Continuous Quantile Query Processing in Wireless Sensor Networks

Johannes Niedermayer[1], Mario A. Nascimento[2], Matthias Renz[1],
Peer Kröger[1] and Hans-Peter Kriegel[1]
[1]Institute for Computer Science, LMU, Munich, Germany
[2]Dept. of Computing Science, University of Alberta, Canada
{niedermayer, renz, kroeger, kriegel}@cip.ifi.lmu.de
mario.nascimento@ualberta.ca

ABSTRACT

A major concern when processing queries within a wireless sensor network is to minimize the energy consumption of the network nodes, thus extending the networks lifetime. One way to achieve this is by minimizing the amount of communication required to answer queries. In this paper we investigate exact continuous quantile queries, focusing on the particular case of the median query. Many recently proposed algorithms determine a quantile by performing a series of refining histogram queries. For that class of queries, we recently proposed a cost-model to estimate the optimal number of histogram buckets within an algorithm for minimizing the energy consumption of a query. In this paper, we extend that algorithm for continuous queries. Furthermore we also offer a new refinement-based algorithm that employs a heuristic to minimize the number of message transmissions. Our experiments, using synthetic and real datasets, show that despite its theoretical runtime complexity our heuristic solution is able to perform significantly better than histogram-based approaches.

1. INTRODUCTION

Wireless Sensor Networks (WSNs) are usually defined as large-scale, wireless, ad-hoc, multi-hop unpartitioned networks of homogeneous, small, static nodes deployed in an area of interest [25]. A single sensor node consists of one or more sensors, e.g., for temperature, acceleration, and light intensity, in combination with a microprocessor, a small amount of memory and a radio transceiver. By using these components, a node can take measurements of its surroundings, derive further information from the collected data, and send the resulting values to a root node via a path of adjacent nodes. Applications of WSNs include monitoring volcano activity [29], building structures [12] or natural habitat monitoring [18].

The lifetime of WSNs is typically limited by the amount of energy each individual node requires for its corresponding function, because sensor nodes usually rely on battery power. Due to several practical reasons e.g. remote loactions, often batteries cannot be easily replaced or recharged. Hence, the energy consumption of each node must be reduced as much as possible in order to increase the lifetime of the entire network.

Aggregate queries, e.g., *average*, *minimum*, *maximum* and *count* are useful when analysing large quantities of data, which is something a WSN is expected to produce over time. In-network aggregation can be optimized to decrease the energy consumption of such types of aggregate queries in a WSN [17]. Essentially, such optimizations avoid sending all values to the sink (root) node. However in order to solve more complex aggregate queries such as finding the *median* or *quantiles* of the node values, it is still necessary to transmit all values to the root node and compute the result in a centralized manner.

This class of queries are particularly useful in the sense that they are robust with respect to outliers and/or eventually some missing values. Due to the queries robustness to outliers, the effect of defective nodes producing incorrect measurements is mitigated. In contrast, aggregates such as the *average* can lead to siginificantly different results in the presence of outliers. Exemplarily, in a set of values $3, 3, 3, 3, 103$ with 103 representing an outlier, the median query would return 3, while the average would be 23. The effect of missing values is quite interesting when considering WSN data, as some observed data values may eventually not be transmitted due to link failure.

This paper addresses the *median* and more generally the *quantile* query in hierarchical WSNs, i.e., computing a quantile of all sensor values in the network where the network topology is given by a tree. The WSN's base station, i.e., where the query's answer is expected, serves as the tree's root node. The contributions of this paper are as follows:

- We extend our recently proposed cost model-based algorithms for snapshot quantile queries [21] to solve *continuous* quantile queries .
- We propose, as a main contribution, a heuristic algorithm that exploits the temporal correlation of values for reducing the number of refinement rounds.
- We show that our heuristic solution typically performs much better than histogram-based approaches, often consuming significantly less energy than theoretically optimal solutions, while still retrieving correct results.

10.5441/002/edbt.2014.24

The remainder of this paper is organized as follows. Sections 2 and 3 introduces basic concepts and discusses related work concerning quantile queries in WSNs. Section 4.1 extends the cost model-based algorithm from [21] to continuous queries. In Section 4.2 we propose our heuristic approach that exploits the temporal correlation of consecutive quantiles. An extensive performance analysis in Section 5 evaluates our results. Section 6 concludes this paper.

2. PROBLEM DEFINITION

A WSN can be modeled as a graph $G_p = (N \cup \{\mathfrak{r}\}, E_p)$, where the set of vertices is composed of the set of all sensor nodes, N, and the (distinguished) root node (sink) $\mathfrak{r}$. The set $E_p = \{\{n_i, n_j\} | dist(n_i, n_j) \leq \rho \wedge n_i, n_j \in (N \cup \{\mathfrak{r}\}) \wedge n_i \neq n_j\}$ describes the radio connections between nodes, with ρ being the radio range of all nodes and $dist(n_i, n_j)$ being the distance between n_i and n_j, e.g. the Euclidean distance. We further assume that each $n_i \in N$ can reach $\mathfrak{r}$ over multiple hops, i.e., there exists at least one path from each node n_i to $\mathfrak{r}$. G_p describes the physical neighborhood of nodes. However, since we will address the problem of quantile queries in *hierarchical* WSNs, the set of physical connections E_p has to be reduced to a set of logical connections $E_l \subseteq E_p$ such that $G_l = (N \cup \{\mathfrak{r}\}, E_l)$ is an acyclic connected graph. Furthermore we assume the simplification that messages can be transmitted reliably from node to node. The root node $\mathfrak{r}$ has a special status since it has access to an infinite energy supply and additionally does not have any sensor for measuring specific parameters, i.e., measurements are only taken by nodes $n_i \in N$. For clarity reasons, we assume that each node n_i measures a single value $v_t(n_i)$ at each discrete time stamp t. An extension of the concepts proposed in this paper to nodes producing multiple values at a time is trivial since additional values could be interpreted as received from artificial child nodes.

Let us now introduce the quantile query:

Definition 2.1 (ϕ-quantile query). *Given an input parameter $\phi \in]0, 1[$, a ϕ-quantile query returns the element with rank $k = \lceil \phi |N| \rceil$.*

The *median query* is the special and most important case of a quantile query where $k = \lceil |N|/2 \rceil$. However, the solution we present is in fact independent of the value of k, i.e., it can be used to solve the more generic quantile queries as well by simply setting k to the desired value.

A continuous ϕ-quantile query continuously updates a quantile over time. We exploit this continuity property, aiming at reducing the overall energy consumption of the network since quantiles are often temporally correlated.

A table containing the notation used in this paper can be found in Table 1.

Table 1: Table of Notation

N	Set containing all network nodes, without root node $\mathfrak{r}$.
$\|N\|$	Number of nodes in the network.
n_i	A network node, $n_i \in N$.
$\mathfrak{r}$	The root node; $N \cap \{\mathfrak{r}\} = \emptyset$.
$\phi \in]0, 1[$	The value of the ϕ-quantile.
k	$k = \lceil \phi \|N\| \rceil$
$r_{min} < r_{max}$	Minimum and maximum *possible* value.
r	Number of values in an integer range, $r = r_{max} - r_{min} + 1$.
v_{min}, v_{max}	The smallest (largest) *existing* node measurement.
t	A discrete time stamp.
$v_t(n_i)$	The measurement (value) of node n_i at time t.
v_k^t	The resulting ϕ-quantile (k-th value) at time t.
B_i	Bucket i of a histogram.
$[B_i.lb, B_i.ub[$	Range of bucket B_i, $B_i.lb, B_i.ub \in \mathbb{N}$.
$B_i.count$	Counts the number of measurements in $[B_i.lb, B_i.ub[$.
b	The number of buckets in a histogram.
b_{opt}^{exact}	The optimal number of buckets.
b_{opt}^{lb}	Approximation of b_{opt}^{exact}.
s_h	Header size of a message $[bit]$.
s_p	Payload size of a message $[bit]$.
s_b	Size of a bucket $[bit]$.
s_r	Size of a refinement payload $[bit]$.
s_v	Size of a measurement $[bit]$.
s	Size of a message, $s = s_h + s_p$ $[bit]$.
l, g, e	State variables (counters) for POS, HBC, and IQ.
lt, eq, gt	State variables (intervals) for POS, HBC, and IQ.
$\mathfrak{a}, \mathfrak{b}$	Counters for the IQ approach.
A, R	IQ: Sets transmitted during validation (A) and refinement (R).
Ξ	IQ:The interval where values are transmitted directly.
ξ_l, ξ_r	Offset of the bounds of Ξ from v_k^{t-1}.

