And of knowledge, you (mankind) have been given only a little.

– Quran, chapter 17, verse 85

University of Alberta

HISTOGRAM AND MEDIAN QUERIES IN WIRELESS SENSOR NETWORKS

by

Khaled A. Ammar

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2011
Edmonton, Alberta

Examining Committee

Mario A. Nascimento, Computing Science

Benoit Rivard, Earth and Atmospheric Sciences

Janelle Harms, Computing Science

To my Father, Mother and my wife Doaa...

To my angels Hamza and Razan...

For making me a better me.

Abstract

Recently, Wireless Sensor Networks (WSNs) have been used in many monitoring applications, e.g., environment monitoring. A WSN consists of a set of nodes, each having one or more sensors to measure a phenomena. Nodes are connected to each other using wireless radio communications. Typically, there is at least one base station that functions as an interface between the monitored area and the end user.

In many applications, users are interested in statistical summaries of the observed data, e.g., histograms reflecting the distribution of the collected values. In this thesis we propose two main contributions: (1) an efficient algorithm for answering *Histogram* queries in a WSN, and (2) efficient algorithms for answering snapshot and continuous *Median* queries in a WSN.

While designing applications for the WSN, the main challenge is the battery life time. For WSN nodes, sending a message consumes a significantly higher amount of energy than processing information inside the node. Therefore, we design our proposed algorithms in order to minimize the energy consumption and/or transmission cost, i.e. number of sent bytes, and consequently extend its lifespan. Our experimental results, using synthetic and real datasets, show that our proposed solutions are indeed able to substantially extend the lifespan of the WSN when compared to previously proposed solutions.

Acknowledgements

I thank Allah, the only true god and the most merciful, for his non-countable blessings in my life. I praise him for giving me the energy and enthusiasm to complete this piece of work and I hope he allowed me to be a better person and a better researcher by helping my community and the whole population in this world.

”Whosoever does not thank people,has not thanked Allah”, Prophet Mohammed PBUH. I would like to thank my wife, Doaa, for her great continuous support during my master degree. She did every thing possible to empower me and inspire me to finish this thesis. My mother and father are the most important people to me in this life. I owe them every single success or happiness moment I live. Finally, I thank my sisters, my brother, my son and my daughter. You all made me a happy and strong person because I know you are there to save my back.

I’m grateful for the support Prof. Mario A. Nascimento offered me. I deeply appreciate his patient on developing my research, writing, and presentation skills. Mario, you are the best supervisor ever. You did not just support my thesis, you actually supported my life here in the University of Alberta, Thank you.

While being in the beautiful Edmonton, I was pleased by many friends who supported myself and my family. I spent long times with Mohamed Shazly discussing ideas during very early stages of my research. Mohammed Saber helped me taking care of my son Hamza allowing me to spend longer times working on my thesis. Ehab Hamza was very kind and supportive during the last few days before my defense. I would use many pages to mention all my friends. Thank you all for allowing me to experience the best community ever.

Finally, I would like to thank my committee; Prof. Rivard and Prof. Harms for their positive feedback and helpful comments.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

WSNs	Wireless Sensor Networks
HIU	Histogram Incremental Updates algorithm
RBM	Refinement Based Median algorithm
CRBM	Continuous Refinement Based Median algorithm
CPU	Central Processing Unit
SPT	Shortest Path Tree
RBM*	Using the RBM several times to answer a Continuous *Median* query
DST	Dominating Set Tree
BISPT	BIased Shortest Path Tree

List of Symbols

S	A set of all nodes in the WSN
N	Number of nodes in the WSN S
s_i	A node in the WSN S
v_i	The value for a node s_i
$\tilde{v_i}$	The cached value for a node s_i from the previous round
Q	An aggregate query
$epoch$	The frequency of running a continuous query.
H	A histogram
B	Number of bins in a histogram
b_i	A bin number i in a histogram
h_i	Number of values in bin number i in the histogram
Lb	The lower bound for a histogram
Lb_i	The lower bound for bin b_i in the histogram
Ub	The upper bound for a histogram
Ub_i	The upper bound for bin b_i in the histogram
I_i	The initial state of a node s_i. (It is represented by $p_i = (i, h_i)$)
P_j	The partial state, also known as the local histogram, for a node s_j. (It is represented by an array of p_is)
M	An array of all messages received by the children of a node s_j
U_j	An update message sent from node s_j to its parent
E_j	Extra information attached with the node s_j's message to its parent to represent an answer for an aggregate query
K	The index of the value answering a quantile query, assuming all values are sorted
R	The radio range in meters for all nodes in the WSN
δ	The average amount of change in a sensor's value in each round. It is represented as a percentage of the maximum possible value measured by a sensor in the WSN
ρ	The probability that a sensor's value change in each round
σ	Assuming initial values for all nodes are following the normal distribution, the is the standard deviation for that distribution.
H_s	The header size in each message sent by a node in the WSN.
E_t	The energy consumed by sending a message.
E_r	The energy consumed by receiving a message.
s	The setup cost, in terms of energy, to send a message.
t	The cost of sending one bit for 1 meter. It is measured in Joule.
r	The lower bound for a histogram.
d	The distance a message can reach, in meters.
b	The message size in bits.

Chapter 1

Introduction

1.1 Introduction to WSNs

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. In the context of this thesis we will refer to sensor nodes simply as nodes. The fact that a node may contain more than one sensor and then produce more than one value is irrelevant for our purpose. We assume that 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 environment monitoring [3, 11, 27]. 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 the answer back to the user. Typically, WSN nodes have limited resources in terms of energy, CPU, memory and the battery is considered the most important resource in a WSN node. The required energy for transmission is significantly higher than the required energy 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 [18]. For this reason, it is very important that all algorithms executed on top of a WSN minimize transmissions.

For the purpose of our discussions in this thesis, we assume that the WSN has N nodes and it can be modeled as a graph $G = (S, E)$, where S is the set of

vertexes that represents all N nodes; E is the set of edges that represents existing communication links between any two nodes in the network. We assume that all nodes have the same radio model, thus the same radio range, denoted as R, and thus, if $d(s_i, s_j)$ denotes the Euclidean distance between two nodes s_i and s_j in S, we have $E = \{\langle s_i, s_j \rangle \mid d(s_i, s_j) \leq R$, and $s_i, s_j \in S$, and $s_i \neq s_j\}$. Each node $s_i \in S$, except the base station, periodically measures a value v_i. In fact, every value has an associated timestamp, however in order to lighten the notation, and considering that we process snapshot queries, we do not explicitly denote it unless necessary. We assume all nodes are connected to the base station, likely through a multi-edge path. In the remainder of our discussion we assume the distance between two nodes is the minimum number of hops, i.e. edges, between them. The graph G represents the physical network. We assume the existence of a routing tree $T = (S, E') \mid E' \subset E$, connecting all nodes in S and where the base station is T's root. Typically, this tree is constructed to minimize the number of hops between a node and the base station. Finally, we assume that the connection between nodes is reliable (i.e, there is no link failure), and concentrate on the query processing aspect of the problem.

1.2 Motivation

Simple aggregate queries such as *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. Complex aggregates, where the answer size is not fixed and depends on number/distribution of all values, for instance *Quantile* and *Histogram* provide more insight about the data and are mandatory for many applications. 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 [11]. Petrochemical engineers can 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).

Quantile queries, as known as order statistics queries, can provide a better characterization of the values' distribution than simple aggregate queries such as *Max*, *Min*, or *Average*. Additionally, they are more robust to outliers, which are common in sensor networks due to failures, poor calibration, or interference from the environment. For example, a single reading from a faulty sensor can significantly change the average reading value, but order statistics such as the median, 95th percentile and 5th percentile are resilient to these failures [6].

The database community classifies aggregation functions into four main categories: distributive (*Max*, *Min*, *Sum*, *Count*), algebraic (*Average*), content sensitive (*Histogram*), and holistic (*Quantile*) [13, 17]. Computing distributive and algebraic functions on a set of distributed nodes on WSN can be easily achieved using the in-network aggregation concept proposed in [17]. Although the authors in [17] proposed an algorithm that uses an in-network algorithm to reduce number of required transmissions to compute a *Histogram* query, this query has not received much attention in the related literature since then. Also, it is known that holistic functions need all the candidate values to be centralized at one node [13, 17]. Kuhn et al. [13] shows that distributive and algebraic functions could be calculated for a general network of N nodes and diameter D in $O(D)$ time while holistic has a lower bound of $\omega(D \times log_D N)$. Part of our work is to propose efficient distributed algorithms to answer a *Quantile* query, but focusing on *Median* without loss of generality, using *Histogram* answers.

1.2.1 Histogram Queries

The *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 are adjacent, consecutive, non-overlapping and often are chosen to be of the same size. The *Histogram* query is also useful for estimating the values distribution and then for computing approximate answers for other aggregate queries. We discuss this in detail in Chapter 2.

A *Histogram* query is denoted as $Q(Lb, Ub, b_1, b_2, b_3, ...b_B, epoch)$, where: $epoch$ is 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 where:

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

The answer for a *Histogram* query is denoted as $H = (h_1, h_2, ..., h_B)$, 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.

1.2.2 Median Queries

The *Median* query is a special case of the quantile query. Although we will use *Median* as an example in this thesis, the same algorithms can be easily extended for any quantile query. A quantile query looks for the K^{th} value in a list of N values. In case of the *Median* query, $K = \lceil \frac{N}{2} \rceil$. Because any value in the WSN is a candidate to be the median query answer, collecting all candidates at the base station is a straightforward approach to answer such a *Median* query. However, earlier proposals in the literature suggested decreasing the number of candidates through the use of histograms. If nodes' values follow a uniform distribution then a *Histogram* query of B bins will cluster them in B clusters and the size of each cluster is about $\frac{N}{B}$. The correct answer for a median query cannot be in two clusters in the same time. The number of candidates will drop from N to N/B. We can repeat such a query to refine the list of candidate values until the number of candidates is very small and could be retrieved easily to the base station.

1.3 Contribution

We present the following contributions in this thesis. First, we propose an efficient distributed algorithm to answer *Histogram* queries. This algorithm requires

less than half the amount of energy used by the classical TAG algorithm [17] to construct a histogram thus, it can at least double the network lifetime. This algorithm outperforms the TAG algorithm (current state-of-the-art to answer a Histogram query) multiplying the network lifetime, on average, about three times.

Our second contribution is a set of algorithms to find an approximate and exact answers for other aggregate queries. The Approximate answers can be obtained with no overhead and with a bounded accuracy directly related to the *Histogram* query's structure, e.g., number of bins and their ranges. Exact answers for some aggregate queries could be computed with a very small overhead.

Then, we explore how to use a histogram in the base station to compute a *Median* query. We proposed three ways to do that: (1) Compute an approximate answer with no overhead, (2) Use a *Histogram* query once to reduce the number of candidate values while computing an exact answer, and (3) Use multiple Histogram queries to significantly reduce the cost of a Median query. In particular, we explore how to minimize the cost of all *Histogram* queries that refine the list of candidate values and then minimize the cost of the Median query. To do that we address two questions: (1) How many refinements are required before retrieving all candidates to the base station? (2) What is the histogram structure of each refinement query?

To answer the first question we show that a central algorithm is required but not practical for the WSN context, thus we use a heuristic distributed algorithm. Answering the second question is more challenging because it includes three dimensions: using a single histogram size in all refinements vs. multiple histogram sizes, determining the size of each Histogram, and using equal size bins or using bins with different sizes. We propose two approaches to answer the second question with all of its dimensions. Using the average amount of bytes sent as our performance indicator, our experiments show that adapting each *Histogram* query based on available information in the base station minimizes the number of bytes sent, on average. Because the energy consumption per-node depends mainly on the amount of bytes each sensor sends, minimizing the number of bytes sent also minimizes the power consumption and then increases the network life span.

Finally, we extend the median algorithm to process the Continuous *Median*

queries.

1.4 Thesis Structure

This thesis is organized as follows: Chapter 2 reviews the classical TAG approach that answers a *Histogram* query and explains in detail our proposed algorithm, HIU. The same chapter discusses the computation of approximate as well as exact answers for other aggregate queries using a *Histogram* as a starting point. In Chapter 3, we present algorithms to answer snapshot and continuous *Median* queries in the WSNs context using a *Histogram* query. Finally, Chapter 4 summarize our contributions and proposes directions for future research.

Chapter 2

Histogram Queries in WSNs

2.1 Introduction

A straightforward technique to answer a continuous *Histogram* query in the WSN context is to periodically gather all values from all nodes at the base station and then build a histogram. The classical TAG algorithm decreases the number of required messages extensively compared to the straightforward technique [17]. 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 [17]. 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 every time a 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 algorithm collects values at the base station and then constructs the histogram. The energy saving comes from efficiently constructing approximate histogram instead of an exact one.

In [17], the authors define the TAG model to answer aggregate queries using an in-network algorithm in the WSN context. Each node has initial and partial states, to be defined later. 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 the root. This process continues

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

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 the formats of the initial and partial states in this case are not discussed in details in [17], 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 node s_j's children. In general, as we will see later in Section 2.1.1, a message in M can 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.

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

```
1: l ← L {L is number of levels in tree T}
2: while l ≥ 0 do {Iterate on all T's levels}
3:    for each sensor s_j with a value v_j in level l do
4:       I_j ← (i, 1) | i ← arg{b_i | v_j ∈ b_i} {Initial state for s_j}
5:       P_j ← 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 ← l − 1 {move to the upper level}
8: return P_j in T's root
```

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.

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 ← 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) | i ← arg{b_i | v_m ∈ b_i}
10:      else {the message m is a set of pairs}
11:         P_m ← all pairs in the message m
12:      for each pair p_i = (i, h_i) ∈ P_m do
13:         if ∃ p_k = (k, h_k) ∈ R | k = i then
14:            h_k ← h_k + h_i
15:         else
16:            Copy p_i from P_m to R
17:   return 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 $zero$.

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 (in bits) 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 [17] 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 to use in-node caching

and send incremental updates instead of actual values.

2.1.1 Histogram Incremental Updates (HIU) Algorithm

A node 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 partial state every epoch, a node can build an update message based on the previous round's partial state. This idea was used in several algorithms in the database's literature. For example, in [9] the authors proposed algorithms to maintain materialized views incrementally. In our algorithm, nodes 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. Note that in the TAG algorithms, all nodes send their partial states to their parents regardless of how these partial states differ from ones in the previous round.

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 we assume $\tilde{P_j} = \{0, 0, 0, ..., 0\}$.

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.

Nodes do not always send their partial states to their parents. A leaf node sends its partial state only if the new value, generated in the current round, 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, $\Sigma_{i=1}^{B} h_i = 0$.

