m$
      {E_m is a value in m represents the sum of sender's sub tree.}
  $E \leftarrow SUM$ {This is sufficient for both *Avergae* because *Count* is known}
return E

3.0.2 Exact answer using refinement queries

Typically, *Median* and K^{th}*value* queries require sending all values to the base station [8]. Recently a new algorithm for Top-K [10] query was proposed to find exact top K values and can be used to find the k^{th} value, hence the median. However, this algorithm will require sending all K values to the base station. If the WSN has

large number of sensors (values), computing the *Median* value might have a large cost.

Using a *Histogram* result, we can narrow the requirements from collecting all values in the WSN to collecting all values in the median bin. A bin that contains the median value can be identified based on the *Histogram* answer in the base station. It is a bin where total number of values in all preceding and all following bins in the *Histogram* are both less than half the number of values in the whole histogram. Table 2 uses this formula: $m = arg\{b_m \mid \Sigma_{i=1}^{m}\{h_i\} > \frac{Count}{2}$, and $\Sigma_{i=m}^{B}\{h_i\} > \frac{Count}{2}\}$ to identify the bin id[2]. Instead of reporting the average of the median bin as an approximate answer, the base station may send another query to the WSN requesting all values in this bin. The query will be guided to the relevant nodes only using the cached partial states. The full algorithm is presented in Algorithm 5. It has two parameters: the WSN logical routing tree (T) and the *Histogram* answer (H) in the base station and returns the exact median.

Function GetVALUES returns all values in the median bin as an array. After sorting the returned values and using the number of total values in the WSN, the ExactMEDIAN algorithm computes and returns the exact *Median*. The algorithm for function GetVALUES is straightforward and is represented in Algorithm 6. It collects all values between values L and U in the given tree T. However, instead of sending the request to all sensors in the tree, it uses the cached partial states to prune branches leading to subtrees that have no values within the requested boundaries. In the worst case, all values would be in leaf nodes in the maximum tree's level L. The maximum possible overhead every round is the number of bits to retrieve these values which is $log_2(h_m \times L \times max(v_j))$ where $1 \leq j \leq N$ and m is the bin id that contains the median value.

Algorithm 5 ExactMEDIAN(Logical Routing Tree T, *Histogram* Answer H)

1: $Count \leftarrow \Sigma_{i=1}^{B}\{h_i\}, \forall h_i \in H$ {Histogram query Q has B bins}
2: $b_m \leftarrow$ A bin in query Q of index $m \mid \Sigma_{i=1}^{m}\{h_i\} \geq \frac{Count}{2}$ and $\Sigma_{i=m}^{B}\{h_i\} \geq \frac{Count}{2}$,$\forall h_i \in H$ {Find the median bin to use its boundaries}
3: V = GetVALUES(T, Lb_m, Ub_m) {V is an array of all values in T within b_m boundaries (Lb_m and Ub_m)}
4: V = SORT (V) {Sort V values in ascending order}
5: $C \leftarrow \frac{Count}{2} - \Sigma_{i=1}^{m-1}\{h_i\}$
6: **return** $Median \leftarrow C^{th}$ value in V.

Algorithm 6 GetVALUES(Tree T, Lower bound L, Upper bound U)

1: {This function uses the partial state P_j in each node s_j to collect required values efficiently.}
2: V = { } {It will contain a set of collected values}
3: $s_j \leftarrow$ The root of tree T
4: **if** $v_j \in [L, U]$ **then**
5: Insert v_j in V {v_j is the value of sensor s_j}
6: **if** $\Sigma_{i=1}^{B}\{h_i\} = 1 \mid b_i \cap [L, U] > 0, \forall h_i \in P_j$ and $\forall b_i \in$ *Histogram* query Q **then**
7: **return** V {The single value in the tree is already found}
8: **if** $\Sigma_{i=1}^{B}\{h_i\} > 0 \mid b_i \cap [L, U] \neq \phi$, $\forall h_i \in P_j$ and $\forall b_i \in$ *Histogram* query Q **then**
9: **for** each children c of s_j **do**
10: Insert $GetVALUES(c$'s Tree$, L, U)$ to V {Insert returned values to the array of values V}
11: **return** V