3. RELATED WORK

3.1 Overview

Algorithms for finding quantiles and medians in WSNs can be classified as approximate solutions, probabilistic algorithms, and exact methods. Since this work addresses exact solutions for computing quantiles, we will concentrate on exact solutions.

Approximate algorithms such as [26, 6, 10, 8, 16, 24] return quantile estimates within user-defined error bounds, typically by creating quantile summaries. Such quantile summaries can only be computed if the resulting value does not have to be the exact quantile because an accurate quantile summary will always contain all values, which is clearly not efficient.

Probabilistic algorithms do not provide strict error bounds, instead they guarantee that the result is within a given error bound with a given probability. One method for computing probabilistic quantiles is sampling [4]. For WSNs, specialized sampling algorithms such as described in [1, 5, 14] can be applied. Shamir [22] used approximate counting to compute a probabilistic median. The authors of [30] employed order statistics to estimate quantiles. Note that exact solutions can usually be made probabilistic by querying only a subset of nodes, e.g. by employing a layered architecture as described in [28].

TAG [17] provides the possibility to compute the exact

median with a worst-case complexity in per-node transmitted values of $O(|N|)$. For such a query all values are collected by the root node and the median computation is performed centrally. In [23], the authors suggested a specialized routing tree that is built according to the value distribution in the network in order to improve the performance of queries like *min*, *max*, *top-k* queries and the *quantile* query. Greenwald and Khanna [10] handle the case where an exact quantile has to be determined by applying an extension to their approximate quantile summary algorithm. They solve the given problem by transmitting $O(\log^3(|N|))$ values.

Many existing solutions employ histograms with b buckets, however none of those employs cost models to estimate the optimal number of buckets, as it is done in our approach. For instance, in [22] and POS [9] the authors propose performing a binary search, which is the special case of $b = 2$, leading to $O(\log_2(r))$ refinement iterations where r denotes the universe of all possible values. The authors of [19] also propose to perform a binary search; their continuous solution is similar to POS, however similar to our Interval-based Quantiles (IQ) algorithm the number of refinement iterations is reduced to one. However in contrast to this solution we aim at completely avoiding refinements by employing heuristics. Liu et al. [16] suggested to determine b by taking the message size into account; the number of refinements in this scenario drops to $O(\log_b(r))$ with one of their refinement strategies. Further publications [3, 2] use techniques similar to ours to compute quantiles in WSNs, however they neither use a formal cost model nor investigate the performance of histogram-based approaches in realistic scenarios, comparing to heuristic solutions; indeed we will show in this paper that heuristic solutions are often the best choice. Kuhn et al. [14] do not equidistantly divide the universe of possible values into b subintervals but instead sample measurements from the network and count the number of measurements between two consecutive samples in the received ordered list of values. The number of refinements with this approach does not depend on r, however it depends on the number of measurements in the network, i.e., $|N|$, and, therefore, the number of refinements on average becomes $O(\log_b(|N|))$. Since this algorithm is based on a snapshot scenario, we will not further address it in this paper. Finally, the authors of [15] investigated the complexity of median queries in terms of time, message and energy complexity by employing the algorithms of Kuhn et al.

3.2 POS

Both our suggested algorithms reuse some of the research efforts made in [9]. Since the knowledge of this algorithm is fundamental for understanding our continuous algorithms, we will review it within this section.

POS is a binary-search-based continuous approach that uses the most recently computed quantile as a filter to reduce the number of message transmissions during the next round.

During the *initialization* round $t = 0$, POS computes the first quantile by using an aggregation technique equivalent to TAG [17], i.e., all measurements are forwarded to the root node. After the root node computes the quantile value v_k^t, this value is broadcasted into the tree in order to provide all nodes with a filter for the next round. Complementary to v_k^t the root node stores some additional state information after each round t, namely the number of values greater than, equal to, and smaller than v_k^t; we denote them as g, e, and l respectively. Hence, in the beginning of a round t, the root node knows the intervals $lt_{t-1} =]-\infty, v_k^{t-1}[$, $eq_{t-1} = [v_k^{t-1}, v_k^{t-1}]$, $gt_{t-1} =]v_k^{t-1}, \infty[$ and additionally the outdated number of measurements in each of these intervals. Since v_k^t is known to each node at the end of a round, every node can compute these intervals as well, but e, g and l are not known to them.

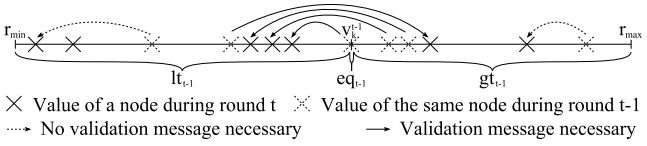

Figure 1: Validation - which nodes should send validation packets? Arrows describe the motion of a value and additionally indicate if a validation message has to be sent or not.

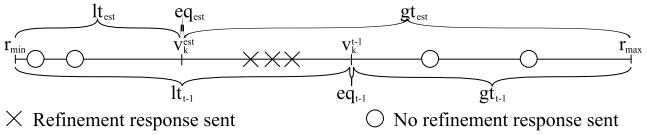

Figure 2: Deciding whether or not a node has to respond to a refinement request.

The following rounds are split up into two stages, *validation* and *refinement*. *Validation* is performed at the beginning of each round t. Starting at the leaf nodes, each node checks if the interval containing the new reading is different from the interval that contains the reading of round $t-1$ (see Figure 1); in this case the quantile value might change and therefore a validation message is necessary.

Validation messages contain four counters describing the movement of values, $into_{lt}, into_{gt}, outof_{lt}, outof_{gt}$ and therefore they can be easily merged by intermediate nodes in a TAG-like manner by simply adding up corresponding counters. The root node can use the received counts to update its state variables $g = g - outof_{gt} + into_{gt}$, $l = l - outof_{lt} + into_{lt}$ and $e = |N| - g - l$. If g and l are still correctly distributed — for the median it is necessary that $g \leq \frac{|N|}{2} \wedge l \leq \frac{|N|}{2}$ — then v_k^{t-1} is still a valid quantile. However, if g or l becomes too large, the *refinement* protocol has to be executed. This is done by performing a binary search in the range $[v_k^{t-1}, \infty]$ if the new quantile is greater than the last one or in the range $[-\infty, v_k^{t-1}]$ if the new quantile is less than the last one. The root node broadcasts the midpoint of the refinement interval which is interpreted as the new quantile v_k^{est} by each node. If the value of a node switched its interval, e.g. if it moved from lt_{t-1} to gt_{est}, but not if it moved from gt_{t-1} to gt_{est}, a message equivalent to the ones transmitted during validation is sent (see Figure 2). The root node can then update g, l and e again and stop if v_k^{est} is a valid quantile; in this case $v_k^t = v_k^{est}$. If it is not, another refinement has to be performed that halves the range of possible values again. Note that the new quantile does not have to be broadcasted in this scenario since it is already known to each node from the last refinement.

Cox et al. introduced some additional improvements as well. First of all, the refinement bounds ∞ and $-\infty$ are by far not optimal. However an additional variable *hint* can

be transmitted in validation messages that bounds the new quantile better than $-\infty$ or ∞; for the calculation of the hints we refer to the original paper. This hint replaces ∞ and $-\infty$ during refinement which can significantly reduce the length of the refinement interval and therefore reduce the number of refinements.

It is also possible to avoid further refinements if the number of values in the refinement interval becomes low. In this case, POS requests all values in the remaining interval directly. Note that with this improvement a final broadcast becomes necessary to update the filter of all nodes.