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 $\tilde{b}$ is undefined **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 $Agg \neq$ 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

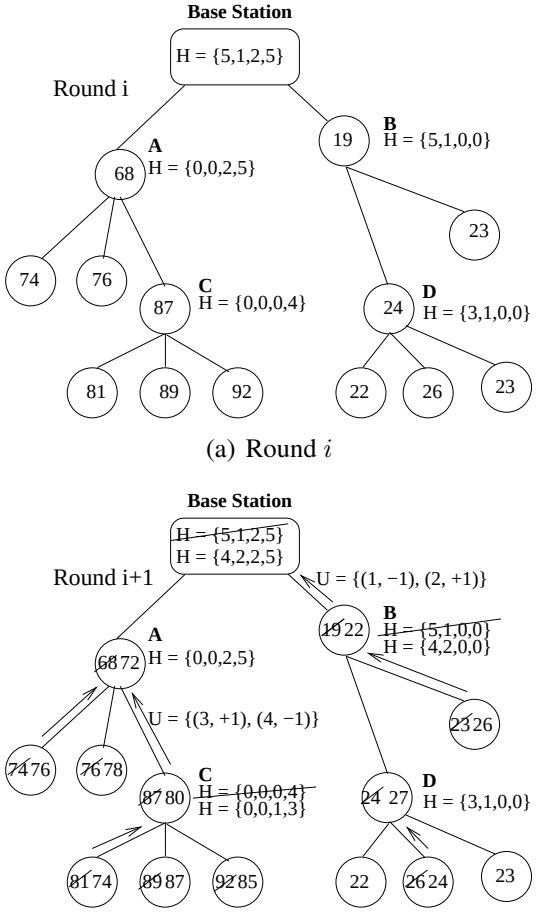
Received update-messages in any non-leaf node may cancel each other in which case nothing is sent forward. For example, consider non-leaf node C that has two subtrees, A and B. Subtree A has x nodes where their value moved from bin b_k to b_l. On the other hand, the subtree B has $x-1$ nodes where their 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 the number of bits, is chosen. A more complex compression can be implemented for this function, e.g., [24, 32]. However, a detailed discussion about compression algorithms in WSN is beyond the scope of this thesis.

In figure 2.1, we show an example for the HIU algorithm. The *Histogram* query is Q(0,100,4). In the Round $i+1$, only leaf nodes who change their bins send a message to their parents. Node **D** receives a value update from one of its children but its own value was changed also. However, the partial state, local histogram of its subtree, did not change and then it does not need to send any message to its parent. Node **C** receives one update from one of its children. Although its own value changed, its histogram bin did not. Node **C** should send a message to its parent to describe the change happens in its partial state. Instead of sending the whole histogram again, it only sends two pairs of (bin id, change in the count). The whole histogram would cost 8 bytes but the two pairs costs only 6 bytes.

Node **A**, received an update message from Node **C** but the bins of its own value and one of its leaf node children were changed as well but on the opposite side. So that, the partial state did not change and then it does not have to send any message. Node **B**, received only one update from its children. Its value did not change the histogram bin so that the change of its children was the only influence on its partial state. It communicates this change to its parent as a pair of updates because it is cheaper than sending the whole histogram.

Once the histogram receives the update message from Node **B**, it applies it to



(a) Round i

(b) Round $i + 1$

Figure 2.1: HIU example showing round i and $i + 1$

its partial state by decrease the count in the first bin by 1 and increase the count of the second bin by 1. The partial state of the base station node is the query answer.

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 some other aggregate queries in the base station.

2.2 Answering Other Aggregate Queries

A histogram provides a broad picture for values in the WSN and may be used as a starting point for more statistical analysis. Occasionally, a user might like to know

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	$\frac{Ub_m+Lb_m}{2} \mid h_m \neq 0, \forall i < m, h_i = 0$	
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 2.1: A summary for supported aggregate queries

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. Since the main target of this thesis is the Median queries, we discuss it in details separately in Chapter 3.

Recall from Section 1.2.1 that 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 is the count of values within the range of Lb_i and Ub_i. Table 2.1 shows how to compute approximate answers for some aggregate queries in the base station using a *Histogram* query's result. Because, all computations are made at the base station, there is absolutely no overhead on the WSN.

Table 2.1 also shows the error margin limit for each approximate aggregate answer. A *Histogram* query can provide an exact answer for *Count* aggregate query. All error bounds 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* query's result because it increases the number of bytes sent.

Next we propose algorithms to compute exact answers for different types of aggregate queries. We can obtain exact answers by adding some overhead to the HIU messages but not extra messages.

Exact answers using per Message overhead

Communication devices in some WSN mandate the sensor to send messages of fixed size only [23]. In this case, sending less information will not decrease the energy consumption because all buckets should have the same size. For example, a communication device that sends a packet of fixed size, 128 bytes, will send the same packet even if the algorithm requires it to send 10 bytes only. The remaining 118 bytes are considered idle and not useful. These idle bytes can be used to send the extra information, at 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, the ExtraInformation function (called at line 20 in Algorithm 3) collects required information and aggregate them to facilitate computing the exact answer in the base station. Note that, if an exact answer is required, then leaf nodes values should be sent every round.

The behavior of the 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 simultaneously.

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. 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.

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 the difference defined as $SUM - \Sigma_{i=1}^{B} \frac{Ub_i + Lb_i}{2} \times h_i$. Here, 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 $log_2(\Sigma_{i=1}^{B} \frac{Ub_i - Lb_i}{2} \times h_i)$.

2.3 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, (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 requires 3 bytes, 1 for the bin id and 2 for the bin count.

We investigate our algorithms with respect to two datasets (a 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 a 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 maximum depth of the logical tree. Varying the Histogram size shows that our algorithm does not have any limitation on the histogram size and can efficiently work regardless of this parameter. Studying δ and ρ shows the sensitivity of our algo-

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

1: $E = 0$ {E is the returned value to allow computing the exact answer for an aggregate query Agg}
2: **if** $Agg =$ *Max* **then**
3: $MAX = v_j$ {v_j is the current value of sensor s_j}
4: **for** each message m sent to a sensor s_j **do**
5: **if** m is a value v_m **then** {m was sent by a leaf node}
6: $temp \leftarrow v_m$
7: **else** {m was sent by an intermediate node}
8: $temp \leftarrow E_m$
9: {E_m is a value in m represents the max of sender's sub tree.}
10: **if** $MAX < temp$ **then**
11: $MAX \leftarrow temp$
12: $E \leftarrow MAX$
13: {The code for *Min* is very similar to *Max* and omitted from this code}
14: **if** $Agg =$*Sum* or $Agg =$*Average* **then**
15: $SUM = v_j$
16: **for** each message m sent to a sensor s_j **do**
17: **if** m is a value v_m **then** {m was sent by a leaf node}
18: $SUM \leftarrow SUM + v_m$
19: **else** {m was sent by an intermediate node}
20: $SUM \leftarrow SUM + E_m$
21: {E_m is a value in m represents the sum of sender's sub tree.}
22: $E \leftarrow SUM$ {This is also sufficient for *Average* because *Count* is known}
23: **return** E

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 nodes)	$1000, 2000, \mathbf{3000}, 4000, 5000$

Table 2.2: Studied parameters for HIU analysis and their values (default values in **bold**)

rithm against the behavior of the values in the WSN monitored area. 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 nodes N, shows the algorithm scalability from the point of view of the WSN density.

Table 2.2 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 figures show the average values obtained over 15 runs. During each run, the sensor locations are randomly distributed and the base station is randomly selected among one of the nodes. In order to ensure a fair comparison, both TAG and HIU use the same setup.

Our synthetic dataset consists of N nodes uniformly distributed in an area of $200m \times 200m$. The values of all nodes use 2 bytes only and 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 likely to be negative or positive). The exponential random variable was chosen to allow mostly small and eventually very large changes. The average of the exponential random variable is $\delta\%$ of 2^{16}. We assume that all nodes 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 the Intel Berkeley Research Lab [10]. 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 not changeable at 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] :

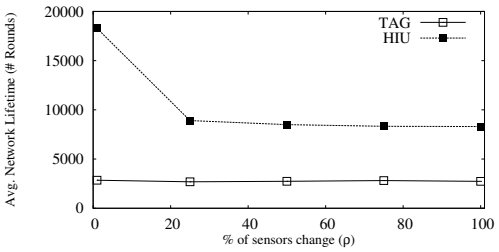
$$E_t = S + t \times b \times d^2 \tag{2.1}$$

$$E_r = r \times b \tag{2.2}$$

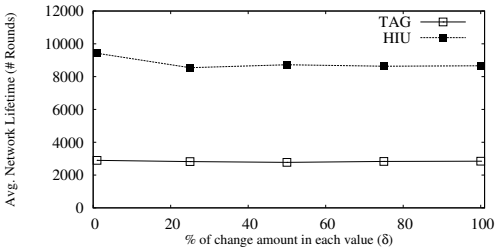
where $S = 50$ nJ is the setup cost to send any message, $t = 10$ pJ and $r = 50$ nJ are the required amount of energy to send or receive one bit for one meter, respectively. The message size in bits is b, while the Euclidean distance (in meters) between the sender and the receiver is d.

2.3.1 Performance Evaluation for *Histogram* and Approximate Aggregate Queries

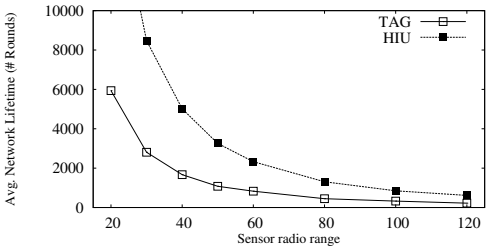
Figure 2.2 shows the HIU and TAG performance when changing each of the studied parameters using the Synthetic dataset. Because the TAG performance does not depend on changes in nodes' values, a network using TAG algorithm died after about 2700 rounds regardless of the change probability (ρ) or amount of change per round (δ). Figures 2.2(a) and 2.2(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), the 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 are not communicated as they cancel each other in the early stages of the routing tree.



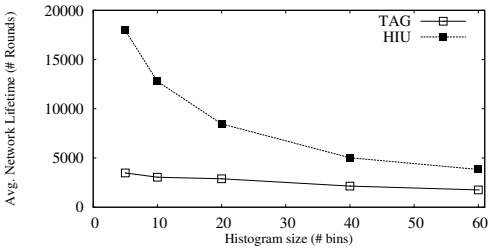
(a) Probability a value changes (ρ)



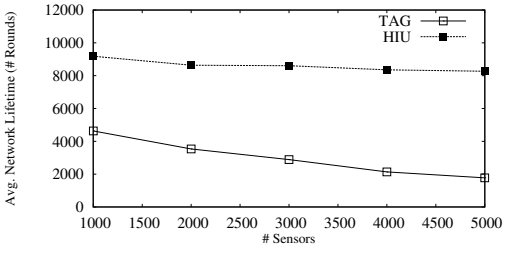
(b) Average amount of change per value (δ)



(c) Radio range (R)



(d) Histogram size (B)



(e) Number of Nodes (N)

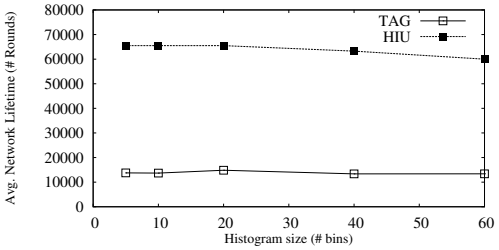
Figure 2.2: Network Lifetime analysis using the Synthetic dataset

Figure 2.2(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 in Equation 2.1 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 are better when the radio range is smaller, HIU multiplies the network lifetime three or four times compared to TAG.

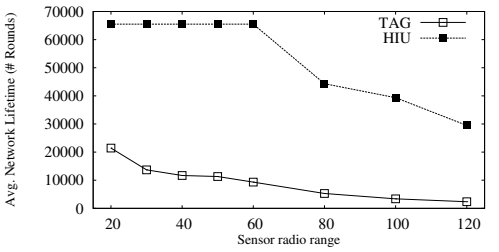
Figure 2.2(d) is evidence that HIU can still multiply the network lifetime by at least a factor of two as number of bins increases. A larger number of bins means a higher probability that the number of changed bins gets higher and then HIU performs worse. However, TAG requires all intermediate nodes to send their partial state regardless of number of nodes, i.e., TAG also performs worse, though not as noticeably when increasing number of bins.

Figure 2.2(e) shows that HIU can scale efficiently and handle WSNs with large numbers of WSN nodes better than TAG. HIU has the same performance regardless of the number of nodes in the field. We basically increase the network density in the field by increasing number of nodes while using exactly the same area. TAG's performance decreased because the more nodes 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 the HIU performance. The key factor is how the frequency of change in values.

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 so 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 2.3



(a) Histogram size (B)



(b) Radio range (R)

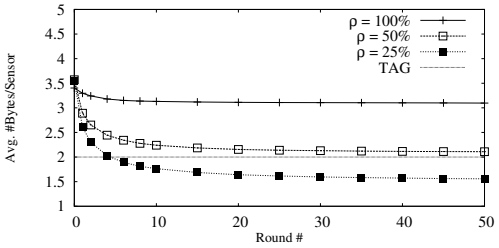
Figure 2.3: Network Lifetime analysis using the Intel Berkeley dataset

shows that HIU allows the network to last significantly longer than TAG (sometimes by a factor of 10). In the real dataset, there is a limitation on the number of rounds because the dataset has about 65,000 rounds only. HIU curves in Figure 2.3 reach the upper limit for number of rounds but none of the network's nodes dies.

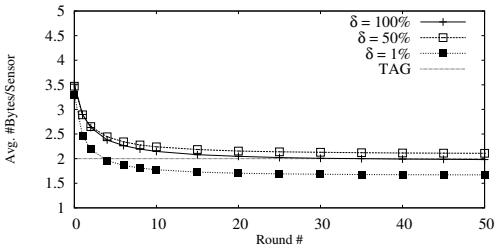
2.3.2 Performance Evaluation for Exact answers

Regardless of the algorithm used to construct a histogram in the base station, a histogram allows the computation of approximate answers for several other aggregate queries without any overhead as discussed in Section 2.2. The base station can also compute the exact answer for an aggregate query by using the HIU algorithm with some overhead, i.e., larger messages. In the following discussion, we will use the *Max* query as an example. However, the same discussion applies to *Max*, *Min*, *Average*, and *SUM* queries.