4 Performance Evaluation

In our simulation we implemented TAG and HIU assuming both of them are using a Shortest Path (logical) Tree (SPT) for the underlying tree T. We make the following assumptions about the required storage: (1) A node value consumes 2 bytes, and (2) a complete *Histogram* size depends on the number of bins, i.e., it requires $2 \times B$ bytes, where B is the number of the bins in the histogram, and (3) updating a *Histogram* bin require 3 bytes, 1 for the bin id and 2 for bin's count.

We investigate our algorithms with respect to two datasets (the synthetic dataset and a real dataset) and five parameters (Radio range R, *Histogram* size in terms of number of bins B, average amount of change in sensor's value δ, the probability that a sensor's value change ρ, and number of nodes in the WSN N).

The radio range controls the logical network topology and may increase/decrease network depth. Varying the *Histogram* size shows the scalability of our algorithm from the point of view of the *Histogram* size. Studying δ and ρ shows the sensitivity of our algorithm against the behavior of the values in the WSN field. It is important to show the influence of these two parameters because our algorithm depends on incremental updates which might be very large if many changes happen. Finally, increasing the number of sensors N, shows the algorithm scalability from the point of view the WSN density.

Table 3 has a list of all tested values for all parameters. While testing one parameter, we use the default value (denoted in bold) of all other parameters. The reported show the average values obtained over 20 runs. During each run, the sensor locations are randomly distributed and the base station is randomly selected among one of the sensors. In order to ensure a fair comparison, both TAG and HIU use exactly the same setup.

Our synthetic dataset consists of N nodes uniformly distributed in an area of $200m \times 200m$. The values of all sensors are initialized uniformly between 1 and 2^{16}. In each round, a sensor's value could change with a probability ρ. In case of change, a sensor value is increased by an exponential random variable (equally

2. The formula can be easily changed to solve a K^{th}*value* instead of a *Median* query

Parameter	Used Values
R (WSN node's radio range)	$20, \mathbf{30}, 40, 50, 60$
B (Histogram size in terms of number of bins)	$5, 10, \mathbf{20}, 40, 60$
δ (Average amount of change)	$1\%, 25\%, \mathbf{50\%}, 75\%, 100\%$
ρ (Probability of change)	$1\%, 25\%, \mathbf{50\%}, 75\%, 100\%$
N (Number of Sensors)	$1000, 2000, \mathbf{3000}, 4000, 5000$

TABLE 3
Studied parameters and their values (default values in **Bold**)

likely to be negative or positive). The average of the exponential random variable is $\delta\%$ of 2^{16}. We assume that all sensors capable of sensing values between 0 and 2^{16} only. If a value exceeds that range in either direction, it is assumed to be either 0 or 2^{16}, respectively.

The real dataset was generated by Intel Berkeley Research Lab [6]. It has 54 WSN nodes deployed in a $50m \times 50m$ area. The dataset has values for about $65,000$ rounds. Missing values from the original dataset were placed using simple interpolation. In this dataset, we only studied two parameters: the radio range (R) and the *Histogram* size (B), because the number of nodes (N), the change probability (ρ) and the average amount of change (δ) are all fixed in any real dataset.

Since the main typical goal within the realm of WSN research is the minimize energy consumption we use network lifetime as the performance indicator. Network lifetime is counted in number of rounds until the first node dies. In all our experiments we assume that each battery's initial budget is $30mJ$. Energy consumption is calculated after [4] :

- $E_t = S + t \times b \times d^2$
- $E_r = r \times b$

where $S = 50$ nJ is the setup cost to send any message, $t = 10$ pJ and $r = 50$ nJ are the required amount of energy to send or receive one bit for one meter, respectively. The message size in bits is b, while the euclidean distance (in meters) between the sender and the receiver is d.