4. ALGORITHMS

Next we describe the two algorithms for continuous quantile queries proposed in this paper. The first one, based on histograms is asymptotically optimal under the (simplifying) assumptions made, while the second one, a heuristic solution, performs better in realistic settings.

4.1 A Histogram-Based Algorithm

As we have already shown, POS performs a binary search in the determined refinement interval. However, as we have shown in [21], a binary search is non-optimal from an energy consumption point of view. Therefore, in this section, we apply the cost model proposed in [21] to POS, improving the efficiency of that approach.

The cost model proposed in [21] generally aims at reducing the sending energy of hotspot nodes. To do so, the cost model computes a lower bound of the optimal number of buckets as $b_{opt}^{lb} = e^{W(\frac{2 \times s_h + s_r}{s_b \times e})+1}$, with s_h the size of header and footer of a message, s_r the size of a refinement request, s_b the number of bits used to encode a bucket, and $W(x)$ the Labert W function [7]. While this cost is only a lower bound of the actual cost, experiments have shown that this approximation already leads to a good estimate of the optimal number of buckets. The solution for finding the exact number of buckets can be found in [21].

In [21] we followed the assumption that the root node does not have any previous knowledge of the value distribution. However, it is often suitable to perform an initial query, e.g. equivalent to the one in [21] and then, during consecutive query rounds, use the knowledge of the last round to tighten the search range. Therefore we suggest the Histogram Based Continuous (HBC) algorithm, for performing continuous quantile queries. It is based on both POS [9] and the cost-model based approach from [21].

4.1.1 The HBC Algorithm

After the initial query, the root node broadcasts the calculated k-th value v_k^{t-1} into the network as suggested by POS (see Section 3.2). As we assume temporal correlation between consecutive rounds, this value is also a good prediction of future quantiles. Therefore, this value is used by network nodes as a filter during the next round. The validation convergecast in the beginning of the next round t uses this filter and is equivalent to POS [9] validation (see Section 3.2), counting values that have been larger (smaller) and are now smaller (larger) than the filter v_k^{t-1}. After validation, the root node knows a *hint* and the updated variables g, l, and e. Based on g, l, and e, the root node can decide whether the new value of the quantile is greater than, equal to, or less than the old value. If $v_k^t = v_k^{t-1}$, no refinement is necessary. Otherwise, the root node can compute an interval $[lb, ub[$ that contains the k-th value. This interval is either $[hint, v_k^{t-1}[$ if $v_k^t < v_k^{t-1}$ or $[v_k^{t-1}+1, hint+1[$ if $v_k^t > v_k^{t-1}$. The root broadcasts a refinement request containing this interval into the tree. Based on this refinement request, each node n_j divides the range $[lb, ub[$ received from the root into b buckets and sort its value $v(n_j)$ into the histogram bucket B_i if $B_i.lb \leq v(n_j) < B_i.ub$ holds. A histogram is then transmitted to the respective parent node where histograms are aggregated until they reach the root node. Given the aggregated histogram and the variables g, e, l from POS, the root node can compute the bucket B_j that contains the new k-th value; the calculations can be easily derived from POS. B_j is then refined iteratively equivalent to the non-continuous solution from [21] by sending a refinement request containing the bounds $[B_j.lb, B_j.ub[$ into the network and receiving the corresponding histogram from the network. The refinement finishes if $B_j.ub - B_j.lb = 1$; in this case the quantile is uniquely determined. Finally, the root node broadcasts the new value of the quantile, v_k^t, into the tree if $v_k^t \neq v_k^{t-1}$, in order to update the filter of network nodes.

Note that within our implementation, we did not recompute b during each round since we observed that the difference in performance was marginal. Furthermore, the improvements proposed in [21] can also be applied to this continuous algorithm, such as compressing histograms by removing empty buckets and sending values directly if the refinement interval is nearly empty.

4.1.2 Eliminate Threshold Broadcasts

Our continuous approach (but not POS) always has to broadcast the newly determined k-th value if it differs from the quantile determined during the last round. In order to eliminate this additional broadcasting, the following extension can be applied.

Equivalent to POS, a root node using the continuous histogram based algorithm stores the three variables l, e and g. These variables count the values in the intervals $]-\infty, v_k^t[$, $[v_k^t, v_k^t]$ and $]v_k^t, \infty[$, respectively, after round t. Since v_k^t is not known to each node it has to be broadcasted. However, during every refinement the bounds of the refined bucket $B_i.lb$ and $B_i.ub$ are broadcasted into the tree anyway, and they restrict the position of the k-th value nicely. In order to avoid the threshold broadcasting, the intervals can be altered to $]-\infty, lb[$, $[lb, ub[$ and $[ub, \infty[$ where lb and ub denote the lower and upper bound contained in the last refinement request. During validation, each node sends an update equivalent to POS if its value jumped from one of these intervals into another; the necessary adaptions to achieve this are trivial and the computation of the *hint* can be adapted as well. If $l \geq k$, then the interval containing the quantile is $[hint, lb[$. For the case that $l < k \wedge l + e \geq k$ the refinement interval is $[lb, ub[$, and the refinement interval $[ub, hint]$ is chosen if $l + e < k$.

In addition to the eliminated threshold broadcasting, this solution provides a good estimate if the k-th value changes only a bit during two consecutive rounds - in this case only the interval $[lb, ub[$ has to be refined. However, if the k-th value does not change during two consecutive rounds this solution will typically have to perform additional refinements in comparison to the basic solution. Also note that this extension can not be simply combined with the last one.

4.2 A Heuristic Algorithm

The algorithm suggested in the last section depends on the universe of values, yielding a worst-case complexity of $O(b * \log_b(r))$ per-node transmitted values. In this section we introduce a heuristic approach, Interval-based Quantiles (IQ), for continuous quantile queries that depends on the number of measurements $|N|$ with a *worst-case* complexity of $O(|N|)$ per-node transmitted values. Although this approach depends linearly on $|N|$, it can outperform POS and even HBC in many cases, especially if the observed quantile changes slowly over time. Due to the similar structure of POS, HBC and IQ it is possible to switch between these approaches without reinitializing the network and always use the best algorithm within a given environment, however we leave heuristics to select the best solution for future research.

The intention of this approach is to avoid the overhead for performing several refinements while heuristically minimizing the number of transmitted values at the same time. It increases the number of values transmitted during validation, and performs at most one refinement if the validation packet did not contain enough information to compute the exact quantile.

Equivalent to POS, IQ is based on the observation that a physical phenomenon often changes gradually over time, i.e., the results v_k^{t-1} and v_k^t of two consecutive rounds differ only slightly (granted the term "slightly" depends on the measured data). This observation is often refered to as temporal correlation. Given POS, if $v_k^{t-1} = v_k^t$ a refinement is not necessary, because the root node can derive that the quantile did not change by analyzing the information from received validation messages. However, if in an integer setting there is for example $v_k^{t-1} = v_k^t + 1$, POS will have to perform a refinement. Depending on the *hint*, determining the new k-th value can take several refinement rounds and these refinements increase the communication cost. To avoid these refinements, we define an interval $\Xi = [v_k^{t-1}+\xi_l, v_k^{t-1}+\xi_r], v_k^{t-1} \in \Xi$ where values are transmitted directly. The values ξ_l and ξ_r are determined at runtime. If Ξ is chosen wisely, the new k-th value is contained in Ξ, but Ξ is also small enough to avoid overhead for unnecessarily transmitted values. If $v_k^t \in \Xi$, a refinement can be avoided since the value is transmitted to the root node during validation. If $v_k^t \notin \Xi$ a refinement is necessary, but in order to finish the refinement as fast as possible we send values directly during refinement as well, i.e., a round finishes after at most two convergecasts.

A more formal description of the algorithm can be found in Algorithm 1. For the sake of brevity we constrain this formal description to the control loop at the root node; the behaviour at the network nodes is straightforward.