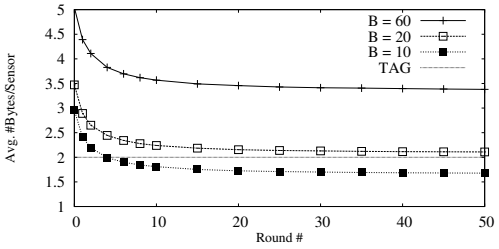
Because the main aim of our experiment is to study the HIU overhead cost for computing an exact answer, we use the amortized average amount of bytes sent per sensor per round as our performance indicator. Every round, the total number of sent bytes from all nodes during all previous rounds are calculated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



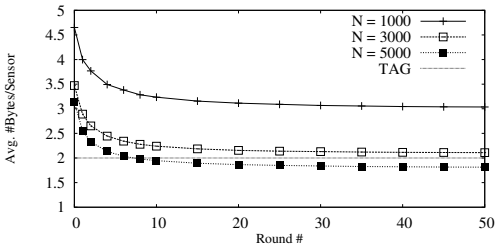
(a) Probability a value changes (ρ)



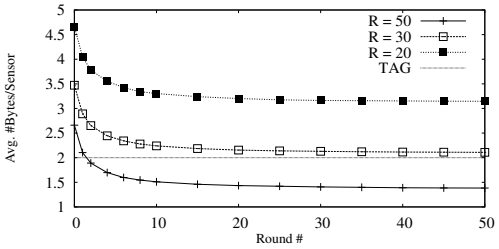
(b) Average amount of change per value (δ)



(c) Histogram Size (B)



(d) Number of Nodes (N)



(e) Radio Range (R)

Figure 2.4: Cost of running exact-Max query

Based on [17], every sensor should send exactly 2 bytes to collect the maximum value using TAG. HIU collects the *Max* information while constructing the histogram. Recall that the first round in HIU consumes a large amount of energy due to sending the largest amount of bytes compared to other rounds because there is no cached information. 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 amortized number of sent bytes per round is decreased and eventually reaches a steady state.

Figure 2.4 shows the amortization analysis for the TAG and HIU algorithms in computing the exact *Max* using five parameters: ρ, δ, B, N, and R. While testing one parameter, other parameters are assumed to have their default values (Table 2.2). The main goal is to show that HIU can outperform TAG in terms of the average cost in the long run. We use the synthetic dataset only to have more control on the experiment. Figure 2.3 also shows that HIU performs better for the real dataset.

Figure 2.4(a) shows the influence of the 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 lead to a smaller HIU cost but TAG's performance stays 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 in the long term only if nodes change their values infrequently ($\rho \leq 40\%$).

In Figure 2.4(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 outperforms TAG. Recall that a sensor can sense a specific range of values. If the value is bigger than the 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 nodes 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, the influence of changing ρ has more influence on changing δ. The same figure shows that changing δ does not have a large impact on the performance curves. The only exception is the $\delta = 1\%$ curve because changing a sensor value by 1% on average will unlikely change its bin in the histogram, if the bin's width is reasonably large, and then is unlikely to cause a sensor to send any data.

The 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 gets worse with an increase in the bin size, the HIU algorithm performs better while computing exact *Max*. Figure 2.4(c) shows that decreasing number of bins can make HIU outperform TAG very early even if the probability of change and amount of change are both 50%. Recall that TAG outperforms HIU in Figures 2.4(a) and 2.4(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.

The Network density depends on the number of nodes (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 nodes available in the area the more opportunities to save and decrease the amount of sent messages. Figure 2.4(d) shows that increasing the network density more than 3000 nodes in 200×200 area (0.075 sensor/m^2), makes the HIU outperform 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 nodes 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). The shorter the radio range, the more hops which

requires HIU to send more bytes but at a smaller cost as discussed earlier. Figure 2.4(e) shows that increasing the radio range makes HIU's total cost less than TAG's total cost after 3 rounds only.

2.4 Conclusion

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

HIU outperforms the TAG algorithm in the synthetic and real datasets. On average, HIU multiplies the network lifetime about three times. HIU also outperforms TAG's algorithm to answer the *Max* query if the amount and/or probability of value changes are small. Figures show that a small Histogram size can compute an exact answer for a *Max* query cheaper than TAG.

Despite the importance of the *Histogram* query, we find a potential for histograms to help compute an exact answer for *Median* queries. In the next chapter we discuss that in detail.

Chapter 3

Median Queries in WSNs

3.1 Introduction

A number of papers concerning algorithms for processing typical aggregate queries, e.g., *Max* and *Top-k*, within a wireless sensor network have been published in recent years [19, 28]. However, relatively few have addressed *Median* queries. In this chapter we propose algorithms to process snapshot and continuous *Median* queries and to compute an accurate median answer. These algorithms are based on a series of refinement queries. Each refinement query is a *Histogram* query, with the aim of continuously refining the range where the actual median value resides. Because the cost of a *Histogram* query depends mostly on the structure of the histogram itself, we aim at optimizing each *Histogram* query, hence optimizing the overall cost of the *Median* query.

3.2 Related Work

There are several algorithms for finding quantiles in databases, data streams and distributed data systems. In this section we cover only approaches that are applicable to WSNs. Algorithms for quantile queries, of which the median is a special case, can be partitioned into three categories: exact, approximate and probabilistic. Approximate algorithms return a quantile estimate within a user-defined error bounds. The error bound can be defined in terms of a given rank-distance to the actual quantile's rank or in terms of an error in the actual returned value. Probabilistic algorithms provides approximate answers with a given probability. Those

algorithms return a value whose rank is within a given error bound with a given probability. Last but not least several exact quantile finding algorithms for WSNs have been suggested. In the following we will briefly discuss approximate and exact algorithms because they are more relevant to our work.

3.2.1 Approximate Algorithm

Shrivastava et al. [26] introduced a quantile summary structure, q-digest, that allows answering quantile queries with an error bound of $O(\log(r)/s_p)$ where r is the cardinality of the set of possible values and s_p the maximum packet size in the WSN. By increasing s_p the maximum error can be reduced at the cost of a higher energy cost. Since the actual error is often lower than the worst-case error, q-digest also provides a confidence factor that gives a more precise overview over actual errors. Considine et al. [5] extended q-digest for the case of fault-tolerant multi-path routing where duplicate values are inevitable. The Greenwald-Khanna Quantile Algorithm (GK) [8] is another solution for computing quantile summaries in a distributed environment. The number of transmitted values is bounded by $O(\log^2(N)/\epsilon)$ where $\epsilon \in\ ]0, 1]$ bounds the allowed rank error to $N \times \epsilon$, where, again N is the number of values observed in the network.

In Chapter 2, we proposed the HIU algorithm to compute the answer of a *Histogram* query 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 $h_i = \mid \{(s_j, v_j) \mid Lb_i \leq v_j < Ub_i, s_j \in S\} \mid$. Using a histogram of B bins, the median value can be computed with an error bounded by $r/(2 \times B)$, where r is the cardinality of the set of all possible values. First a median bin, b_m, is selected; it is the bin where the 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, i.e.: $\Sigma_{i=1}^{m}\{h_i\} \geq \frac{Count}{2} \wedge \Sigma_{i=m}^{B}\{h_i\} \geq \frac{Count}{2}$. Once b_m is selected, the median value can be approximated to be $\frac{Ub_m - Lb_m}{2}$. A better approximation is $Lb_m + \frac{Ub_m - Lb_m}{h_m} \times (\frac{N}{2} - \Sigma_{i=1}^{m-1}(h_i))$. The latter approximation is more accurate because it takes into consideration the rank of the median value inside the bin. However, it is still an approximation and it assumes that the distribution of all values in the median bin, b_m, is uniform.

The main difference between a median value computed by q-digest, GK Quantile algorithms and the HIU algorithm is that the two former algorithms have an error bound on the median's rank while the latter has an error bound on the median value. When considering of a sorted list with length n, the approximate quantile calculated by q-digest or GK is allowed to be *ranked* with a distance of at most $\epsilon \times n$ of the real quantile while the approximate quantile *value* computed by HIU is, in the worst case, off by $\frac{Ub_m - Lb_m}{2}$ of the real quantile value.

3.2.2 Exact Algorithms

Madden et al. introduced the TAG approach [17] to compute several exact aggregate queries using an in-network aggregation. However, for a *Median* query all values are transmitted and the median computation is performed centrally at the base station. Prakash et al. [25] suggested a specialized routing tree that is built according to the value distribution in the network in order to improve the performance of the queries. Greenwald et al. [8] extended his approximate approach to answer exact quantiles. The quantile is found in multiple passes by transmitting $O(\log^3(N))$ values.

Shamir [22] suggested to perform a binary search on the range of possible values to find exact quantiles. The POS algorithm [6] extends that approach to a continuous setting and applied additional improvements. Liu et al. [16] extended the binary search into a B-ary search by splitting the range into B sub-intervals in order to reduce the number of refinements. Like POS, it is also designed for a continuous setting. Kuhn et al. [13] used a B-ary search but the boundaries of the sub-intervals are based on random sampling. Li et al. [14] investigated the complexity of median queries in terms of time, message and energy complexity, by employing the algorithms by Kuhn et al.

Our proposed algorithms are exact but do not require a specialized routing tree-like [25]. Similar to [8] it depends on a quantile structure, a histogram, that can find an approximate answer. A Histogram of B bins is a B-ary search by definition. However, we do not use a fixed size histogram as [16] nor do we enforce a sampling technique to find the boundaries for the sub-interval as in [13]. Instead, our

approach uses a well-defined rule to decrease the number of bins, B, as the number of candidate answers decreases.

3.3 Snapshot Median Query Algorithm

In this section we denote a *Histogram* query as $Q(Lb, Ub, B)$, where Lb and Ub are the lower and upper bounds on the observed values within the WSN, respectively, and B is the number of bins in the histogram. Unless noted otherwise we assume that all bin ranges have the same length, and each Lb_i and Ub_i denotes the upper and lower bounds of bin $i, (i = 1, 2, \ldots, B)$. The *Histogram* answer is a vector $H = (h_1, h_2, \ldots, h_B)$, where $h_i = \mid \{\langle s_j, v_j \rangle \mid Lb_i \leq v_j < Ub_i, s_j \in S\} \mid$.

In order to find the optimum structure of all *Histogram* queries to compute the exact median answer, the base station should be aware of the whole network and each sensor's value. Such a centralized optimal solution is not feasible in the WSN context because we cannot assume that the base station is aware of the whole network and/or all values. If all values are known to the base station then it can find the median and there is no need for any queries.

In the remainder of this section we address two questions raised in Section 1.3, namely: (1) how to minimize the number of refinement *Histogram* queries, and (2) how to optimize the structure of the refining histograms?

3.3.1 How many Refinement Queries?

One of the main points in our algorithm is that, when participating in a *Histogram* query, nodes are autonomous to decide whether to forward the histogram or the values they have locally (both relative to their subtree). This decision is based on the cost and usefulness of the data to be forwarded and follows a simple rule. Let us assume that a local histogram H consumes $size(H)$ bits, while a sensed value v consumes $size(v)$ bits, hence all n values stored in a node's local storage consume $n \times size(v)$ bits. If $n \times size(v) \leq size(H)$, *and* all locally available values are sufficient to represent the current local histogram, then this node should send n values, otherwise it should send the histogram. If the cost is the same, it is more

beneficial to forward values instead of a histogram, because having values may help avoiding refinement queries (we discuss this in more detail shortly).

We use our previously proposed HIU algorithm [2] to answer each histogram refinement query. HIU assumes a continuous query and caches the calculated histogram in each node to be used in the next round. Although this chapter discusses snapshot queries only, the refinement queries in our algorithm can use the cached answer of the previous refinement query to prune network branches, namely those with no candidates to the *Median* query. For the time being we postpone the discussion about the *Histogram* query's structure to explain our RBM algorithm first.

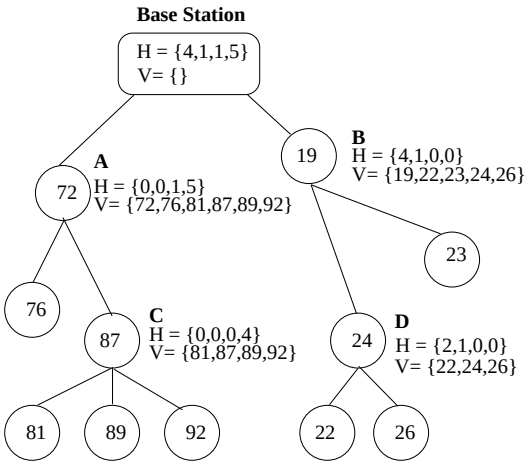


Figure 3.1: Processing a median query.

Figure 3.1 shows an example demonstrating how our algorithm works. We assume the number of bins in each *Histogram* query is fixed, i.e., $B = 4$, and all have ranges of equal length. All observed values reside in a range between 0 and 100. For simplicity, let us assume that each bin in the histogram consumes 1 byte and so does each sensor value. Each intermediate node has two objects, H and V. H represents the local histogram for the node's sub-tree, while V is a list of collected values so far in the node.

In the first refinement the initial (and first refinement) query is $Q(0, 100, 4)$, and all leaf nodes send their values to their parents. Node C decides to send the values because $size(H) = size(V) = 4$ and with the values in V one can reconstruct H. Node D also decides to send the values because $size(H) > size(V)$. Node A and

B both send H because $size(H) < size(V)$.

At the end of the first refinement, the base station reduces the number of candidate values to only 1. According to the histogram in the base station, the value in the third bin (values between 50 and 74) is the median answer because the number of values in $[0, 50[$ is 5 and the number of values between $[75, 100]$ is 5, as well. However, at this point only an approximate answer could be given, i.e., the actual median value unknown, thus another refinement query is required.

In the second refinement, the base station sends another *Histogram* query for the range between 50 and 74, i.e., it broadcasts the query $Q(50, 74, 4)$. Node B's subtree is pruned because the cached histogram in B shows no value in the third bin. Node A does not need to forward the query to node C because it already collected all values in its subtree. (In general, any node that already sent all values in its subtree should not participate in the next refinement queries.) At this point node A can use its cached values to answer the new refinement query producing $H = \{0, 0, 0, 1\}$ and $V = \{72\}$. Since $size(V) < size(H)$ only V is forwarded. When the base station receives this single value from B it can return the query's answer to the user.

We note that in the example above we always assumed the full histogram to be sent in order for simplification. In reality a node should send the full histogram or the set of pairs ⟨bin-id, count⟩ for the bins with non-zero count, whichever is smaller.

This processing of a *Median* query is materialized in Algorithm 5, which works in the base station. The algorithm keeps sending refinement queries to the WSN until the base station acquires all values in the median bin. Once the base station acquires enough candidates, it computes the median and the algorithm stops.