4.1 Performance Evaluation for *Histogram* and Approximate Aggregate Queries

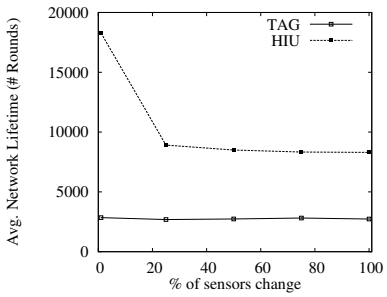
Figure 1 shows the HIU and TAG performance when changing each of the studied parameters using the Synthetic dataset. Because TAG performance does not depend on changes in sensors' values, a network using TAG algorithm died after about 2700 rounds regardless of the change probability (ρ) or amount of change per round (δ). Figures 1(a) and 1(b) show that HIU performs better when the changes of nodes' values happen less frequently because it caches the result of the previous round and send updates only, if required. In case of a higher update frequency (ρ) or update with large changes (δ), HIU performance becomes stable. The reason for that is two fold. First, HIU selects whether to send updates only (update pairs) or send the full histogram. This arbitration saves HIU from sending non useful data if all bins are required. Second, because the partial state (local histogram) is constructed in network, many of these changes do not get communicated as they cancel each other in the early stages of the routing tree.

Figure 1(c) shows that both TAG and HIU perform better when the radio range is small. This seems to contradict the following intuition: the smaller the radio range the more hops are required from leaf nodes to reach the base station, the more messages and then the shorter network lifetime. In reality, each node sends a message to reach all the other nodes within its range regardless of the real distance between the sender and the receiver. The larger the radio range the larger the energy consumed, because energy consumption is based on how far a message can reach and is not based on the euclidean distance between the sender and receiver. The Figure shows that even though the performance of both HIU and TAG is better when the radio range is smaller, HIU multiplies the network lifetime three or four times comparing to TAG.

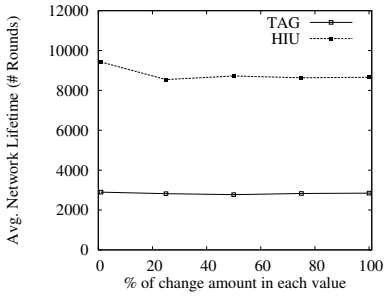
Figure 1(d) is an evidence that HIU can still multiply the network lifetime by at least a factor of two as number of bins increases. A larger number of bins means a higher probability that the number of changed bins gets higher and then HIU performs worse. However, TAG requires all intermediate nodes to send their partial state regardless of number of nodes, i.e., TAG also performs worse when increasing number of bins.

Figure 1(e) shows that HIU can scale efficiently and handle WSNs with large number of bins better than TAG. We basically increase the network density in the field. HIU has the same performance regardless of number of sensors in the field. TAG's performance decreased dramatically because the more sensors in the field the higher probability of occupying all *Histogram* bins. Recall that TAG sends the bin's count if the bin is occupied by one or more values. On the other hand, because of our encoding, the values distribution does not influence HIU performance. The key factor is the how frequent values change and by how much.

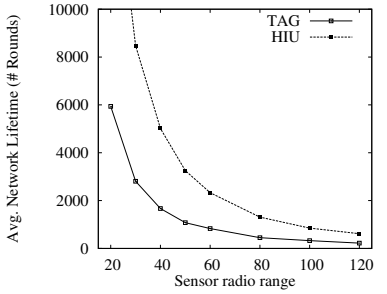
The performance of TAG and HIU on the Intel Berkeley dataset is better than their performance on a synthetic dataset because the number of nodes (54) is significantly smaller and is the amount of communication. However, the performance of HIU on the Intel Berkeley dataset is much better because nodes' values in a real dataset do not usually change with a high probability or high amount. Figure 2 shows that HIU allows the network to