Algorithm 1 IQ-Algorithm at the Root-Node

```
Perform initialization of g, l, and e equivalent to POS; initialize ξ
Disseminate (v_k^0, ξ) into the network
for each round t do
  Receive validation messages from children
  Update e, g, l and hint equivalent to POS
  Compute the set of values in Ξ received by child nodes, i.e. A = ∪_i A_i, sorted in ascending order
  if l < k ∧ l + e ≥ k then
    v_k^t = v_k^{t-1}
  end if
  if l ≥ k then
    𝔞 = |{x ∈ A | x < v_k^{t-1}}|
    if l − 𝔞 < k then
      v_k^t = A[𝔞 − (l − k) − 1]
      e = |{A[i] | A[i] = v_k^t}|
      l = k − |{A[i] | i < 𝔞 − (l − k) − 1 ∧ A[i] = v_k^t}|
      g = |N| − l − e
    else
      {l − 𝔞 ≥ k}
      Refinement: request the f_l = l − k − 𝔞 largest values in ]−∞, v_k^{t-1} + ξ_l[
      v_k^t = f_l-th largest value in the set R of received refinement responses
      e = |{x ∈ R | x = v_k^t}|
      l = k − (|R| − f_l + 1)
      g = |N| − l − e
    end if
  else
    if l + e < k then
      𝔟 = |{x ∈ A | x > v_k^{t-1}}|
      if l + e + 𝔟 ≥ k then
        v_k^t = A[|A| − l − e − 𝔟 + k − 1]
      else
        {l + e + 𝔟 < k}
        Refinement: request the f_g = k − l − e − 𝔟 smallest values in ]v_k^{t-1} + ξ_r, ∞[
        v_k^t = f_g-th smallest value in the set R of received refinement responses
        l = l + e + 𝔟 + |R| − |{x ∈ R | x = v_k^t}|
        e = |{x ∈ R | x = v_k^t}|
        g = |N| − l − e
      end if
    end if
  end if
  if v_k^t ≠ v_k^{t-1} then
    Disseminate v_k^t into the network
  end if
end for
```

4.2.1 Initialization

The initialization can be performed by using TAG or by using a histogram-based solution like the one described in [21]. After initialization, v_k^0 is broadcasted and g, e, l (see Section 3.2) are initialized at the root node equivalent to POS. The initialization algorithm is independent from our solution, therefore every solution that allows to compute v_k^0, g, l, and e is applicable. Since POS uses TAG during initialization, we will use the same algorithm.

Additionally to the POS state variables, the interval Ξ has to be initialized as well. Ξ will later represent the changing pattern of the underlying physical phenomenon, and therefore adapt to the behaviour of the k-th value. If there is an upward trend, then v_k^{t-1} will be close to the left interval bound of Ξ and if there is a downward trend v_k^{t-1} will be close to the right bound. But unfortunately during initialization nothing is known about the behaviour of the measured phenomenon, therefore we assume that $v_k^t \approx v_k^{t-1}$. In this case Ξ should not contain too much values – otherwise the validation would become very expensive –, but it should also contain at least some values in order to avoid later refinement. Since the root node knows at least parts of the value distribution when using TAG, or more precise at least the

k smallest values $[v_1, ..., v_k]$[1], it can use this information to initialize Ξ. We suggest either to compute $\xi = c * \frac{v_k - v_1}{k}$ (c is a constant to tweak the number of values transmitted during validation) or the median of distances between consecutive values. The latter one could be used to reduce the impact of outliers, for example when the physical phenomenon shows a normal distribution. These outliers would increase the length of Ξ such that validation becomes more expensive. For an initialization that is based on a b-ary search a technique similar to the average could be used by selecting a representative refinement interval and dividing its length by the number of candidates contained in the interval.

The filter broadcast after initialization contains a tuple (v_k^0, ξ). Receiving nodes store v_k^0 and set $\xi_l = -\xi$, $\xi_r = \xi$.

4.2.2 Update Rounds

Validation. During validation in round t the transmitted state variables $outof_{lt}$, $into_{lt}$, $outof_{gt}$, $into_{gt}$ are calculated equivalent to POS, but the validation packet contains an additional multi-set A containing all values in the interval $\Xi = [v_{t-1}^k + \xi_l, v_{t-1}^k + \xi_r]$, $\xi_l \leq 0, \xi_r \geq 0$. If a leaf node's ($n_i$) value falls into Ξ, but if $v_t(n_i) \neq v_k^{t-1}$, it adds its own value to A independent to whether or not it would have transmitted a POS validation packet. If at least some information has to be sent, a node sends a packet $(outof_{lt}, into_{lt}, outof_{gt}, into_{gt}, A)$ to its parent.

The multi-sets A_i received from children of an intermediate node can be merged by simply unioning these multi-sets; merging the remaining variables works equivalent to POS. Note that this technique leads to a worst-case complexity of $O(|N|)$ per-node transmitted values since in the worst case an intermediate node has to forward all values from its subtree.

Given the validation packets from its children, the root node can finally compute the global values for $outof_{lt}$, $into_{lt}$, $outof_{gt}$, $into_{gt}$, and A (and the global $hint$). It updates e, g, l and determines the interval that contains the new k-th value, i.e., $lt =]-\infty, v_k^{t-1}[$, $eq = [v_k^{t-1}, v_k^{t-1}]$ or $gt =]v_k^{t-1}, \infty[$ as described in Section 3.2. Finally, the root node can make a refinement decision and refinement similar to [19].

Refinement for $v_k^t \in eq$. If the k-th value did not change, i.e., $v_k^t = v_k^{t-1}$ and therefore $l < k \wedge l + e \geq k$, then no refinement is necessary, and v_k^t does not have to be broadcasted since it is already known to all nodes.

Refinement for $v_k^t \in lt$. If $v_k^t \in lt$, i.e., $l \geq k$, there are two cases possible. Let $\mathfrak{a} = |\{x \in A | x < v_k^{t-1}\}|$ denote all values in A that are smaller than the last quantile v_k^{t-1} (see Figure 3). Then if $l - \mathfrak{a} < k$ the new k-th value is contained in A. If measurements in A are sorted in increasing order, the k-th value can be determined by taking $A[\mathfrak{a} - (l-k) - 1]$. Let us assume without loss of generality that all values in A are distinct. Then $l - k$ describes all values $v_i \in]-\infty, v_k^{t-1}[$ with $v_i > v_k^t$, therefore $\mathfrak{a} - (l-k)$ denotes the values $v_i \in [v_k^{t-1} + \xi_l, v_k^{t-1}[$ with $v_i \leq v_k^t$. Since A's first element is indexed with 0, 1 has to be subtracted. Updating e, l, and g is straightforward.

The second case where $l - \mathfrak{a} \geq k$ is more costly since in this case the interval $]-\infty, v_k^{t-1} + \xi_l[$ has to be refined. The root node broadcasts a refinement request asking for the $f_l = l - k - \mathfrak{a} + 1$ largest values in $]-\infty, v_k^{t-1} + \xi_l[$ (or the interval $[hint, v_k^{t-1} + \xi_l[$, if POS-like hints are used). Again, $l - k$ describes all values $v_i \in]-\infty, v_k^{t-1}[$ with $v_i > v_k^t$. From these $l - k$ values $\mathfrak{a}$ are already known. Since we do not only want to retrieve the values $v_i > v_k^t$ but also v_k^t itself, we have to send another value.

This technique is based on the assumption that $v_k^{t-1} - \xi_l$ is already a good estimate of the new quantile, and therefore $f_l < k$. To indicates that the refinement has to be performed in the interval $]-\infty, v_k^{t-1} + \xi_l[$, $-f_l$ can be transmitted.

Given the refinement request, leaf nodes return their value if it is contained in $]-\infty, v_k^{t-1} + \xi_l[$ (or $[hint, v_k^{t-1} + \xi_l[$). Intermediate nodes add their value to the refinement response when necessary and remove all but the f_l largest values from the refinement response. Note that if non-unique values are allowed then all values equal to the f_l-th largest value in the refinement response have to be transmitted, in this case the refinement response might contain more than f_l values.

The root node can take the f_l-th largest from all received values R, this is the new k-th value v_k^t. It can also compute $e = |\{x \in R | x = v_k^t\}|$, $l = k - (|R| - f_l + 1)$ and $g = |N| - l - e$.