The RBM algorithm uses three functions: `FlexibleHIU`, `GetValuesInBin`, and `SelectHistogramStructure`. `FlexibleHIU` is very similar to the HIU algorithm (Chapter 2) where each node decides, as discussed earlier, to either send a histogram result or send all values in the histogram range to its parent. The `GetValuesInBin` function returns a list of values, from the base station's local memory, that reside in a specific bin. RBM uses `GetValuesInBin` to check

Algorithm 5 RBM(Logical Routing Tree T, Number of nodes N)

1: $Count \leftarrow zero$ {number of values in bins with ranges smaller than the median bin}
2: $k = \lceil \frac{N}{2} \rceil$ {k is the rank of the *median* in a list of N values –*note that this makes answering a generic Quantile query trivial*}
3: $Lb \leftarrow$ Smallest possible value in the WSN
4: $Ub \leftarrow$ Largest possible value in the WSN
5: $B \leftarrow$ `SelectHistogramStructure` (N)
6: **while** true **do**
7: $H \leftarrow$ `FlexibleHIU` $(T, Q(Lb, Ub, B))$ {H is a histogram containing the answer to the *Histogram* query Q}
8: $m \leftarrow arg\{b_m \mid Count + \Sigma_{i=1}^{m}\{h_i\} \geq k$ and $Count + \Sigma_{i=m}^{B}\{h_i\} \geq k\}$ {m is the index of the bin containing the median}
9: $Count \leftarrow Count + \Sigma_{i=1}^{m-1}\{h_i\}$
10: $BinValues \leftarrow$ `GetValuesInBin`(m)
11: **if** cardinality of $BinValues$ = h_m **then**
12: Sort $(BinValues)$
13: $Median \leftarrow (k - Count)^{th}$ value in $BinValues$
14: **return** $Median$
15: **else**
16: $Lb \leftarrow Lb_m$
17: $Ub \leftarrow Ub_m$
18: $B \leftarrow$ `SelectHistogramStructure` (H)

whether all median candidates in the median bin are already available, and so we can compute the median, or whether a new refinement query is required. The third function, `SelectHistogramStructure`, is related to the next Section, where we address the second of our two main questions: what is the optimum *Histogram* query structure in each refinement?

The number of refining queries is bounded by $O(log_B(N))$ if the values distribution is uniform or close to uniform. Even though we believe this is not a typical case, we note that when all observed values are equal, this algorithm will not terminate because sending values will always be more expensive than sending a histogram. Fortunately this can be easily remedied by adding another stopping condition of the base station: if the range of the histogram is only 1, e.i. $Ub - Lb = 1$, the algorithm terminates and report Lb as the exact median. This stopping condition assumes that all values are integers.

3.3.2 What structure for the refinement (histogram) queries?

We answer this question by investigating what is the optimum (or a good) number of bins for a *Histogram* query and whether to use the same or different sizes for each subsequent refinement query.

On the one hand, the more bins in a *Histogram* query, the more expensive its answer, but the fewer the average number of values in each bin and, subsequently, the fewer the number of necessary refinement queries. Conversely, using fewer bins requires more refinement queries before the base station can acquire all the needed values, hence the larger the overall cost. Our target is to minimize the number of bins in each query and minimize the number of candidates in the median bin.

Typically, a histogram is used to reflect the probability density function that represents the available values. Using different number of bins can reveal different features of the explored data, but, in general, there is no optimum number of bins for a *Histogram* query. Several guidelines and rules of thumb have been proposed to select the number of bins in a histogram. Some of these rules depend on the number of values. For instance, Sturges' rule suggests that $B = \lceil log_2(N) + 1 \rceil$ or $B = \sqrt{N}$. This rule, or one of its variations, is often used in statistical packages as the default

strategy [30]. This rule, and many others, were proposed to minimize the error between the true density function of all values and the density function represented by the histogram. None of them really meets our final goal of minimizing the number of values in the median bin.

We note, however, that using a rule such as Sturges' suggests that a variable histogram size for each refinement query may be a good solution. To check on that hypothesis we ran a small experiment, whose results are illustrated in Figure 3.2. We created three different WSNs with different number of nodes, each holding uniformly distributed values, and located within an area of 200 $\times$ 200 m. Then we measured the performance as the average number of bytes/node generated using our algorithm. First, our intuition above is confirmed, i.e., using a small number of bins is not useful because it requires many refinements, but using a very large B is not good either because it increases the cost of each *Histogram* query. Interestingly, all curves have a value of B that yields the better performance and this performance is very close to the one obtained using Sturges' rule, as detailed in Table 3.1. This supports the idea that using Sturges' rule for determining the histogram's size is indeed a good choice. Moreover, this leads to using a variable histogram sizes across consecutive refinement queries. Because each refinement query reduces the number of values in the median bin, i.e., the "universe" of values for the next refinement query. Thus, keeping the histogram size constant does not seem to make much sense. Sturges' rule addresses this observation automatically since it smoothly decreases the histogram size, hence the cost of processing a *Histogram* query, as the number of values in the median bin decreases. As a final note, Sturges' rule is known to be inaccurate if the number of values is fewer than 31, and the number of bins is fewer than 5. Accordingly, in our algorithm, we enforce a minimum number of bins of 5.

Number of values N	Performance using Sturges' rule	Best Performance in Figure 3.2
1000	4.82	4.09
3000	3.66	3.49
5000	3.32	3.29

Table 3.1: Determining a good value for B

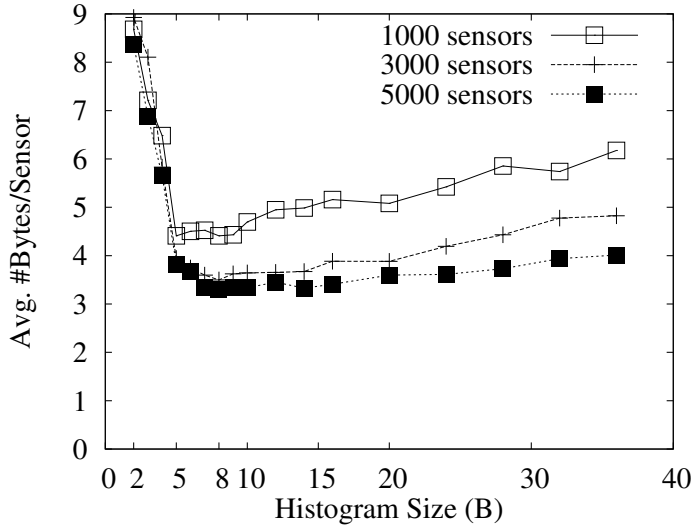


Figure 3.2: Empirical study for effect of B

When we ran the algorithm to obtain Figure 3.2 we were conservative about the choice for the length of the histogram bins, i.e., we assumed that all bin ranges had the same length. This conservative approach ensures that the number of refinement queries is $O(log_B(N))$ on average. One idea to be investigated is the use of variable size histogram bins. Recall that, by design, our algorithm may push some values to a node's parent if sending values is cheaper than sending a histogram answer. Although this is not a truly random selection for the values in the WSN, we can use the acquired values in the base station as a sample thereof. We envision some potential gain in using such sample in order to further fine tune the refinement *Histogram* queries. For instance, one could use histograms with each bin holding ranges of different lengths, in particular, using a finer granularity for bin where the median values is believed to reside. This idea however, may lead to worse performance if such a belief is incorrect. We leave a detailed study of this heuristic for future work.

3.4 Continuous Median Query Algorithms

In the previous section, we discussed how to use refinement *Histogram* queries to decrease the possible candidates for a median query. In many scenarios in the WSN context, a user wants to monitor an area periodically and then the same query

should return an answer every epoch. A straight forward extension for an algorithm is to re-run the same algorithm, every epoch. However, if the same query is going to be answered every epoch, we believe there is a possible opportunity to use the information collected during an epoch to decrease the amount of sent bytes in the next epoch. In fact, the HIU histogram algorithm has a potential to decrease the amount of sent bytes in the case of a *Histogram* continuous query as explained in Section 2.1.1.

In the following, we propose two new algorithms: HIU-Median and Continuous Refinement Based Median (CRBM). HIU-Median uses the histogram efficient algorithm 2.1.1 to find the range of the median bin then get all values in this bin. CRBM algorithm combines the benefit of both HIU and RBM algorithms. It will use cached histogram queries, from HIU, in each sensor to avoid sending non-useful messages between nodes. At the same time, it will use several refinement queries, as RBM, to compute the accurate median value efficiently regardless of the values distribution.

3.4.1 HIU-Median Algorithm

The classical TAG algorithm to answer a median query collects all values from all nodes to the base station, sorts the list and then select the correct value [17]. Instead of retrieving all values to compute the median, we can use the HIU algorithm to compute the answer of a *Histogram* query. The available histogram result in the base station is being used to narrow the requirements from collecting all values in the WSN to collecting all values in the median bin only. A bin that contains the median value can be identified using the *Histogram* answer in the base station. The algorithm to compute the median in the base station using a histogram answer is presented in Algorithm 6. It has two parameters: the WSN logical routing tree (T) and the *Histogram* answer (H) in the base station and it 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 HIU-Median algorithm computes and returns the exact *Median*. The algorithm for function GetVALUES is presented in Algorithm 7. It collects all values between

Algorithm 6 HIU-Median(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 7 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** $\Sigma_{i=1}^{B}\{h_i\} = 1 \mid b_i \cap [L, U] > 0$, $\forall h_i \in$ the local histogram and $\forall b_i \in$ *Histogram* query Q **then**
7: **return** V {The single value in the tree is already found}
8: **if** $\Sigma_{i=1}^{B}\{h_i\} > 0 \mid b_i \cap [L, U] \neq \phi$, $\forall h_i \in$ the local histogram 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

values L and U in the given tree T. Instead of sending the request to all nodes in the tree, it uses the cached histogram in each node 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 cost is the number of bits to retrieve these values which is $h_m \times L \times log_2(max(v_j))$ where $1 \leq j \leq N$, m is the bin id that contains the median value, and L is the maximum depth in the routing tree.

Clearly, the number of retrieved candidate values can be significantly reduced by using a single histogram. If all values in the WSN follow a uniform distribution, using a single histogram refinement query will reduce the number of retrieved values from N to $\frac{N}{B}$. However, if the variance is very small then a single histogram will not significantly reduce the number of retrieved values. In the worst case, the total cost for using a refinement query could be higher than TAG.[1] Next we show a heuristic efficient algorithm that uses multiple *Histogram* answers to reduce the number of candidate answers regardless of the values' distribution.

3.4.2 Continuous RBM Algorithm (CRBM)

The Continuous RBM (CRBM) algorithm will run the RBM algorithm in the first round. After computing the first median value, the next rounds should benefit from the existing cached values/histograms in the base station and in each sensor node. In the consecutive rounds, instead of building the most efficient histogram for computing the median as we do in RBM, CRBM looks for a *Histogram* query that could be answered using cached data in each node. Nodes send updates to the cached answer computed using the previously cached messages in their parents' memory. The CRBM algorithm has three main components: (1) Constructing a refinement query, (2) Computing a cached answer either in the node itself or in its parent, and (3) Optimizing the message to be sent. In the following, we explain each component in more detail.

[1] For example, if all values are equal or reside in a single bin.

Constructing a Refinement Query

Using the in-node cached histograms is only useful if the same *Histogram* query is used every round. In the first refinement, the query is issued from the base station and it depends on the median bin found in the previous round. Since a median value is expected to be stable, we can use the latest refinement query that found the median value in the previous round as a starting point instead of starting by a *Histogram* query that covers the whole possible range of values. However, in some cases, if the median value is substantially changed since the previous round, its value might be out of that query's range. This will lead to a wrong answer. In order to solve this problem and at the same time use the available information retrieved from the previous round, we will use a biased *Histogram* query.

The biased *Histogram* query will have bins with different sizes. In the first refinement we use a *Histogram* query with two types of bins, small-bins and extended-bins. The small-bins consists of the median bin in the second latest refinement query of the previous round, and surrounded by some other bins from each direction. These bins might be defined in any refinement query but should be in the same round. We will simply select the smallest possible consecutive bins defined in any refinement query before and after the median bin. The small bins facilitate finding the median bin quickly while the two extended bins guarantee this query covers the possible range of possible values and are required to guarantee the correctness of the final result.

The *Histogram* query structure is two folds: number of bins and the range of each bin. We will use the Sturges' rule, as we have done in RBM, to decide on number of bins. However, we will round the upper or lower log value to guarantee an odd number of bins. Because we are not sure if the median value will increase or decrease in the next round, we want to have the median value found in the previous round exactly in the middle of the new histogram query and surrounded by an equal number of bins before and after the median bin.

It is important to increase the probability of having *Histogram* refinement queries with similar bins in any two consecutive rounds to increase the probability of using the cached results and then decrease the overall cost. We will craft the histogram

queries to match the refinement queries in the previous round. The first step is to select the range of the median bin, the one in the middle of the new *Histogram* query. We need to chose a bin with small range to decrease the number of candidates but not very small so that any change to the median value makes this bin not useful. In fact, each refinement query has a median bin but we need to decide which refinement we should start with.

Refinement choice	Average number of sent messages after 100 rounds
First Refinement	12.6583
Second Last Refinement	10.3169
Last Refinement	10.8322

Table 3.2: Determining the best strategy for the first *Histogram* query in each round

Table 3.2 shows a comparison between using the first refinement, last refinement, or the second last refinement. It is clear that using a refinement in the middle is the best because it balances between using a small median bin range so that the number of candidates is small but not very small as in the last bin when the new median value is always out of range. Figure 3.3 shows the average cost of each round using each strategy. Using the first refinement only is very close to running the RBM algorithm every round and makes little use of the available info about the median. That is why the cost of each round is almost stable but much higher than the other two strategies. If the CRBM uses the median bin in the last refinement, the median bin will occasionally fall in the extended (not small) bins and then the CRBM will use many refinements frequently, which leads to a higher average for the overall query. The best is using something in between. We leave the optimization of which refinement leads to the minimum cost as future work. In our implementation, we will use the second latest refinement as a starting point.