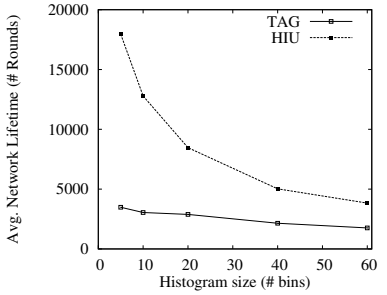
(a) Probability a value changes (ρ)



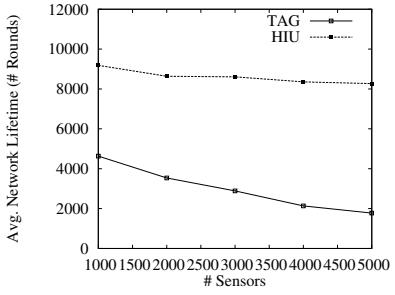
(b) Average amount of change per value (δ)



(c) Radio range (R)

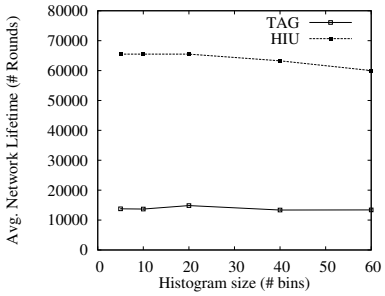


(d) Histogram size (B)

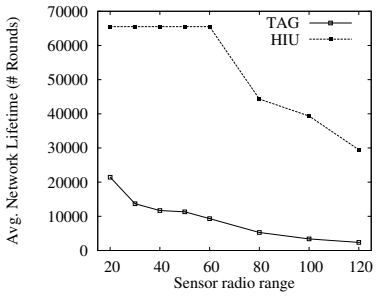


(e) Number of Sensors (N)

Fig. 1. Network lifetime analysis using a Synthetic dataset



(a) Histogram size (B)



(b) Radio range (R)

Fig. 2. Network Lifetime analysis using the Intel Berkeley dataset

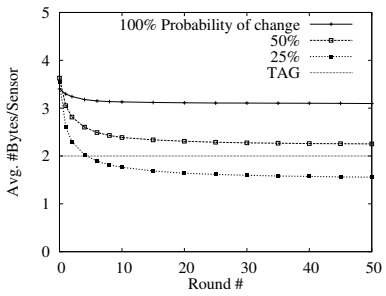
last significantly longer than TAG (sometimes by a factor of 10). In the real dataset, there is a limitation on number of rounds because the dataset has about 65,000 rounds only. HIU curves in Figure 2 reaches the upper limit for number of rounds but none of the nodes die.

4.2 Performance Evaluation for Exact answers

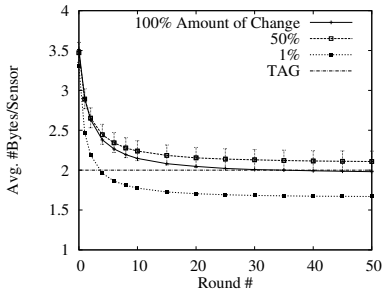
Regardless of the algorithm used to construct a *Histogram* in the base station, a *Histogram* allows computing approximate answers for several other aggregate queries without any overhead as discussed in Section 3. The base station can also compute the exact answer for an aggregate query by using HIU algorithm with some overhead, i.e., extra message.

Because the main target of our experiment is to study the HIU overhead cost for computing an exact answer, we use the average amount of bytes sent per sensor per round as our performance indicator. Every round, the total number of sent bytes from all nodes during all previous rounds are calculated and then divided by number of sensors.