Refinement for $v_k^t \in gt$. If there is $l + e < k$, then the k-th value is contained in the interval $]v_k^{t-1}, \infty[$. The refinement works similar to the case above: let $\mathfrak{b} = |\{x \in A | x > v_k^{t-1}\}|$. Then if $l + e + \mathfrak{b} \geq k$ the new k-th value is contained in A. If measurements in A are sorted in increasing order, the k-th value can be determined by taking $A[|A| - l - e - \mathfrak{b} + k - 1]$.

However if $l + e + \mathfrak{b} < k$ a refinement has to be performed. The refinement requests contains a value $f_g = k - l - e - \mathfrak{b}$ and requests the f_g smallest values in the interval $]v_k^{t-1} + \xi_r, \infty[$ (or $]v_k^{t-1} + \xi_r, hint]$ for POS-like hints). Again, intermediate nodes forward only the f_g smallest values, remaining values are dropped by aggregating nodes.

The root node determines the f_g (and possibly more if non-unique values are allowed) smallest values R in the received set of values, sets the f_g-th smallest value as v_k^t, and sets the value of l to $l = l + e + \mathfrak{b} + |R| - |\{x \in R | x = v_k^t\}|$, $e = |\{x \in R | x = v_k^t\}|$ and $g = |N| - l - e$

Filter Broadcasting. If the k-th value changes between two consecutive rounds it has to be broadcasted since it can not be derived by nodes from refinement requests in a POS-like manner. Therefore the root node broadcasts v_k^t if $v_k^t \neq v_k^{t-1}$. Intermediate nodes and leaf nodes update their filter to the new quantile and use the received quantile to update ξ_l and ξ_r. Note that after round t the quantile v_k^t must be contained in $[v_k^t + \xi_l, v_k^t + \xi_r]$ and therefore it is necessary that $\xi_l \leq 0$ and $\xi_r \geq 0$. It would be possible to remove this constraint, however our research showed that, in our settings, this increases the complexity of the algorithm without a major change in performance, therefore this will not be addressed in this paper.

To compute Ξ, we suggest to store the m most recent k-th values and set

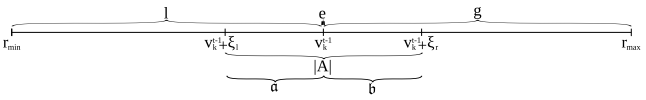

Figure 3: Notation used for explaining the validation and refinement phase of IQ. $\mathfrak{a}$, $\mathfrak{b}$, g, l, and e describe the number of values in a sub-interval of $[r_{min}, r_{max}]$.

[1]This is the case if $|N|$ is known to the root node before the algorithm executed. The basic TAG approach would transmit all $|N|$, and not only the k smallest values.

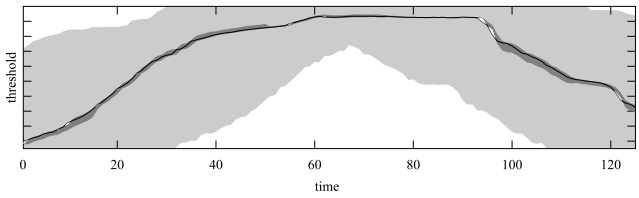

Figure 4: Development of Ξ and v_k^t Over Time

$$\xi_l = \min(\min_{i=t-m+2}^{t}\{v_k^i - v_k^{i-1}\}, 0)$$

$$\xi_r = \max(\max_{i=t-m+2}^{t}\{v_k^i - v_k^{i-1}\}, 0)$$

where m is a system tweaking parameter. This increases the absolute value of ξ_l if there is a downward trend, because in this scenario $v_k^i - v_k^{i-1}$ becomes negative. If there is an upward trend ξ_l becomes zero, because if there is an upward trend it does not make sense to transmit measurements with a value less than the old quantile. For ξ_r the behaviour is vice versa: With an upward trend the value of ξ_r increases, but with a downward trend it becomes zero.

Figure 4 visualizes the development of Ξ (dark grey area) and v_k^t (black line) in an air pressure data-trace over 125 rounds. The light grey area in the background denotes the range between the smallest and largest measurement in the network. White fields close to the black line denote refinements – they visualize the interval between Ξ and the new threshold. Usually no refinement is necessary to compute the new quantile. However after initialization (round 1), and if the underlying trend changes (for example round 95 in Figure 4), it takes some time until Ξ adapted to the altered conditions. Therefore, if there are short-lived trends, the number of refinements and therefore the energy consumption increases. Additionally the energy consumption of this algorithm depends on the number of values in Ξ. If Ξ encloses a lot of values, either because the length of Ξ becomes very large because the quantile fluctuates a lot, or because the density of values in Ξ is just high due to the underlying value distribution, the performance of this algorithm suffers as well.

5. PERFORMANCE EVALUATION

5.1 Test Setup

The performance analysis was performed by using a Java coded framework. Given a set of input variables, several *simulation runs*, all of them consisting of an equal number of *rounds*, were performed. Performance indicators such as the number of transmitted messages, the number of transmitted values, the maximum per-node energy consumption, and the network lifetime were calculated by computing the average of the given indicator over all rounds and simulation runs. For the sake of brevity, we will only focus on the network lifetime and maximum per-node energy consumption in this paper due to the optimization goals of our algorithms. During a simulation run all compared algorithms used the same physical and logical network topology. The topology was changed between two consecutive simulation runs. On real world data sets the topology was only changed by selecting another root node; all nodes' position was fixed. The synthetic dataset also enabled the possibility to reposition the nodes between two simulation runs.

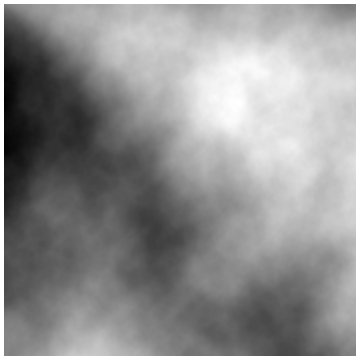

Figure 5: Synthetic Data: Interpolated Noise

5.1.1 Node Distribution

Nodes were distributed in a rectangular area according to the underlying dataset. After distribution, the physical neighbours of each node were computed by finding all nodes in a specified radius – the radio range ρ – in the neighbourhood of the node. This information was then used to create a routing tree by reducing the overall set of physical connections between nodes to a small subset of logical connections. For our simulations, we used a Shortest Path Tree. For each node it is only possible to send messages to its children or to its parent.

5.1.2 Synthetic Dataset

The synthetic dataset was created on the fly when performing a simulation run. Nodes were distributed in an area of 200m in width and length. For initialization of the values we used an image containing interpolated noise to simulate the spatial correlation of values in reality. An example of such interpolated noise can be found in Figure 5. Each node's position in the 200m×200m area was mapped to the corresponding coordinates in the picture, and the resulting greyscale value was used as the node's initial value. Because the input image only produced 256 different values, we added some additional noise by using a pseudo random number generator. The noise magnitude was set to less than $\frac{1}{256}$ of the image's range in order to keep the overall distribution of values in the image. The resulting greyscale value was then scaled to a given integer range.

For the continuous query setting we allowed some kind of trend, and therefore there had to be space to the upper and lower end of the domain such that values could increase or decrease over time. Therefore the initial range of values was set to about a third of the whole range during initialization.

The resulting data was used to initialize the nodes' values. For computing consecutive values, temporal correlation of values had to be taken into account. Temporal correlation is not unusual, e.g. when tracking temperature or air pressure over time. To simulate this behaviour, we used a sinusoidal function with additional noise. The noise was set to 10% of the range of the initial value distribution by default.

5.1.3 Real Datasets

Additionally we tested our algorithms with a more realistic pressure dataset from [27]. The pressure dataset was

derived from data collected by the Live from Earth and Mars project[2] by extracting data traces for 1022 nodes. Unfortunately this dataset did not contain information about the spatial distribution of the data. In order to create a realistic setting for this dataset where neighbouring nodes produce similar values, we used a self-organizing map (SOM) approach similar to [13]. Feature vectors of size one were used as input of the SOM, containing the first measurement of each node, and the output was the position of each node.