After collecting the histogram from the first refinement, if the median value is not found, the median bin will be divided again. However, instead of enforcing the Sturges' rule, the CRBM algorithm uses the bins from queries asked in the previous round. The maximum number of bins is the one suggested by Struges' rule, however, a smaller number of bins is accepted as well. There are two reasons for this: (1) Reduce the cost of sending the refinement query because each node can

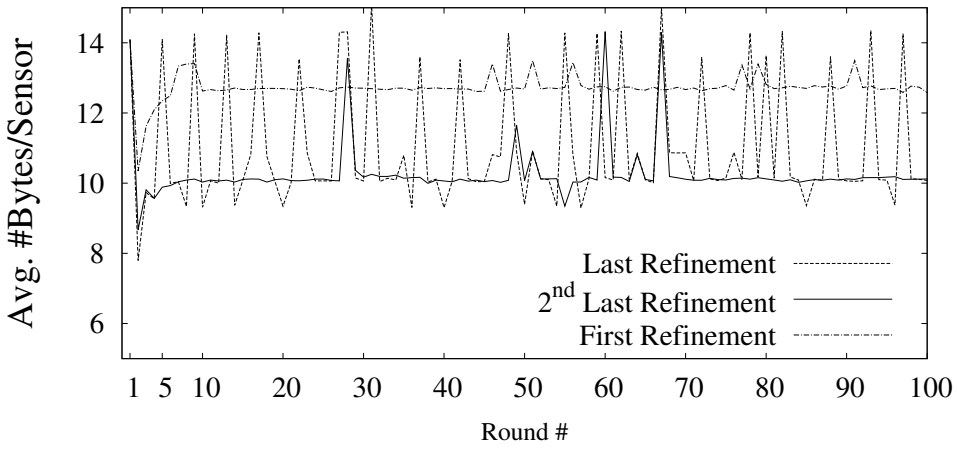


Figure 3.3: Comparison between using a median bin in different refinements in the first *Histogram* query of each round

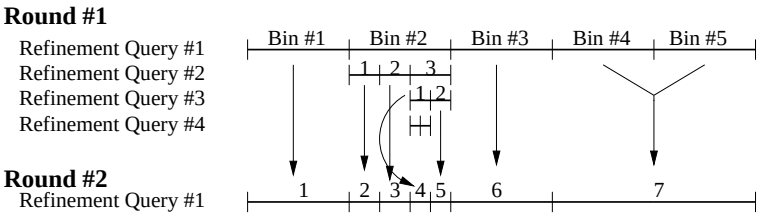


Figure 3.4: Example for constructing the first refinement in the CRBM algorithm

reconstruct the query using the index of the median bin in the last refinement query and number of bins only. Unlike RBM, to allow any node to reconstruct the biased histogram while sending only the median bin's index and number of bin in the new histogram query, all intermediate nodes should receive the histogram query of all refinements even if the node already sent all its value or its subtree should not be involved. In either case, the intermediate node will receive the query, forward it to its children but may or may not respond to the query. (2) Computing a histogram answer that has been cached already by the HIU algorithm is at least 50% cheaper than computing it from scratch as we have seen in Section 2.3.

In Figure 3.4 we show an example for selecting the first refinement query based on previous round's queries. The bin value was found in refinement number 4 during the first round. In the second round, the first refinement was selected to have seven bins. The first bin is the median bin selected in the second latest refinement of the previous round, Bin number 1 at Refinement query number 3 in Round number

1. Two more bins are added to each side of the median bin. From the right hand side we selected Bin number 2 from Refinement query number 3 and Bin number 3 from Refinement query number 1. On the left hand side, we selected bins number 1 and 2 from Refinement query number 2. After that, two extended bins are added to cover the remaining range. On the right hand side, we added one bin that covers both bins number 4 and 5 in Refinement number 1. On the left hand side the extended bin has only one bin from Refinement 1. The constructed query have bins from different refinement queries. Since all these queries already sent to all relevant nodes in the WSN, each sensor can locally compute a cached answer for the newly constructed query.

Computing the Cached Answer

Computing the cached answer by each node for itself and for its children is very important to reduce the amount of communication because the refinement queries may differ from one round to the next. There are two properties in the refinement queries constructed above that facilitate computing the cached answer in each bin. (1) A value will not be counted twice in two different refinement queries except if it falls in the median bin of both queries. (2) All bins in a refinement query exists in one of the refinement queries in the previous round.

Based on these properties, the main equation to compute the cached answer for a bin in a refinement query, ranges from LB to UB, is $\Sigma_{i=0}^{n-1}(\Sigma_{j=0}^{B_i}(h_j^i)) \mid lb_j^i >= LB$ and $ub_j^i <= UB$, where n is number of refinement queries, B_i is number of bins in a refinement query i, h_i^j is the count for a bin j in a refitment query i, also lb_j^i and ub_j^i are the lower and upper bound of bin j in a refitment query i, finally, LB and UB are the lower and upper bound for the bin we want to compute its cached count. In other words, any bin in any refinement query in the previous round that falls between the new bin's boundaries (LB, UB), will be taken into consideration. Please note that we cannot use bins in any refinement queries after the one selected as the main refinement query. For example, in Figure 3.4, Bin number 2 in the second round takes the same count as bin number 1 in the refinement number 3 of the first round. Also, Bin number 7 in the second round, takes the count of bin

number 4 plus the count of bin number 5 in the first refinement query of the first round.

In order to reduce the amount of message transmission, a node's parent should be able to compute the node's cached answer as well. For this reason, a parent keeps a copy from every message received from each node of its children. These cached messages are valid for one round only. For the caching purpose, it is sufficient that each node stores all refinement answers, initiated by itself or by of its children, for two rounds.

Optimizing the Message to be Sent

Leaf nodes will send the difference between current and previous values, either positive or negative. A sensor may need 1 or 2 bytes to represent it. In the worst case, the differences will use 2 bytes which is similar to the RBM algorithm. Intermediate nodes in the RBM algorithm could either send list of values or send the histogram. In CRBM, the algorithm chooses from three possible message types based on the message size: (1) sending the histogram, (2) sending the list of all values, and (3) sending the list of changed bins only. An intermediate node, computes its own cached and accurate answer for the current query then constructs a message that can update the cached version. The parent can also compute this cached answer and by applying the sent message, it can update the node's answer and then aggregate it to compute its own histogram.

3.5 Performance Evaluation

In this section we evaluate the performance of our proposed RBM algorithm and compare its performance against the well-known TAG approach [17]. The B-ary search in [13] was not used because the algorithm minimizes the number of time slots required to compute the median instead of the more common goal of minimizing the amount of bytes transmitted. Moreover, our preliminary results show that this algorithm needs more bytes than TAG. Since the typical goal within the realm of WSN research is to minimize the energy consumption and since the data transmission is the main reason for the energy consumption, we will use the average

number of bytes sent in the network as our performance indicator. It also allows a comparison between algorithms in terms of overhead and a comparison with the minimum bound, explained later.

Similar to the setup in Section 2.3, all algorithms use the same physical network and logical trees. The physical network is represented by the graph that connects all nodes. We use three different physical networks generated by placing nodes in the monitored area using a uniform distribution. The routing tree is a typical shortest path tree. For each physical network we build our experiments using five logical trees by selecting five different base stations. We make the following assumptions about the required storage in each node: (1) an observed value consumes 2 bytes, (2) each bin in the histogram consumes 2 bytes to accommodate the count of all nodes in the network, if necessary, and (3) If a node decides to send updates about b histogram bins only, instead of sending the complete histogram, then the message size is $3 \times b$ bytes because additional bin IDs, consuming 1 byte, need to be sent as well. Finally, the maximum packet size per message is 128 bytes. A complete histogram requires $2 \times B$ bytes, where B is the number of the bins in the histogram.

We investigate our snapshot algorithms with respect to four parameters: Radio range R, Network's message header size H_s, the number of nodes N and the standard deviation σ of the observed value. Continuous algorithms have two more parameters: average amount of change in a sensor's value δ per round, and the probability that a sensor's value change ρ between rounds. The radio range controls the logical network topology and it changes both the branching factor and routing tree depth. Studying the header size is important because it may be a tunable parameter in the communication protocol and increasing the message header effectively is detrimental to our approach because it increases the overhead of the refinement queries. Increasing the number of nodes N, shows the algorithm scalability from the WSN density point of view. Varying the standard deviation allows to examine the worst case of our approaches compared to TAG's, e.g., when all values are equal. Finally, δ and ρ control how the sensor' values change between two consecutive rounds. Table 3.3 contains a list of investigated parameter' values. While testing one parameter, we use the default value (denoted in bold) of all other pa-

rameters. The figures show the amortized average of the transmission cost in our experimental runs. In order to ensure a fair comparison, all algorithms use exactly the same setup, during all simulations.

We used two datasets, a synthetic and a realistic dataset similar to ones used in Section 2.3. A synthetic dataset allows the evaluation of more parameters and to discuss how the algorithms behave with respect to them. Our synthetic dataset consists of N connected nodes uniformly distributed in an area of $200m \times 200m$.

Instead of using a uniform distribution to initialize the sensor' values as done in Section 2.3, we use a normal distribution because it may have a very small standard deviation which makes all value resides in a single bin. This is the worst case for our proposed algorithm in this chapter while it is considered the best case for the previous chapter. Our median algorithms use several refinements to shorten the range of the median value and decrease the number of candidates. If all values resides in a single bin, then all values should be involved in the following refinement. However, in the HIU algorithm, if all values reside in a single bin, nodes need to send an update to a single bin which is cheaper than sending updates for all bins in the histogram, if using a uniform distribution. There is no need to send any updates in the following rounds if all values change within this bin. In our synthetic dataset, values are initialized using a normal distribution with an average 0, a standard deviation σ and ignoring values in the distribution's tails, namely outside the interval [-2,2]. We use a normal distribution to show the impact of having equal values or values uniformly distributed in the network. After that, all values are scaled to the range between 1 and 2^{16}, i.e., using 2 bytes per sensed value. In the continuous queries, we assume that each value changes every round with a probability $\rho\%$. The random distribution for the change value is an exponential distribution with an average $\frac{1}{\delta}$. An exponential distribution allows a sensor to have very small or very large changes to its value while maintaining the same average.

The realistic dataset we use was also used in [28]. This dataset represents the atmospheric pressure and it was derived from data collected by the Live from Earth and Mars project [2] by extracting data traces for 1022 nodes. However this dataset

[2]http://www-k12.atmos.washington.edu/k12/grayskies/nw_weather.html

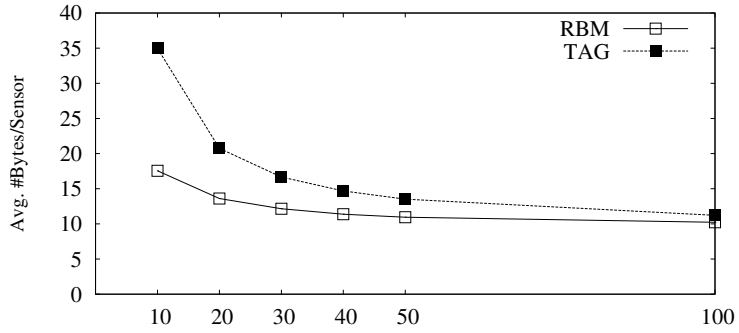
Parameter	Used Values
R (WSN sensor's radio range)	$10, 20, \mathbf{30}, 40, 50$
H_s (Header Size)	$0, 4, \mathbf{8}, 16, 32$
N (Number of Nodes)	$250, 500, 1000, 1500, 2000, \mathbf{3000}, 4000, 5000$
σ (standard deviation)	$0, 0.25, \mathbf{0.5}, 0.75, 1$
δ (Average amount of change)	$1\%, 25\%, \mathbf{50\%}, 75\%, 100\%$
ρ (Probability of change)	$1\%, 25\%, \mathbf{50\%}, 75\%, 100\%$

Table 3.3: Studied parameters for RBM and CRBM analysis (default values in **bold**)

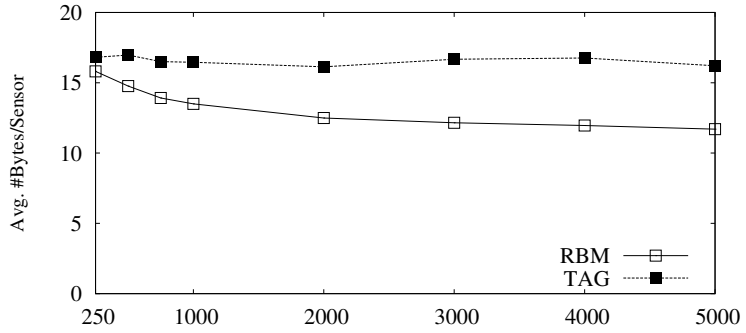
did not contain information about the spatial distribution of the data. In order to create a realistic setting for this dataset, we made a reasonable assumption that neighboring nodes produce similar values. Then, we used a self-organizing map approach similar to [12]. We only used the first measurement of each node as an input of the self organizing map, and the output is the position of each node. The first measurement was also used to initialize the sensor' values in the first round in case of the continuous algorithms and used as the sensor' values for the snapshot algorithms.

3.5.1 Snapshot Median Query

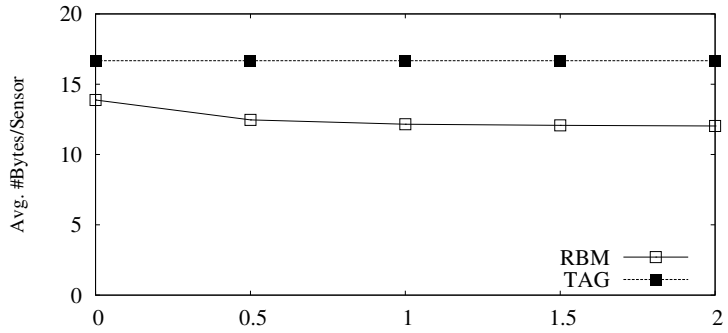
All graphs in Figure 3.5 show that RBM decreases the amount of sent bytes by up to 50% of what TAG sends, e.g., when the radio range is small and consequently the network's maximum depth is large. In Figure 3.5(a) the two algorithms decrease the number of bytes sent as the radio range increases until they both reach the minimum bound, namely 10 bytes/node, when all nodes can connect directly to the base station and no in-network aggregation is required. Sending 10 bytes on average per node is the minimum bound because each node should at least communicate its value. A value message is 2 bytes and needs 8 bytes as a default header size. It is worth mentioning that RBM gets closer to the minimum bound when the radio range is $30m$, while TAG does not do so until the range is $100m$. Being able to approach the lower bound at smaller radio ranges is important because in WSNs the radio range is typically in the order of a few tens of meters and, more importantly, the energy cost of transmissions grows (at least) quadratically with the growth in the radio range.



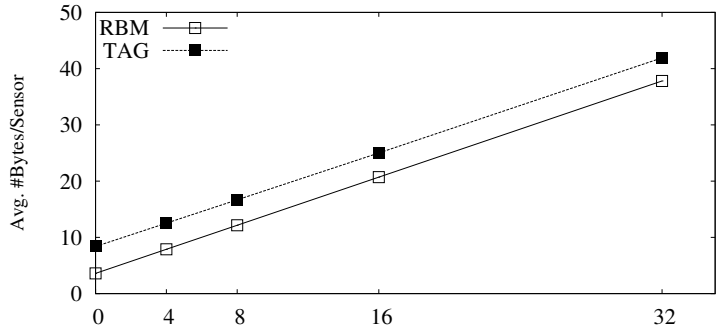
(a) Range



(b) Number of Nodes



(c) Standard Deviation



(d) Header Size

Figure 3.5: Performance of RBM using average number of bytes sent per a sensor node (Synthetic dataset)

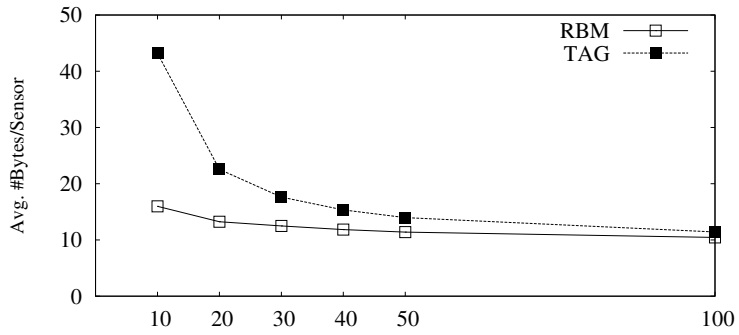
TAG's performance does not change too much while increasing the number of sensor nodes as one can see in Figure 3.5(b). This is due to the fact that because all nodes send their values, increasing the number of nodes increases the traffic linearly and that does not affect the average traffic very much. On the other hand, RBM slowly decreases the amount of bytes sent because the more nodes in the network the more leaf nodes. Leaf nodes and, often whole sub-trees participate only in the first refinement when they communicate their values, hence the gain.