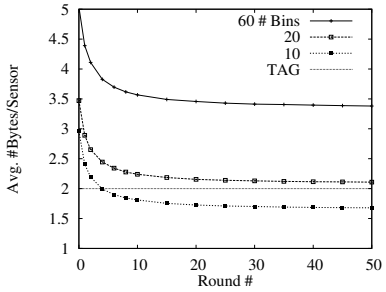
Based on [8], every sensor should send exactly 2 bytes to collect the maximum value using TAG. HIU collects the *Max* information while constructing the histogram. HIU's performance depends on the amount of changes in the network because it uses in-network caching and send data to update this cache. Initially HIU requires more bytes to be sent, but as time goes, the average total number of sent bytes per round is decreased and eventually reaches a steady state. Recall that the first round in HIU consumes a large amount of energy due to



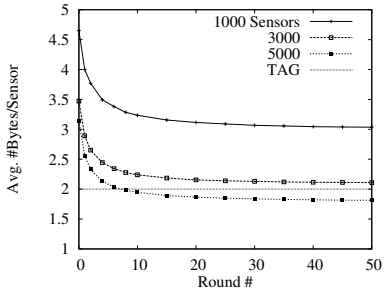
(a) Probability a value changes (ρ)



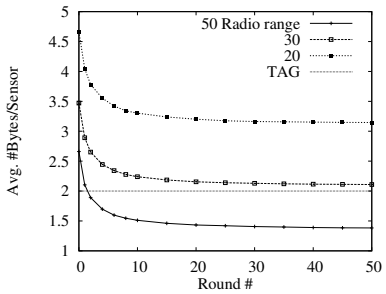
(b) Average amount of change per value (δ)



(c) Histogram Size (B)



(d) Number of Sensors (N)



(e) Radio Range (R)

Fig. 3. Cost of running exact-Max query

sending the largest amount of bytes comparing to other rounds because there is no cached information.

4.3 *Max* queries

Figure 3 shows the amortized analysis for TAG and HIU algorithms in computing the exact *Max* using three parameters: ρ, δ, B, N, and R. While testing one parameter, other parameters are assumed to have their default values (Table 3). The main goal is to show that HIU can outperform TAG in terms of the total cost on the long run. We use the synthetic dataset to have more control on the experiment. Moreover, Figure 2 shows that HIU performs even better on the real dataset any way.

Figure 3(a) shows the influence of change probability on HIU. If the probability is 100% then HIU needs one extra byte from each sensor (on average) per round. As the probability gets smaller, the overhead decreases. The figure shows that lower values of ρ leads to a smaller HIU cost but TAG's performance stay the same. If the probability is 1% only, HIU outperforms TAG by about 1.8 bytes which means 90% less bytes than TAG. It is worth mentioning that HIU's cost includes, also, constructing an accurate *Histogram* in the base station while TAG (in this experiment) computes the maximum value only. The *Histogram* in the base station offers computing approximate answers for many other aggregate queries. This means, if the target is computing the *Max* query only, then HIU is better only if sensors change their values not very often ($\rho \leq 40\%$).

In Figure 3(b) we assume that $\rho = 50\%$ and investigate the influence of the amount of change (δ). If δ is very small (1%) HIU will outperform TAG. If δ is very large (100%), HIU ties with and slightly out performs TAG. Recall that a sensor can sense a specific range of values. If the value is bigger than maximum value, a sensor will report its maximum limit. If the average amount of change is 100% then there is a high probability that all sensors end up detecting only the maximum and minimum limits because the change could be positive or negative. It is clear that the amount of change does not have a significant influence on the results. In fact, regardless of the amount of change, the probability of change (ρ=50%) has the main influence not the amount of change. The only exception is the change of 1% curve because changing a sensor value by 1% on average will unlikely changing its bin in the histogram, if the bin's width is reasonably big, and then unlikely to cause a sensor to send any data.

HIU performance depends on the bin size (number of bins) because the number of values in smaller bins is more likely to change every epoch. Although the error bound of all approximate answers get worse when bin size increase, the HIU algorithms performs better while computing exact *Max*. Figure 3(c) shows that decreasing number of bins can make HIU outperforms TAG very early even if the probability of change and amount of change are both 50%. Recall that TAG outperforms

HIU in Figures 3(a) and 3(b) when δ or ρ equals 50%. The major fraction of the HIU cost is paid to construct the histogram. Decreasing *Histogram* size decreases the *Histogram* overhead but increases the *Max* overhead ($log(Ub_i - Lb_i)$), if the maximum value changes. This overhead is already very small comparing to *Histogram* cost.