5.1.4 Cost Function

To determine the energy consumption of a node within a sensor network, we used a well known cost function that was for example used in [11]. It distinguishes between three different modes: sending a message, receiving a message and sleeping. The costs for sending a message is computed by the function $s * (\alpha + \beta \times \rho^q)$ where s denotes the size of the transmitted packet, α is a distance independent constant, β is a distance dependend constant multiplied by ρ^q, with ρ the radio range of a node. Equivalent to [11] we assumed $\alpha = 50nJ/bit, \beta = 10pJ/bit/m^2$ and $q = 2$. The energy consumption for receiving data was computed by the function $s * \gamma, \gamma = 50nJ/bit$. We set the energy consumption in sleeping mode to 0 since the sleeping cost depends highly on the underlying MAC layer, e.g. the scheduling strategy. The initial energy supply of each node was set to 30 mJ. The maximum payload size s_p^{max} was set to 128 bytes, the header length s_h was set to 16 bytes by default. These values were derived from, but simplify the IEEE 802.15.4 standard[3].

When sending a packet, the consumed energy is deducted from the sender's energy supply. If a node sends a message, energy is also subtracted from the receiving node(s) energy supply since we assume that due to a scheduling strategy each node knows when it might receive a message.

5.1.5 Performance Metrics

As performance metrics we employed the maximum per-node energy consumption on the one-hand side and the network lifetime on the other hand. The maximum energy consumption per node describes the energy of the node that consumes most energy, i.e. a hot spot node that runs out of power first. To measure the network lifetime we decided to measure the number of rounds until the first node runs out of energy. For additional experiments on the number of transmitted values and the number of transmitted messages we refer to [20].

5.1.6 Algorithm Setup

We are going to compare TAG [17], POS [9], and LCLL [16] with our solutions HBC and IQ. HBC has been introduced in Section 4.1, additionally we used the heuristic from [21] that retrieves values directly, and the improvements from Section 4.1.2. Instead of transmitting two hints containing the minimum and the maximum measurements that changed their state (see POS), we sent the maximum difference between the last quantile and the minimum and maximum instead. This leads to a reduction of transmitted values during validation, but it can increase the size of the refinement interval. IQ was implemented as suggested in Section 4.2 with the same hints as HBC.

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

[3]c.p. http://standards.ieee.org/about/get/802/802.15.html

POS was configured to transmit values directly during refinement, if all values in the refinement interval fit into a single message. Since s_p^{max} was set to 128 bytes by default, 64 two-byte measurements could be transmitted. In validation messages, two hints containing the minimum and a maximum of all values that changed their state were sent.

LCLL was implemented with recursive Hierarchical Refining (LCLL-H) and Slip Refining (LCLL-S). Hierarchical Refining leads to a worst-case behaviour of $O(\log_b(r))$ refinements while Slip Refining yields only linear worst-case performance. However, since Slip Refining performs very well if the distance between two consecutive quantiles is low, we implemented this approach as well.

Furthermore we improved LCLL's validation stage. In the original implementation each node sorts its value into the corresponding bucket and forwards it to its parent node. Since this behaviour leads to high overhead for sending nodes, we chose to implement a different technique: If a node's value stays in the same bucket as during the last round, no update is sent. However if a node's value slipped to another bucket between two rounds, two buckets are transmitted. The last bucket of the node is reduced by 1 to show that the node left this bucket. The count of the new bucket is increased by one to indicate the new bucket. This technique can significantly decrease the number of message transmissions since nearly all nodes with measurements in the boundary buckets do not have to transmit anything during validation. LCLL was also improved by applying improvements suggested in [21], and a node did only transmit its value during a refinement if it was contained in the refinement interval, but not if it was contained in a boundary bucket. b was set according to the message size as suggested by [16].

For TAG we assumed that the root node knows the number of nodes in the network. Therefore k can be broadcasted during query dissemination and only k values have to be forwarded by intermediate nodes to retrieve the k-th value at the root node. Note that we cut off the graphs of TAG if its maximum per-node energy consumption was too high.

5.1.7 Variable Setting

Given a set of input variables, we performed 20 simulation runs with 250 rounds each. The independent variables were set according to Table 2. We varied the number of nodes ($|N|$), the noise of the sinusoidal function (ψ), the period (τ) of the sinusoidal function, and the radio range (ρ) of a node. The requested quantile was fixed to the median, i.e., ($k = \lceil |N|/2 \rceil$).

Table 2: Independent variables. Bold text: default values.

Variable	Values
$\|N\|$	125, 250, 500, **1000**, 2000
ψ	0, 5, **10**, 20, 50 percent
τ	250, **125**, 63, 32, 8 rounds
ρ	15, **35**, 60, 85 m

5.2 Evaluation Results

5.2.1 Varying the Number of Nodes ($|N|$)

With increasing node count $|N|$ (Figure 6), the maximum per-node energy consumption grows for all approaches. The vast majority of their increase in energy consumption comes

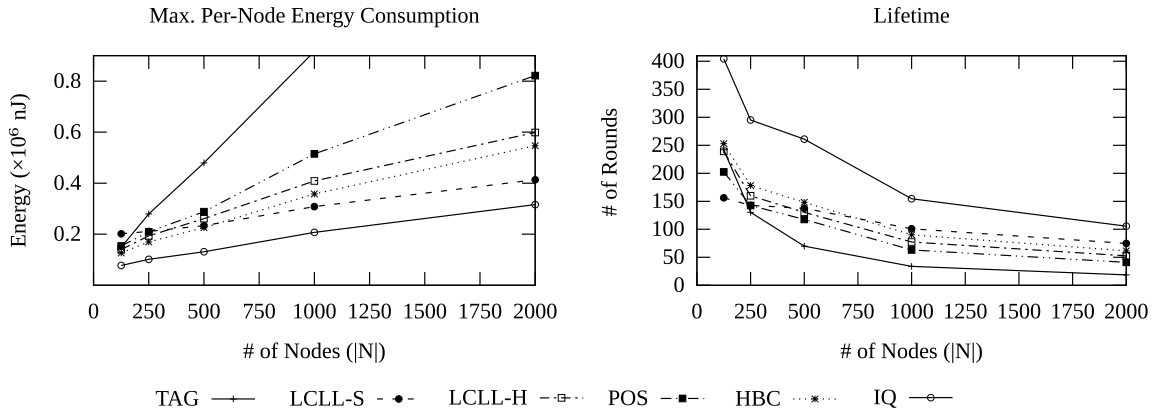

Figure 6: Synthetic Dataset, Varying $|N|$

from the growing number of values an intermediate node has to receive. With higher $|N|$ the network becomes more dense because the area containing the nodes does not change. This leads to intermediate nodes having more children, and more children imply more message receptions.

Note that the maximum per-node energy consumption of LCLL-S scales very good with $|N|$. This is mainly the case because the refinement interval of this approach is very selective. Therefore an intermediate node only has to receive a few values from children even with large $|N|$, although the number of refinements of LCLL-S is much higher than for, e.g. LCLL-H. Unfortunately for small $|N|$ this technique leads to a worse performance because the many refinements can not be justified by only a small amount of nodes that answer to refinement responses.

For IQ the sending energy increases similar to POS. Since we assume the same distribution and trends during all simulations, the length and position of Ξ is similar for all $|N|$, however Ξ contains more values for large $|N|$, forcing intermediate nodes to transmit more values. LCLL-H shows a similar behaviour concerning the sending energy because the efficiency of histogram compression drops with $|N|$. Since LCLL-H histograms are very large, in our setting 64 buckets, a high value density can lead to a relatively high energy consumption. Note that the number of refinements does not change with $|N|$ for LCLL-H; it depends only on the distance between the old quantile and the new quantile. For LCLL-S the sending energy increases with $|N|$, because refinement responses have to be forwarded by intermediate nodes more often since the density of values increases.