Figure 3.5(c) shows that, as expected, the TAG algorithm is completely independent of the standard deviation value. The RBM algorithm, on the other hand, decreases the amount of bytes sent as the normal distribution gets closer to the uniform shape. In RBM's worst case, when all values in the network are equal, TAG sends 20% more bytes than RBM while it sends 40% more if the distribution is closer to a more uniform one.

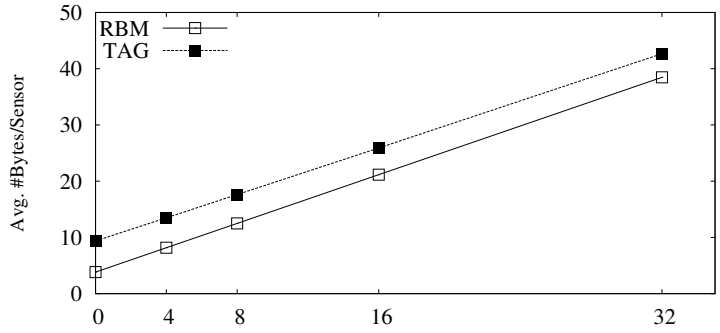
Figure 3.5(d) shows that performance of the TAG and RBM degrades similarly with an increase in the header size. Although RBM sends more message as a result of refinement process, these messages are still relatively small. TAG sends fewer but relatively larger messages (in fact, leaf nodes send small messages of a single value while nodes near to the base station send very large messages that include all values in their subtree) and is forced to split those because of the maximum packet size of 128 bytes, thus it is not able to diminish the gap between itself and RBM.

It is worthwhile to note that, in all experiments above, RBM's performance was on average, only 25% away from the minimum bound of 10 bytes/node, which further supports RBM's efficiency.

Figure 3.6 shows the performance of the TAG and RBM algorithms using the realistic dataset. Since some parameters are fixed in the realistic dataset, i.e., N and σ we could only investigate the range R and header size H_s. Figures 3.6(a) and 3.6(b) show that, qualitatively, the performance of RBM and TAG using the realistic dataset is very similar to our synthetic dataset and confirms RBM's superiority with respect to TAG.



(a) Range



(b) Header Size

Figure 3.6: Performance of RBM using average number of bytes sent per node (Realistic Dataset)

3.5.2 Continuous Median Query

For the sake of comparison we show the performance of four algorithms to evaluate our continuous algorithms. First, TAG [17], that collects all values to the base station in order to compute an exact answer. Second, HIU-Median (Section 3.4.1), that answers a continuous histogram efficiently, then computes the median value by finding the median bin and then requests all values in this bin. Third, repeating the RBM (Section 3.3) every round. We call this algorithm RBM*, and finally, the CRBM algorithm. We use the TAG and RBM algorithms as base lines for our proposed algorithms for the continuous *Median* query.

We will start with a discussion of the performance of both HIU-Median and CRBM with respect to the baseline algorithms. Then, we compare the four algorithms. In the discussion of each algorithm we use a Synthetic dataset and five parameters: Radio range R, Network's message header size H_s, the standard devi-

ation σ of the observed values, the average amount of change in the sensor's value δ, and the probability that a sensor's value change ρ. In the comparison discussion, we assume the default value for all parameters as mentioned in Table 3.3.

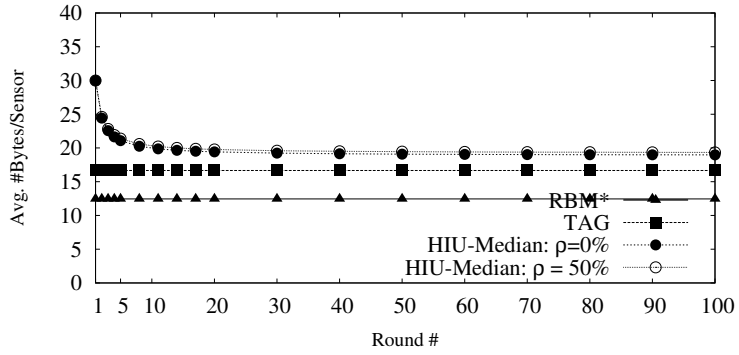
HIU-Median Algorithm

Figures 3.7 and 3.8 show that the TAG approach of collecting all values to compute an exact *Median* may outperform HIU-Median while RBM* always performs better than both HIU-Median and TAG. The figures show the amortized average of sent bytes per node. The cost of the *Median* query using HIU has two components: (1) Cost to acquire the histogram answer and then define the median bin, (2) Cost to collect all values in the median bin. In the first round, HIU does not reuse existing information. However, in consecutive rounds, HIU reduces the 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 explained in Section 3.4.1. In some figures TAG outperforms HIU-Median because HIU-median sends more messages.

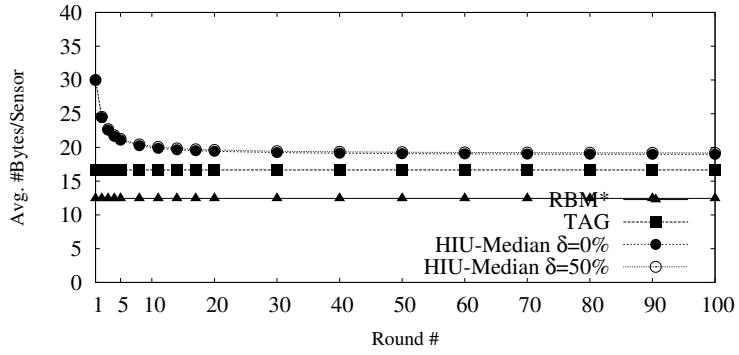
Figures 3.7(a) and 3.7(b) show the influence of changing ρ and δ as the number of rounds increase which affects the cost of constructing a histogram with no influence on the values collection's cost. In principles, the smaller the ρ (or δ), the cheaper the *Histogram* and then the cheaper the Median. In practice, the figures show that ρ and δ have little influence on the performance of HIU-Median because the difference is dominated by the default header size (8 bytes).

In Figure 3.8(a) the HIU-Median algorithm significantly outperforms TAG and slightly outperforms RBM* if the header size is zero (H=0). Although this is a non-realistic assumption, it is useful to show that HIU-Median actually sends less bytes than both RBM* and TAG but more messages.

Figure 3.8(b) shows the influence of the sensor's radio range (R) on the *Median* query cost. This parameter affects the TAG, RBM* and HIU-Median cost because it changes the logical routing tree. However, the costs of TAG and RBM* are fixed regardless of the number of rounds. The larger the node's radio range, the smaller

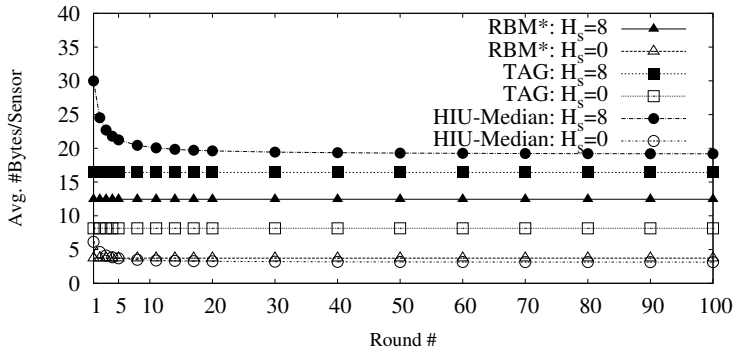


(a) Studying HIU-Median using the Probability a value changes (ρ)

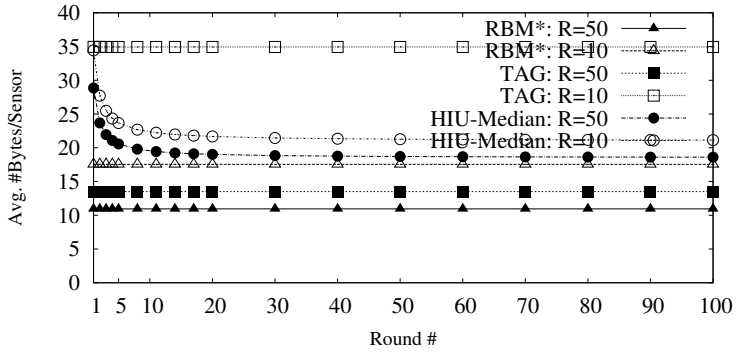


(b) Studying HIU-Median using the Average amount of change(δ)

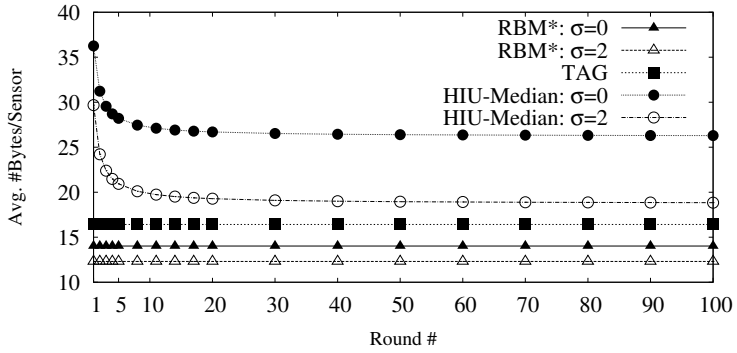
Figure 3.7: Influence of the continuous parameters on the cost of running HIU-Median in terms of number of sent bytes (X-axis is number of rounds)



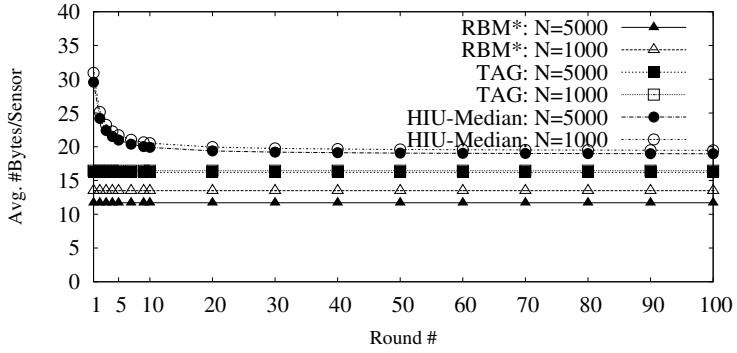
(a) Studying HIU-Median using the Header Size (H_s)



(b) Studying HIU-Median using the Radio Range (R)



(c) Studying HIU-Median using the Standard Deviation (σ)



(d) Studying HIU-Median using the Number of Nodes (N)

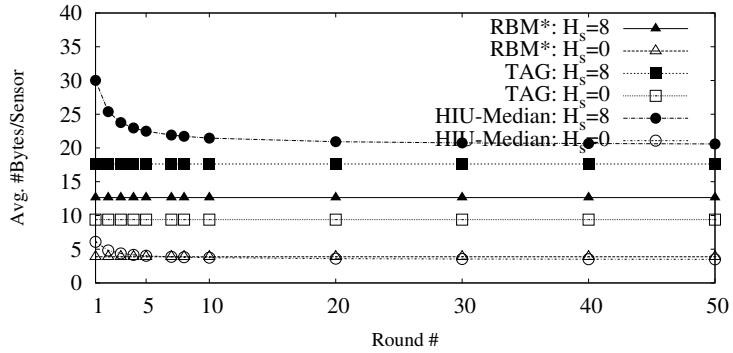
Figure 3.8: Influence of the setup parameters on the cost of running HIU-Median in terms of number of sent bytes (X-axis is number of rounds)

the number of hops to reach the base station and the smaller the number of bytes sent by TAG, RBM* and HIU-Median. Conversely, the number of messages increases as the radio range gets smaller. In Figure 3.8(b), TAG performs significantly worse than RBM* and HIU-Median if the radio range is small ($R = 10m$). HIU-Median outperforms the TAG algorithm when radio range is very small. In fact, the cost of TAG significantly increases by increasing the radio range while the cost of HIU-Median is similar regardless of the radio range because the HIU algorithm avoids sending messages by many nodes.

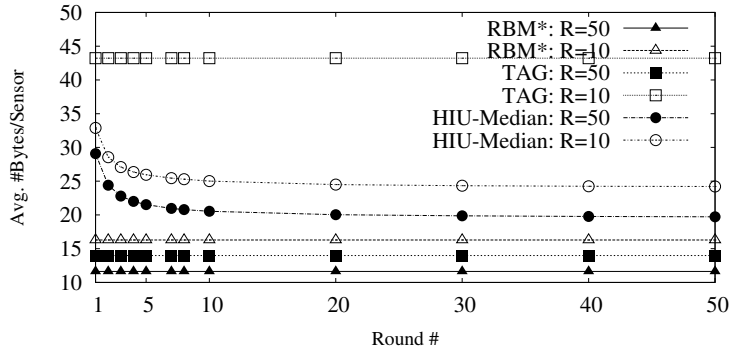
Figure 3.8(c) shows the HIU-Median's worst case. If the standard deviation is zero then all values reside in the same bin and then HIU-Median should perform worse than TAG. In this case, HIU would send bytes to compute the histogram every round and then collect all values to the base station as TAG. However, the values distribution has no influence on TAG and only minor influence on RBM*'s performance.

In the last figure, Figure 3.8(d) shows that increasing the number of nodes in the network has a limited impact on the HIU-Median algorithm. As seen in Section 2.3, the HIU algorithm gains a small benefit when the number of nodes in the network increases. At the same time, the more nodes in the network the more nodes in the median bin and the higher the cost of the value collection phase. Because the value collection phase benefits from cached histograms in the nodes and prune some subtrees and because all leaf nodes do not participate in the value collection, because they already sent their values in the HIU phase, increasing the number of nodes in the network has little overhead on the values collection phase.