Network density depends on the number of sensors (N) that reside in the same fixed area. Network density has no influence on the TAG algorithm to compute *Max*. In all cases, each sensor should report its value. In the case of HIU, the more sensors available in the area the more opportunities to save and decrease the amount of sent messages. Figure 3(d) shows that increasing the network density more than 3000 sensors in 200×200 area (0.075 sensor$/m^2$), makes the HIU outperforms TAG.

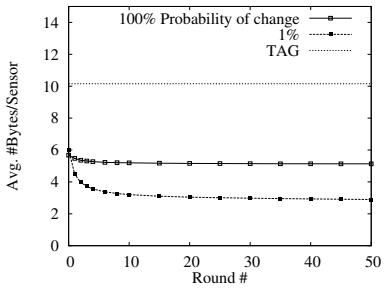
The sensor's radio range influences the logical tree structure. A short radio range requires the WSN to build a logical tree with larger depth than a long radio range. Increasing the average number of hops for sensors to reach the base station does not have any influence on TAG because every sensor will send a single message of fixed size (2 bytes) any way. The shorter the radio range, the more the number of hops which requires HIU to send more bytes. Figure 3(e) shows that increasing the radio range makes HIU's total cost less than TAG's total cost after 3 rounds only.

4.4 *Median* queries

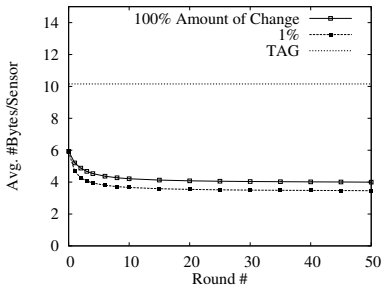
Based on [8], TAG suggests collecting all values to the base station in order to compute an exact answer. Overall, Figure 4 shows that collecting all values to compute an exact *Median* cannot outperform HIU even in the first round. The cost of the *Median* query using HIU is twofold: histogram's cost and values collection's cost. In the first round, HIU does not reuse any existing information. However, in the consecutive rounds, HIU reduces number of bytes sent by each sensor per round to construct the *Histogram*. Typically, HIU requests a smaller number of values to be collected than TAG. Instead of sending the collection query to all nodes in the WSN, the cached information (in each node) is used to direct the query only to relevant nodes as we explained earlier in Section 3.0.2.

In Figures 4(a)and 4(b) we show the influence of changing ρ and δ which affect *Histogram*'s cost only with no influence on the values collection's cost. The smaller the ρ, the cheaper the *Histogram* and then the cheaper the Median. Figure 4(b) confirms our earlier finding that δ does not have a significant influence on the query cost. On the other hand, the smaller the ρ, the less expensive the *Histogram* and then less expensive is the *Median*.

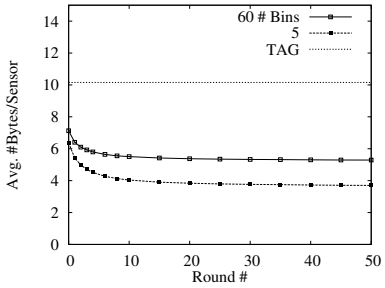
In Figure 4(c), changing the *Histogram* size affect both histogram's cost and values collection's cost. The less number of bins in the histogram, the cheaper the *Histogram* cost but the more values required in the values collection phase. However, the overall query cost is



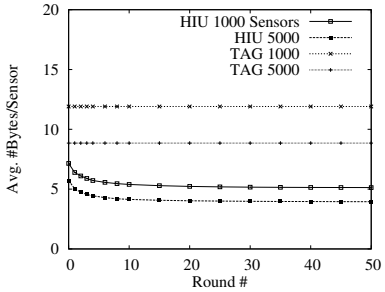
(a) Probability a value changes (ρ)