5.2.2 Varying the Period (τ)

With varying the period (τ) we aimed at investigating the impact of a highly changing quantile on the introduced solutions. If a period lasts over many rounds, the difference between two consecutive quantiles is low, but with a large time interval between consecutive rounds the quantile underlies more extreme changes if the amplitude of the sinus function is not changed. It is clearly visible that all solutions perform best for high τ, i.e., a small change in the underlying physical phenomenon. In the following we will explain the reason for this behaviour.

For POS and HBC the hints are closer to the recent quantile if measurements change only slightly, and therefore the width of the refinement interval becomes small. This reduces the number of candidates and therefore the number of message transmissions and transmitted values, leading to a lower energy consumption. Furthermore, the new quantile often falls into HBC's *eq* interval if τ is large. Therefore, values often can be requested directly after validation instead of performing expensive refinements bevore sending the values. However if τ is small, the gain achieved by using this technique diminishes because the quantile moves too much as that two consecutive quantiles can be contained in the same range *eq* – and this causes additional refinements.

For LCLL-H the number of refinements and therefore the energy consumption increases if τ becomes small because the number of refinements of this approach depends logarithmically on the distance between two consecutive quantiles. This is also the reason why LCLL-S depends much more on τ since its number of refinements depends linearly on the distance between two quantiles.

Given IQ, the length of the interval Ξ is very small for large τ and therefore only a few values are transmitted during validation, leading to a low energy consumption. However if the cycle duration is very small, other solutions like HBC and LCLL-H can outperform IQ because, due to the large interval Ξ, IQ nodes have to transmit many values. This underlines the optimality of HBC and LCLL-H that is directly related to their logarithmic behaviour. However note that, in contrast to HBC or POS, the number of refinements barely changes with τ if IQ is used. This is the case because Ξ is chosen in a way that it is large enough to avoid as many refinements as possible.

5.2.3 Varying the Noise (ψ)

By manipulating the period τ of the sinusoidal function we explain the impact of large changes in the quantile on the analyzed algorithms. But it is also possible that the quantile stays nearly constant but the measurements of different nodes vary a lot, i.e., the effect of changing measurements on the quantile cancel out. We simulated this by adding noise (see Figure 8) to the value of each node. Note that noise only slightly affects the median, however if another quantile like $k = 1$ would be requested, noise could significantly change the resulting value.

HBC, POS and IQ nodes send updates if a node's measurement jumps from one side of the quantile to the other side between two consecutive rounds, hence the number of update messages increases with higher noise because more

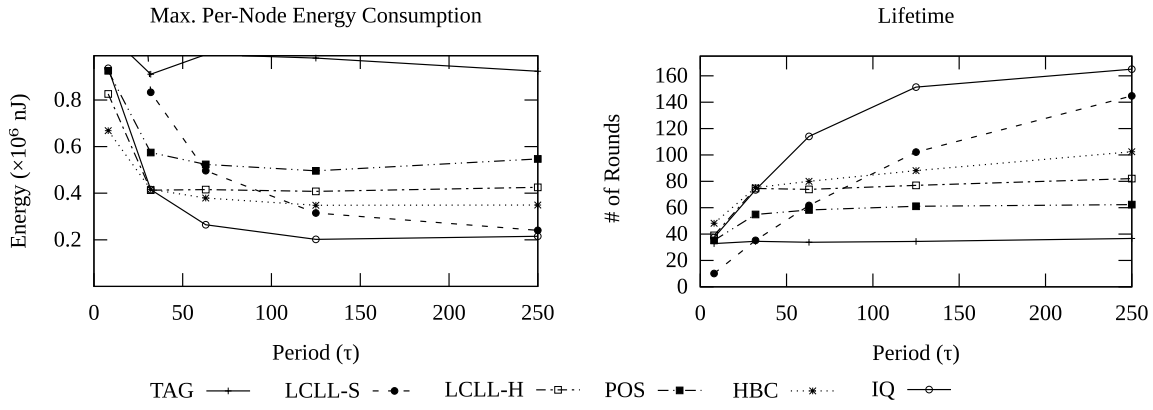

Figure 7: Synthetic Dataset, Varying τ

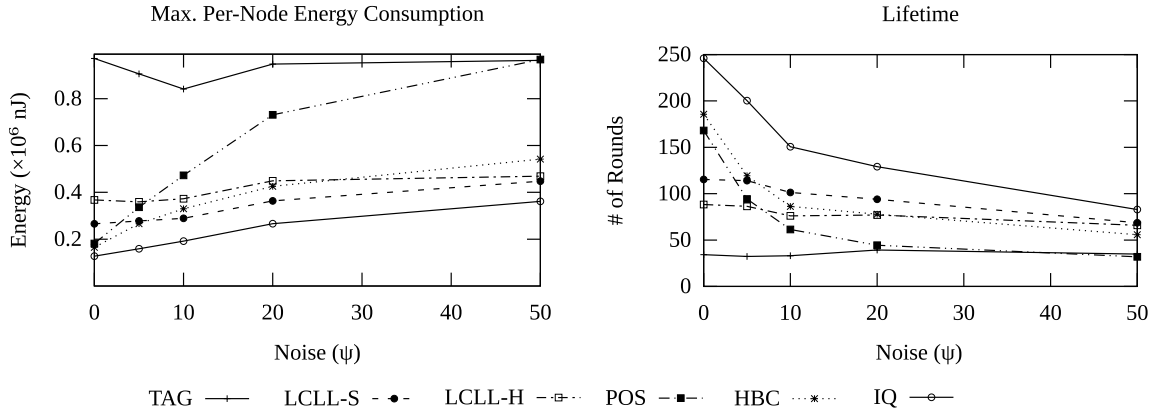

Figure 8: Synthetic Dataset, Varying ψ

nodes change their state, increasing the energy consumption of these approaches as well. Additionally, POS hints, which are also used by HBC and IQ in a slightly different manner, depend on the maximum distance of values (that changed their state) to the old quantile. With increasing noise this maximum distance increases as well and therefore the length of the refinement interval increases, leading to a larger set of candidates and increasing the number of refinements for POS. This effect should be similar to HBC, however the in-depth analysis of our solution showed another interesting result: Although with a noise of 0% quantiles never fell in the interval *eq*, the probability that the new quantile was contained in *eq* was quite high if the noise was set to 50%. The explanation is straightforward. If there isn't any noise then only the sinusoidal function affects the the quantile. Therefore, if the inclination of this function is too high, the new quantile will never be contained in *eq*. However, if there is noise, there is the possibility that the new quantile is very close to the recent quantile.

Also note that, for LCLL-H, noise does not have a large impact on the maximum per-node energy consumption. For this approach only the changing behaviour of the quantile is important. Since the changing pattern of the quantile is only slightly affected by noise, LCLL-H's performance is not affected as well. Therefore LCLL-H is a good choice if the quantile changes slowly but measurements oscillate a lot. Unfortunately this is not true for LCLL-S. Since this approach depends highly on the distance between two consecutive quantiles even minor changes of the quantile related to noise can increase the number of refinements. Therefore the curves for LCLL-H and LCLL-S are not parallel but the performance of LCLL-S drops to a performance similar to LCLL-H for high noise.

5.2.4 *Varying the Radio Range (ρ)*

The graphs visualizing the performance of the compared solutions with varying radio range ρ show similar results to the graphs from the snapshot query setting. Since the number of children of a node grows with ρ, a large fraction of the increase in energy consumption is related to the increasing number of values a node has to receive. This is also the reason why LCLL-S performes well for large ρ: due to its very restricted refinement range only a few nodes report refinement responses after each refinement request.