The realistic dataset in Figure 3.9 confirms our previous findings. Since some parameters are fixed in the realistic dataset, e.g., N and σ and others are not known to us, e.g., δ and ρ, we could only investigate the range R and header size H_s. Figure 3.9 shows that the performance of RBM* and HIU-Median using the realistic dataset is similar to using our synthetic dataset and confirms that RBM* outperforms HIU-Median if the header size is not zero.

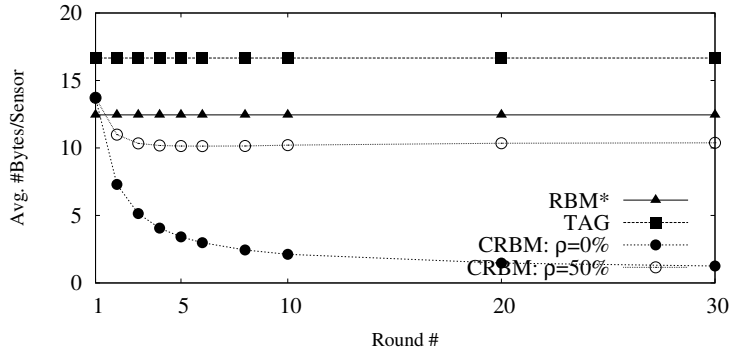


(a) Studying HIU-Median using the Header Size (H_s)

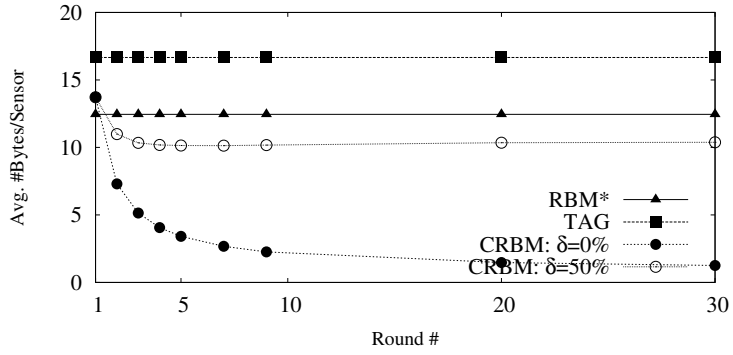


(b) Studying HIU-Median using the Radio Range (R)

Figure 3.9: Studying the HIU-Median using the realistic dataset in terms of number of sent bytes (X-axis is number of rounds)



(a) Studying CRBM using the Probability a value changes (ρ)



(b) Studying CRBM using the Average amount of change(δ)

Figure 3.10: Influence of the continuous parameters on the cost of running CRBM in terms of number of sent bytes (X-axis is number of rounds)

CRBM Algorithm

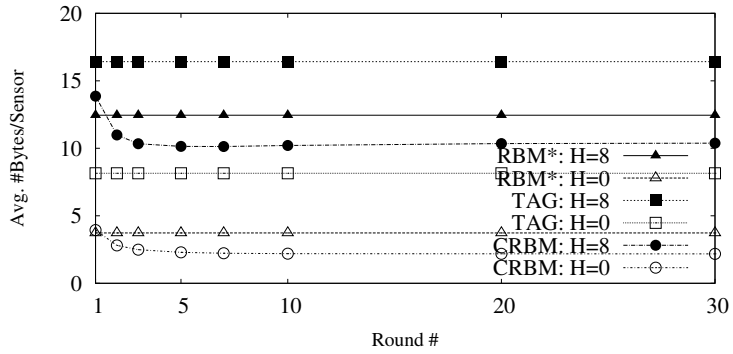
The performance of the CRBM algorithm is compared with our two baseline algorithms, TAG and RBM*. Figures 3.10 and 3.11 show the amortized average of sent bytes per node. The X-axis of all figures is the number of rounds and the Y-axis is the amount of sent bytes. The figures show that CRBM is capable of computing the median with a cost very close to the minimum required[3], in fact all graphs show that CRBM uses slightly more bytes than ten when the header size is eight.

Figures 3.10(a) and 3.10(b) show that the CRBM algorithm uses less bytes than both TAG and RBM*. Decreasing the probability and/or the amount of changes from 50% to 0% significantly reduces the amount of sent bytes from ten to one byte per node per round. If the values are almost static or slightly change from a round to the next then the average amount of sent bytes will approach zero. We note that in the first round, CRBM uses more bytes than RBM* because RBM* prunes a subtree of a node once this node sends all its values while CRBM continues sending the refinement queries to all intermediate nodes in the network even if they already have sent all their values. As was mentioned, in Section 3.4.2, CRBM does so to allow all queries to reconstruct the refinement query with minimal information even though not all bins have the same size.

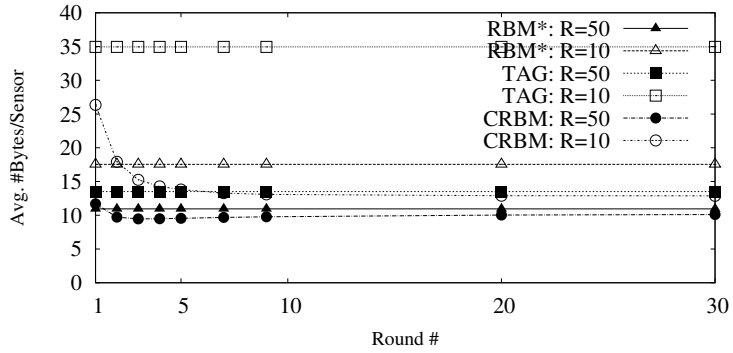
Figure 3.11(a) shows that the CRBM and RBM* use almost the same amount of bytes in the first round if the header size is zero. If the header size increases CRBM performs worse in the first round because CRBM sends more messages, again because the refinement query is sent to all intermediate nodes. However, in the second round, CRBM performs better regardless of the header size. At our default header size, header = 8, CRBM decreases the amount of sent bytes by 20% than RBM*.

In Figure 3.11(b), we show how the network structure influence CRBM, RBM*, and TAG. If the radio range is ten, then the maximum number of hops between a leaf node and the base station is large. If the radio range is 50, then almost all nodes are very close to the base station and then the number of intermediate nodes

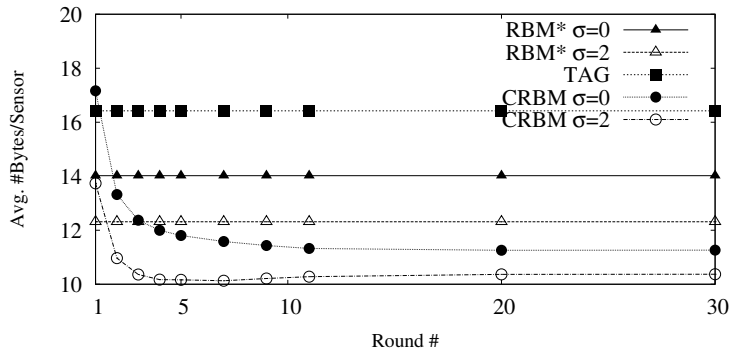
[3]The minimum number of required bytes every round equals to the number of bytes required to represent a sensor's value plus the number of bytes for the header size.



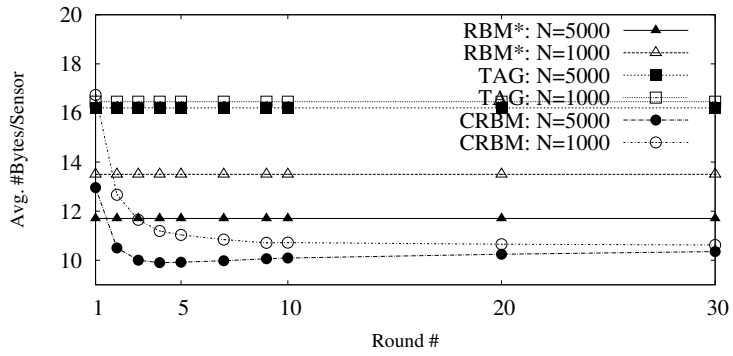
(a) Studying CRBM using the Header Size (H_s)



(b) Studying CRBM using the Radio Range (R)



(c) Studying CRBM using the Standard Deviation (σ)



(d) Studying CRBM using the Number of Nodes (N)

Figure 3.11: Influence of the setup parameters on the cost of running CRBM in terms of number of sent bytes (X-axis is number of rounds)

is very small. Using a radio range of ten makes the CRBM algorithm worse than RBM in the first round because it sends more messages to force all refinement queries to reach all intermediate nodes. This difference in the number of messages increases because several hops are needed to reach all intermediate nodes. After a few rounds, CRBM shows enhancement in the performance and decreases the amortized average of sent bytes. However, if the radio range is large, e.g. $50m$, then the performance of the CRBM will be slightly worse than RBM* in the first round and slightly better than RBM* in the next rounds. The reason is the small number of intermediate nodes. In both RBM* and CRBM, if a value changes in a leaf node, this node should send its value every round to ensure an accurate computation for the median. The fact that most nodes are leaf nodes, when the radio range = $50m$, reduces the CRBM potential to decrease the number of sent bytes.

Figure 3.11(c) clarifies how RBM* and CRBM behave when changing the standard deviation for the node's values. The initial value of the nodes follows the same normal distribution with the same standard deviation. If the standard deviation is zero then all values are equal to each other, which is the RBM*'s worst case. If the standard deviation is two, then all values follow a distribution very similar to a uniform distribution. The figure shows that CRBM decreases the number of sent bytes starting from the second round as in all other figures. CRBM performs better when the values have a distribution similar to uniform for two reasons: (1) CRBM uses the RBM algorithm in the first round. (2) If the sensors' values are close to each other, then the median value is expected to fall in one of the extended bins not one of the small bins. Note the worst case of the RBM is when $\sigma = 0$. Earlier in Section 3.4.2, we explained that a refinement query in CRBM usually has some small bins and two extended bins. If $\sigma = 0$, then the small bins will be very small because CRBM will not terminate unless the width of the median bin is one. If the small bins are very small then they will not be capable of accommodating the median value of the next rounds. This will cause the median value to fall in one of the extended bins more frequently and then increase the cost of the CRBM.

Figure 3.11(d) shows that RBM* is the only algorithm that significantly benefits from decreasing the number of nodes in the monitored area. Although increasing

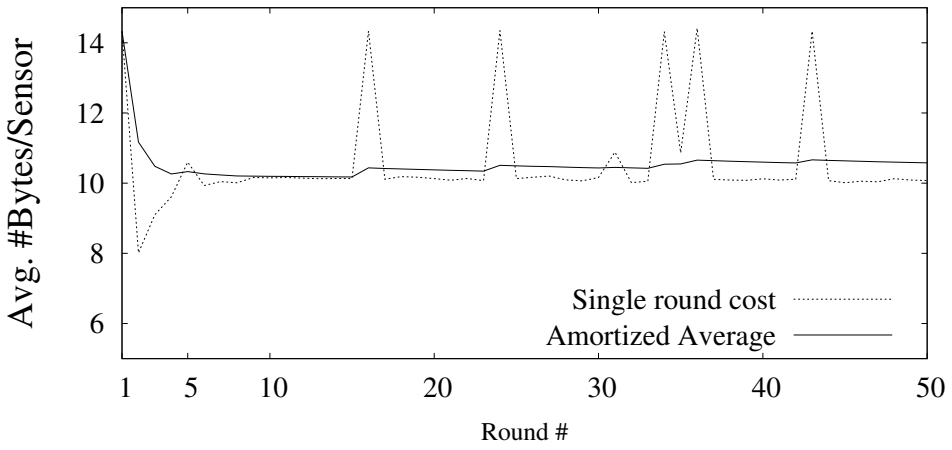
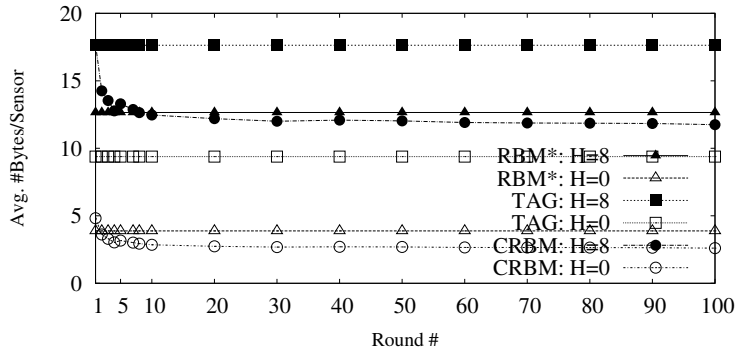


Figure 3.12: Per-round cost vs. Amortized average for the CRBM algorithm

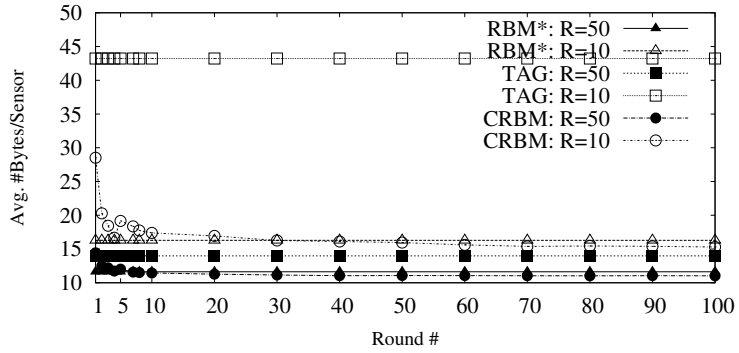
the number of nodes allows CRBM to send less bytes, the improvement is relatively small because CRBM is close to the minimum bound already (10 bytes). As explained in Section 3.5.1, RBM decreases the average amount of sent bytes when the number of nodes increases in the monitored area because the more nodes in the network the more leaf nodes. Leaf nodes and, often whole sub-trees, participate only in the first refinement when they communicate their values, hence the gain.

There is an interesting behavior for some CRBM curves. In the first round the cost is high, then it starts decreasing for a few rounds and increases then again and remains stable. In order to better understand this unexpected behavior of CRBM, we took a closer look at the cost of each round, not only the amortized average.

Figure 3.12 shows the average amount of sent bytes in each round per node and the amortized average for all rounds. It is clear that the CRBM algorithm eventually requires a high amount of bytes every few rounds. Recall that CRBM starts with a regular histogram query where all bins are equal then in the next round customizes the histogram query to reuse the cached results in each node. After a few rounds, as the sensor's values change, the median value may no longer be in any of the small bins and then requires a query for the extended bin. This increases the cost of CRBM for two reasons: 1) the first refinement query brings a minor benefit and is considered just an overhead, 2) the new query for the extended bin has no cached results and then CRBM cannot reduce the number of necessary bytes



(a) Studying CRBM using the Header Size (H_s)



(b) Studying CRBM using the Radio Range (R)

Figure 3.13: Studying the CRBM using the realistic dataset in terms of number of sent bytes (X-axis is number of rounds)

to answer it. As explained in Section 3.4.2, we use the second last refinement query in the previous round as a guid for the first refinement query in the current round. Changing this assumption can change the average number of rounds between the peaks in the graph. Controlling the cycle length of the rounds' cost curve is left for future work.