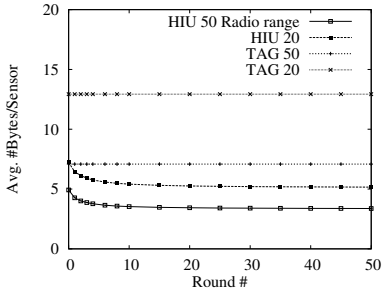
(b) Average amount of change per value (δ)



(c) Histogram Size (B)



(d) Number of Sensors (N)



(e) Radio Range (R)

Fig. 4. Cost of running exact-Median query

reduced for two reasons: (1) All nodes are guaranteed to participate in the *Histogram* construction and then guaranteed to pay its cost. (2) Values collection phase uses the local histogram in each node which makes the cost of this phase is fairly small comparing to *Histogram* construction phase.

Figures 4(d) and 4(e) shows the influence of number of sensors in the area (N) and sensor's radio range (R) on the *Median* query cost. These two parameters affect the TAG cost as well as the HIU cost because they change the logical routing tree. However, TAG cost is fixed regardless to number of rounds. Figure 4(d) shows that increasing the network density (Number of sensors / area) decrease the average cost per sensor for both TAG and HIU. HIU becomes cheaper because the more sensors connected to a parent the more efficient aggregation can be done and better query guidance can be provided while collecting values in the chosen bin. TAG is also becomes less expensive because the cost of intermediate sensors will be shared with more leaf sensors and also because sensors can find a shorter route to the base station. In Figure 4(e), the larger the sensor range, the smaller the number of hops to reach the base station and then the smaller number of bytes sent for both TAG and HIU.

5 Related Work

There has been not much work done in the literature to construct a *Histogram* of WSN values since Madden et. al. proposed TAG algorithm in 2002 [8]. Chow et.al. proposed an algorithm to construct a spatio-temporal histogram [2]. The main idea is to construct an approximate spatio *Histogram* that is updated with every time any sensor reading reaches the base station. This approximate *Histogram* is used for location monitoring. The authors proposed a basic and adaptive approach to construct an approximate *Histogram* in the base station. The main idea is collecting values at the base station and then construct the histogram. The energy saving comes from constructing an efficient approximate *Histogram* instead of an exact one. Since our algorithm construct an exact *Histogram* using an in-network algorithm and we don't require all values to be sent to the base station, we did not compare their approach with HIU. However, their innovative algorithms for location monitoring can work on top of our exact histogram and still save sensors' energy.

On the other hand, there are a few algorithms proposed in the literature to answer complex aggregate queries like Median. Algorithms proposing approximate answer for Median have a bound on the median's rank not the median's value (e.g. [14], [13]). A median value is supposed to be exactly in the middle of a sorted list with a rank of $\frac{N}{2}$, where N is number of values in the list. They ensure the rank of an approximate answer is $\frac{N}{2} \pm \varepsilon$.

A median value supposed to be the one in the middle of an ordered list if the number of values is odd or the average of two values with ranks of $\frac{N}{2}$ and $\frac{N+1}{2}$. Our proposed algorithm to calculate an approximate median does not guarantee a bound on the median's rank but on the median's value. We ensure that the approximate answer of a Median query cannot be worse than $\varepsilon = \frac{Ub_m - Lb_m}{2} \mid Ub_m$ and Lb_m are the upper and lower bounds of the bin contains a median value.

In [14], Shrivastava et. al. proposed a data structure called query digest (Qdigest). The Qdigest data structure is very similar to an equi-depth *Histogram* but its ranges can overlap with each other. The authors propose using Qdigest to compute an approximate median with an error bound ε. HIU ensures the error of the approximate median value is (itself) bounded, regardless of its rank in the values' list.

6 Conclusion

In this paper we proposed a new algorithm (HIU) that uses in-network aggregation and in-node caching to reduce the energy consumption for constructing a *Histogram* query. Obtaining a *Histogram* in the base station helps in computing bounded approximate answers for other aggregate queries including *Median*. Moreover, we proposed algorithms that use HIU to compute exact answers for these aggregate queries.