5.2.5 *Results with the Air Pressure Dataset*

We also performed simulation runs with the more realistic air pressure dataset. Since the correlation of consecutive values plays an important role for continuous approaches, we performed several simulations where we skipped an increasing amount of samples from the input data. This proceeding is equivalent to sleeping a longer time interval between two consecutive rounds. Concerning the range of values, we tested the algorithms within two different settings. The optimistic setting takes the minimum (v_{min}) and maximum

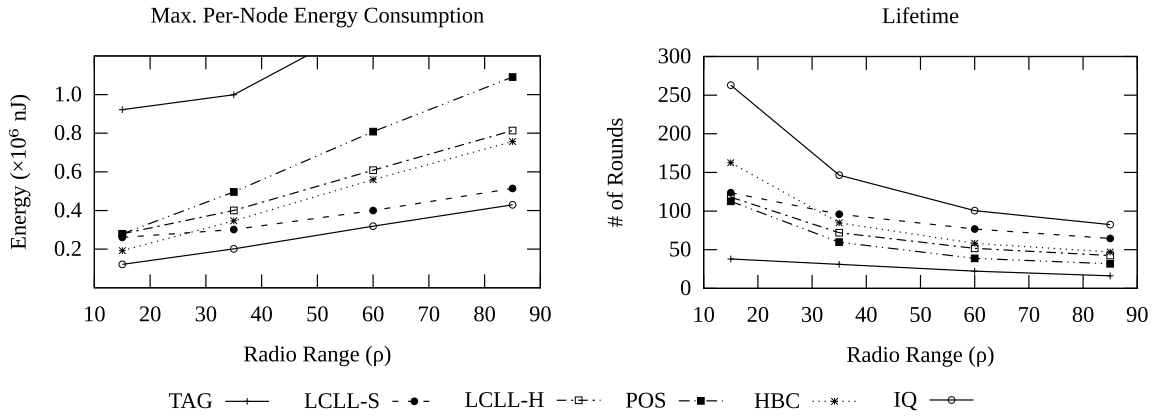

Figure 9: Synthetic Dataset, Varying ρ

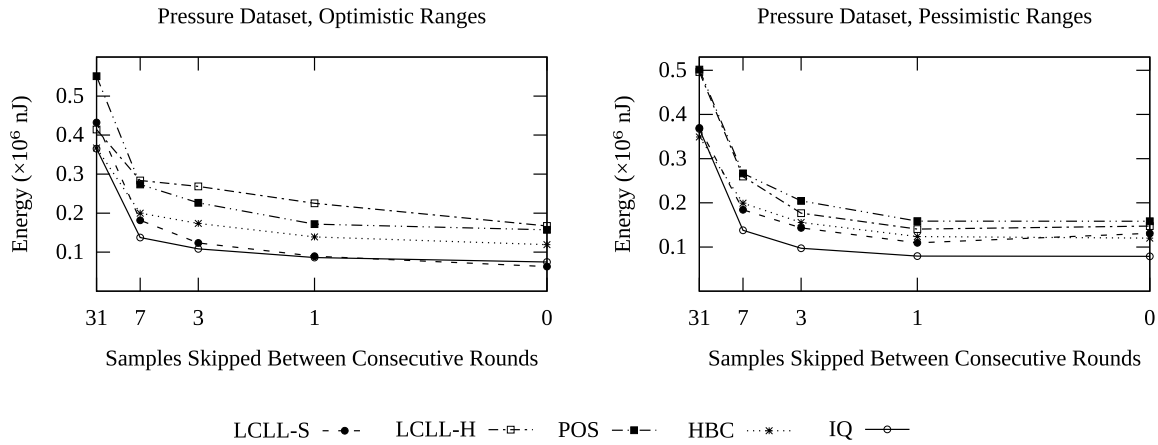

Figure 10: Air Pressure Dataset Dataset, Varying the Sampling Rate

(v_{max}) value from the whole dataset and scales all values such that $r_{min} = v_{min}$ and $r_{max} = v_{max}$. The pessimistic setting assumes higher fluctuations of the measured data and sets therefore the maximum and minimum air pressure values ever measured on earth as the range, i.e., $r_{min} = 856$ and $r_{max} = 1086$. The results of our analysis can be found in Figure 10. Note that only the graphs visualizing the maximum per-node energy consumption are shown for the sake of brevity.

The results are very similar to the results with varying period that we analyzed previously. The more skipped samples, the more the quantile changes, introducing additional refinements. Note that POS-based approaches are barely affected by the scaling of values since their performance only depends on the number of values in the refinement interval that is not influenced by scaling.

However, LCLL-H shows better performance if values are very close together (pessimistic setting). This is the case because if values are close together, LCLL-H has to perform less refinements as if values are further away from each other, and refinements are very costly for LCLL-H for two reasons: First, refinement intervals can become very large and therefore unspecific. Second, each refinement implies at least two steps, zooming in and zooming out.

In contrast, LCLL-S refinements are very cheap if the old and the new quantile are close together, and therefore LCLL-S is able to outperform IQ in the optimistic setting with a high sampling rate, although the number of refinements performed by LCLL-S is larger than in the pessimistic setting. This is the case because in the pessimistic setting more values fall into the focused window such that more validation messages are transmitted, increasing the number of received values for intermediate nodes and decreasing the performance of LCLL-S.

6. CONCLUSIONS

In this paper we addressed the problem of computing quantiles in hierarchical WSNs by employing either a b-ary search and a heuristic solution. When compared to two previously proposed solutions for processing quantile queries, using real and synthetic datasets, the b-ary search outperformed both, i.e., extended the WSN's lifetime, in virtually all cases. However, our heuristic solution was even able to outperform the optimized b-ary search in many cases, as the heuristic solution was optimized to reduce the number of refinements as much as possible. In conclusion, a heuristic algorithm should be employed when there is some temporal correlation between consecutive values. In contrast, the optimized b-ary search is more useful if the temporal correlation between consecutive quantiles is low.

During future research we would like to address the prob-

lem of message loss. If messages get lost, a rank error is introduced and it would be interesting to analyze the behaviour of different approaches under loss in order to restrict the rank error as much as possible, and to provide a real-time error estimate at the base station.

Acknowledgements

Research partially supported by DAAD (Germany) and NSERC (Canada).

7. REFERENCES

[1] H. Akcan and H. Brönnimann. A new deterministic data aggregation method for wireless sensor networks. *Signal Processing*, 87(12):2965–2977, 2007.

[2] K. Ammar and M. A. Nascimento. Histogram and other aggregate queries in wireless sensor networks. In *Proc. SSDBM*, pages 527–536, 2011.

[3] K. Ammar, M. A. Nascimento, and J. Niedermayer. An adaptive refinement-based algorithm for median queries in wireless sensor networks. In *Proc. MobiDE*, pages 9–16, 2011.

[4] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Sampling algorithms: Lower bounds and applications. In *Proc. STOC*, pages 266–275, 2001.

[5] B. A. Bash, J. W. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. In *Proc. DMSN*, pages 32–39, 2004.

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

[7] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the lambert w function. *Advances in Computational Mathematics*, 5:329–359, 1996.

[8] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Space- and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proc. of PODS*, pages 263–272, 2006.

[9] 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.

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

[11] W. B. Heinzelman. *An Application-specific Protocol Architecture for Wireless Networks*. PhD thesis, Massachusetts Institute of Technology, 2000.

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

[13] T. Kohonen. *Self-Organizing Maps*. Springer, Heidelberg, 2001.

[14] F. Kuhn, T. Locher, and R. Wattenhofer. Tight bounds for distributed selection. In *Proc. of SPAA*, pages 145–153, 2007.

[15] 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.

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

[18] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. of WSNA*, pages 88–97, 2002.

[19] H. E. H. Mustafa, X. Zhu, Q. Li, and G. Chen. Efficient median estimation for large-scale sensor rfid systems. *IJSNet*, 12(3):171–183, 2012.

[20] J. Niedermayer. *Quantile Queries in Wireless Sensor Networks*. Diploma thesis, LMU Munich, 2011.

[21] J. Niedermayer, M. A. Nascimento, M. Renz, P. Kröger, K. Ammar, and H.-P. Kriegel. Cost-based quantile query processing in wireless sensor networks. In *Proc. MDM*, 2013.

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

[23] 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.

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

[25] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, 2004.

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

[27] 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*, 2009.

[28] D. Wang, Y. Long, and F. Ergun. A layered architecture for delay sensitive sensor networks. In *Proc. of SECON*, pages 24 – 34, 2005.

[29] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[30] Y. Zhang, X. Lin, Y. Yuan, M. Kitsuregawa, X. Zhou, and J. X. Yu. Duplicate-insensitive order statistics computation over data streams. *IEEE TKDE*, 22(4):493–507, 2010.