The realistic dataset in Figure 3.13 confirms our previous results that CRBM is the most efficient algorithm in the long run. As mentioned earlier, since some parameters are fixed in the realistic dataset, i.e., N and σ and others are not known to us, i.e., δ and ρ, we could only investigate the range R and header size H_s. The instability in the CRBM graph at round number 5 happened because the value of many sensors in the dataset significantly change between round number three and round number four then the change happens again in the negative direction between round number four and round number five. This behavior confuses the CRBM algo-

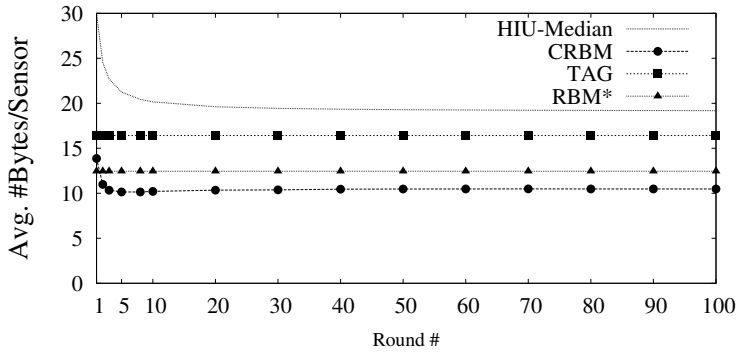


Figure 3.14: Comparison between HIU-Median and CRBM using the default values

rithm and makes unable to use cached results in round number five which lead to an increase of the average performance. Although this behavior happens frequently in the dataset, once the CRBM algorithm runs enough rounds, these expensive rounds will not have a significant impact on the amortized average.

A Comparison between HIU-Median and CRBM

Figure 3.14 shows the performance of TAG, RBM*, HIU-Median, and CRBM using all default values. The amortized average of sent bytes for CRBM is the best among all algorithms. However, the cost of each round is not stable and consistent like HIU-median's. CRBM outperforms HIU-median for two reasons: (1) The distribution of sensor's values has little influence on the CRBM's performance (Figure 3.11(c)) comparing to HIU-Median's Figure 3.8(c), (2) CRBM sends less messages than HIU-Median thus a large header size does not influence its performance. Figure 3.15 shows HIU-Median best case when the values distribution is uniform and the header size is zero. Although HIU-Median outperforms both TAG and RBM*, it is still worse than CRBM.

It is worth mentioning that RBM* and CRBM are expected to use more time to compute the median value since it sends several refinement queries every round. If the epoch between each two consecutive rounds is relatively small, CRBM may not have enough time slots to compute the median. In this case, we may need to combine HIU-Median and CRBM, for example, by using one or two refinements and then collect all values or use the CRBM technique to build the histogram query

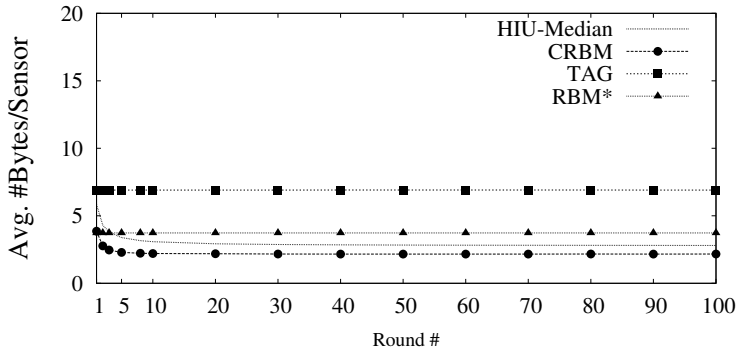


Figure 3.15: Comparison between HIU-Median and CRBM using the best case scenario for HIU-Median

for the HIU-Median.

3.6 Conclusion

In this chapter we proposed algorithms to answer snapshot and continuous *Median* queries in WSNs. RBM is a new in-network algorithm to compute a snapshot quantile answer, e.g., Median, efficiently. The basic idea is to use several refinement queries, each one being itself a *Histogram* query, in order to refine the range where the median value resides, without necessarily retrieving all values to the base station. Such refinements are done until it becomes cost effective to retrieve a relatively small set of candidate values from the network. RBM also relies on a heuristic rule that allows the size of each refining histogram to be determined automatically, based on the obtained values thus far in the refinement process. In the RBM algorithm, each node decides to either send a histogram answering the refinement query or sending all values relevant to the asked query.

The RBM algorithm can be used to compute an approximate answer for the *Median* query with a guaranteed error bound. The base station continues sending a refinement query that exponentially decreases the range of the median answer until the range is very small and satisfies the given error bound or an accurate answer is computed. Using the amortized average of bytes sent per node as the performance metric, the RBM algorithm needs up to 50% less traffic than TAG, and on average generates about 25% more than the minimum bound of 10 bytes per node per round.

CRBM is an extension of the RBM* algorithm and was designed to compute a continuous quantile answer efficiently. The main difference between RBM and CRBM is the histogram query construction. In the RBM algorithm we discussed the structure of the histogram query. In CRBM, we aim to decrease the continuous query cost by minimizing the amount of update messages from one round to the next using similar queries to ones already sent in the previous round. The main idea is very similar to the HIU algorithm in Chapter 2. We achieve this by using similar queries to ones already sent in the previous round. This allows the embedded HIU algorithm to efficiently compute the histogram and avoid sending unnecessary messages.In the first round, CRBM performs a little bit worse than RBM. However, the cost of the consecutive rounds is significantly cheaper because nodes depend on the cached answers from the previous rounds and send updates only. The CRBM uses almost the same number of bytes as the minimum bound of 10 bytes per node per round.

Using the average number of bytes sent per node as the performance metric, our experiments show that RBM* algorithm can generate up to 50% less traffic than TAG, and generates on average about 25% more than the minimum traffic possible, i.e, each node sends only its own observed value. In the first round, CRBM performs a little bit worse than RBM*. However, the cost of the consecutive rounds is significantly cheaper because nodes do not send their local histogram answer from scratch, instead they send updates messages to the cached ones. In all figures, CRBM shows that it uses almost the same number of bytes as the minimum bound of 10 bytes per node per round (2 for the sensor value and 8 for the header size).

Chapter 4

Conclusion and Future Work

In this thesis, we addressed the problem of in-network *Histogram* and *Median* query processing in the context of WSNs. We started our research by exploring the *Histogram* query. We could not find an efficient algorithm to compute an answer for the *Histogram* query since Madden et. al. proposed the TAG algorithm [17] in 2002. We proposed the HIU algorithm that is capable of efficiently computing the answer for a continuous *Histogram* query in Chapter 2. The main idea is to use in-network aggregation and in-node caching to reduce the energy consumption by sending updates to the prior round answer instead of sending the whole answer again. On average, HIU multiplies the network lifetime about three times comparing to TAG algorithm.

In the same chapter, Chapter 2, we showed that a histogram is capable of providing approximate answers for some other aggregate queries. Moreover, accurate answers are applicable with a small overhead. HIU also outperforms the TAG algorithm to answer other queries such as the *Max* query if the amount and/or probability of values change is reasonably small as observed in the real datasets.

Subsequently, our investigation moved beyond the *Histogram* query to the quantile queries for instance, *Median* . We proposed a few algorithms for the snapshot and continuous *Median* queries. In Chapter 3, we proposed a median version of HIU, HIU-Median algorithm, that computes the accurate median. This Algorithm sends less bytes than TAG but uses more messages. Its performance is worse than TAG if the header size is not very small.

The RBM is a new in-network algorithm to compute a snapshot quantile answer,

e.g., Median, efficiently. The basic idea is to use several refinement queries, each one being itself a *Histogram* query, in order to refine the range where the median value resides, without necessarily retrieving all values to the base station. Such refinements are done until it becomes cost effective to retrieve a relatively small set of candidate values from the network. RBM also relies on a heuristic rule that allows the size of each refining histogram to be determined automatically, based on the obtained values thus far in the refinement process.

We can use the RBM to answer continuous queries by repeating the algorithm every round, e.i. the RBM* algorithm. However, we proposed the CRBM algorithm as an extension for the RBM* algorithm. It is designed to compute a continuous quantile answer efficiently. The main difference between RBM and CRBM is the construction of the first refinement query in each round. Instead of initiating a new histogram query every round assuming no prior information available, CRBM uses the previously answered histogram queries to construct the first refinement query of each round.

The CRBM algorithm uses the same amount of bytes as the minimum bound when all nodes can connect directly to the base station and no in-network aggregation is required. In this case, every node should send its value to the base station. The minimum bound is the amount of required bytes for the sensor's value plus the header size of the message.

Our work in this thesis addressed some important problems in query processing within the context of WSNs. The solutions presented in this thesis are simple yet efficient as shown by the extensive experimental studies. These solutions are applicable for other problems in the same context where the communication is the main cost, e.g. wireless devices. We believe the following problems would be interesting to be addressed in future research.

Mining Data Streams

Mining data streams is concerned with extracting knowledge structures represented in models and patterns in uninterrupted streams of data [7]. There are many algorithms in the literature designed for pattern discovery from sensors [15, 21, 31].

Although most of these publications were discussing raw data mining, i.e., sensors' values pattern discovery, we believe extensions could be proposed to discover patterns from aggregated results for a Median query. For example, in the CRBM algorithm, we proposed building the histogram query every round based on the previous cached queries. However, after many rounds, the base station would have enough data to explore and analyze. Mining the computed median values collected every round from the WSN can lead to a more efficient histogram query every round.

Moreover, in Figure 3.3, we show the cost of each round per node. Although we suggested using a refinement query in the middle, i.e. not the first nor the last, we could not ensure which one should be used. Analyzing the median and histogram results every round in the base station can help making a better decision on which refinement query to start with.

Realistic Network Conditions

In our study, we assumed the communication between all nodes are perfectly reliable (i.e, there is no link failure), and concentrated on the query processing aspect of the problem. In reality, this is not true.

HIU, RBM, and CRBM all depend on the in-network caching and in-network aggregation. Nodes prefer to send updates to the cached results instead of sending the full result. This makes the unreliable communication vital because a missed message may affect the result of all the following rounds not only the current one. The more changes in the sensor values the higher the probability of having a wrong answer and the more changes to be communicated. Note that nodes applying the HIU or CRBM algorithm prefers to send the whole messages if more than 50% of the bins in the histogram change. This means, the more changes in the network, the higher the probability of sending a full message that does not depend on any previous message.

Different Routing Trees

The behavior of theCCR06 proposed algorithms depend on the node position in the routing tree. In our analysis we used the common shortest path tree (SPT) as a

routing tree. Recently, some publications show that using other trees could change the behavior of some algorithms. In [29], the authors proposed the Dominating Set Tree (DST). They show that using the DST tree instead of SPT enhance the performance of the TAG in-network algorithm to compute MAX query by substantial margins. It reduces the transmission cost by up to 70% and reduce the overall energy consumption by up to 53%. Although this tree does not offer the minimum distance between any node and the base station, as SPT, it minimizes number of intermediate nodes and then allow more aggregation in each intermediate node. We would like to explore using this tree for our algorithms. It is expected to enhance the overall performance and decrease both the transmission and energy consumption cost.

In [20], the authors proposed the BIased SPT (BISPT) logical tree to be used for one-to-All broadcasting. The BISPT is a logical tree with a smaller set of intermediate nodes, which result to a lower transmission cost for broadcasting. Clearly this will reduce the query processing cost for solutions that require broadcasting as a mandatory part of the algorithms. Some of our proposed algorithms, RBM and CRBM, send several refinement queries and will benefit from such an efficient tree.

Nodes Scheduling and Synchronization

A WSN node does not consume energy only for sending bytes, but also for receiving or even waiting for receiving a message. This overhead was ignored in our analysis. In the TAG algorithm each node will listen just once per round but in RBM (or CRBM) each node will listen multiple times. In the RBM (or CRBM) algorithm, each node should open its radio multiple times to accommodate all possible refinements. If there is no good synchronization protocol between nodes, a node may have to wait for a message for long time which may impact the overall performance. In this context a venue for future research is to explore how to add a good synchronization protocol to the RBM (or CRBM) algorithm.

Bibliography

[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] J. Considine, M. Hadjieleftheriou, F. Li, J. Byers, and G. Kollios. Robust approximate aggregation in sensor data management systems. *ACM ToDS*, 34(1):Article 6, 2009.

[6] L. P. Cox, M. Castro, and A. Rowstron. Pos: A practical order statistics service for wireless sensor networks. In *Proc. of ICDCS*, pages 52–64, 2006.

[7] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34:18–26, June 2005.

[8] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proc. of PoDS*, pages 275–285, 2004.

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

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

[11] 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.

[12] T. Kohonen. *Self-Organizing Maps*. Springer Berlin, 2001.

[13] Fabian Kuhn, Thomas Locher, and Roger Wattenhofer. Tight bounds for distributed selection. In *Proc. of SPAA*, pages 145–153, 2007.

[14] X. Y. Li, Y. Wang, and Y. Wang. Complexity of data collection, aggregation, and selection for wireless sensor networks. *IEEE TC*, 60(3):386 – 399, 2010.

[15] H. Liu, Y. Lin, and J. Han. Methods for mining frequent items in data streams: an overview. *Knowledge and Information Systems*, 26:1–30, Jan 2011.

[16] K. Liu, L. Chen, M. Li, and Y. Liu. Continuous answering holistic queries over sensor networks. In *Proc. of IPDPS*, pages 1–11, 2008.

[17] 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.

[18] 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.

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

[20] Baljeet Singh Malhotra. Efficient and reliable in-network query processing in wireless sensor networks. *PhD thesis, Computing Science Department, University of Alberta*, 2010.

[21] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. *Proc. of VLDB*, pages 560–571, 2003.

[22] B. Patt-Shamir. A note on efficient aggregate queries in sensor networks. *Theor. Comput. Sci.*, 370(1-3):254–264, 2007.

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

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

[25] R. Prakash, E. Nourbakhsh, and K. Sahu. Data aggregation in sensor networks: No more a slave to routing. In *Proc. of Allerton*, pages 1452–1459, 2009.

[26] 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.

[27] 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.

[28] M.H. Thanh, K.Y. Lee, Y.W. Lee, and M.H. Kim. Processing top-k monitoring queries in wireless sensor networks. In *Proc. of SENSORCOMM*, pages 545–552, 2009.

[29] P.J. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9:141–149, 2004.

[30] M.P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51(1):59–64, 1997.

[31] W. Wu and L. Gruenwald. Research issues in mining multiple data streams. In *Proc. of StreamKDD*, pages 56–60, New York, NY, USA, 2010. ACM.

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