HIU outperforms the TAG algorithm (current state-of-the-art to answer a *Histogram* query) in the synthetic and real datasets. On average, HIU multiplies the network lifetime about three times.

HIU outperforms TAG's algorithm to compute *Max* if the amount and/or probability of values changes are low. Figures show that a small *Histogram* size can compute an exact answer for *Max* cheaper than TAG. HIU also outperforms TAG in terms of amortized number of bytes sent in the network when computing an exact *Median* regardless of the change probability or *Histogram* size.

7 Future Work

Despite the importance of the *Histogram* query in industry, we find a potential for *Histogram* to help computing an exact answer for Median. In our future work we would like to propose a cost model that helps computing an exact answer for a *Median* query with the optimum cost. This answer requires (according to HIU) to build a *Histogram* first. Instead of building a *Histogram* once, we can build it in several iterations. For example, if it is required to have a *Histogram* size of 100, we can compute it in two steps, first a *Histogram* with size 10, then send another *Histogram* query for the median bin only of size 10. The cost of two *Histogram* queries of size 10 is much cheaper than cost of one *Histogram* with size 100. Another alternative is to use GetValues function to collect all values in a specific bin. The cost model will minimize the cost by choosing the optimum method in each bin during each iteration.

In our work, we assume that network communication between sensors is perfect with no losses. In real world application, this is not possible. In node caching can be useful to help reduce the impact of any network failure. How to make HIU able to reduce the impact of any network failure is an interesting question that we would like to answer in the future.

REFERENCES

[1] M. Burl, B. Sisk, T. Vaid, and N. Lewis. Classification performance of carbon black-polymer composite vapor detector arrays as a function of array size and detector composition. *Sensors and Actuators B: Chemical*, 87(1):130 – 149, 2002.

[2] C.Y. Chow, M.F. Mokbel, and T. He. Aggregate location monitoring for wireless sensor networks: A histogram-based approach. In *Proc. of MDM*, pages 82–91, 2009.

[3] S. Collins et al. New opportunities in ecological sensing using wireless sensor networks. *Frontiers in Ecology and the Environment*, 4(8):402–407, 2006.

[4] A. Coman, J. Sander, and M.A. Nascimento. Adaptive processing of historical spatial range queries in peer-to-peer sensor networks. *Distrib. Parallel Databases J.*, 22(2):133–163, 2007.

[5] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. *Proc. of ACM SIGMOD*, pages 157–166, 1993.

[6] Intel Berkeley Research Lab. Intel lab data http://www.select.cs.cmu.edu/data/labapp3/index.html.

[7] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. *Proc. of IPSN*, pages 254–263, 2007.

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

[9] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. *Proc. of ACM SIGMOD*, pages 491–502, 2003.

[10] B. Malhotra, M.A. Nascimento, and I. Nikolaidis. Exact top-k queries in wireless sensor networks. *IEEE TKDE*, (To appear), 2010.

[11] E.D. Pinedo-Frausto and J.A. Garcia-Macias. An experimental analysis of zigbee networks. In *Proc. of IEEE LCN*, pages 723 –729, 2008.

[12] S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed compression in a dense microsensor network. *IEEE Signal Processing Magazine*, 19(2):51 –60, 2002.

[13] S. Roy, M. Conti, S. Setia, and S. Jajodia. Securely computing an approximate median in wireless sensor networks. *Proc. of SecureComm*, (6):1–10, 2008.

[14] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. *Proc. of SenSys*, pages 239–249, 2004.

[15] J. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, and A. Wood. Wireless sensor networks for in-home healthcare: Potential and challenges. *Proc. of HCMDSS*, 2005.

[16] B. Ying, W. Liu, Y. Liu, H. Yang, and H. Wang. Energy-efficient node-level compression arbitration for wireless sensor networks. In *Proc. of ICACT*, pages 564 –568, 2009.