A Wavelet Transform for Efficient Consolidation of Sensor Relations with Quality Guarantees

Mirco Stern
Universität Karlsruhe (TH)
Germany
Mirco.Stern@ipd.uni-karlsruhe.de

Erik Buchmann
Universität Karlsruhe (TH)
Germany
Buchmann@ipd.uni-karlsruhe.de

Klemens Böhm
Universität Karlsruhe (TH)
Germany
Boehm@ipd.uni-karlsruhe.de

ABSTRACT

Answering queries with a low selectivity in wireless sensor networks is a challenging problem. A simple tree-based data collection is communication-intensive and costly in terms of energy. Prior work has addressed the problem by approximating query results based on models of sensor readings. This cuts communication effort if the accuracy requirements are loose, e.g., if the temperature is required within $\pm 0.5°$C. For more accuracy, the models need frequent updates, and the communication costs quickly increase. In addition, sophisticated models incur substantial training costs. We propose a query-processing scheme that efficiently consolidates sensor data based on wavelet synopses. The difficulty is that the synopsis has to be constructed incrementally during data collection to ensure efficiency. Our core contribution is to show how to distribute the construction of wavelet synopses in sensor networks. In addition, our approach provides strict error guarantees. We evaluate our distributed wavelet compaction on real-world and on synthetic sensor data. Our solution reduces communication costs by more than a factor of five compared to state-of-the-art approaches. Further, our error guarantees for which efficient data consolidation is possible are better than theirs by more than an order of magnitude.

1. INTRODUCTION

Wireless sensor networks (WSNs) are an important technology with many industrial applications. For instance, monitoring production processes is essential to avoid unscheduled downtimes and for quality management. As a concrete example, consider a drug manufacturer [32] who has to comply with legal requirements for documenting the production process. To do so, he has installed 130 wireless sensor nodes on an existing production line. WSNs are ideally suited for such settings. The fact that they come without wiring lowers installation costs by 80% and installation time by 90% and enables simple reconfigurations [41]. WSNs can consist of many nodes which are equipped with sensors, have constrained communication and computation capabilities and are battery operated. To obtain longevity, energy-efficiency is mandatory.

A common data-acquisition task in monitoring contexts is to make the sensor readings available at a central location periodically. Staff can then take corrective action immediately. In addition, it is often required to log the entire data, as with the drug manufacturer. An important characteristic of the data acquisition is that it usually covers readings from all of the sensors. This is quite intuitive: Sensor nodes are not cheap and, hence, they are placed with care at locations where monitoring is required. Thus, queries often simply consolidate the readings from the network, i.e., *they frequently have a low selectivity* [5]. Beyond monitoring, non-selective queries are common in explorative scenarios where users simply collect data to get an idea of the environment observed. Finally, any query may have subqueries which are not selective. This paper studies the evaluation of non-selective queries in WSNs. Our concern is efficient consolidation of the state of the network ("snapshot", i.e., readings of (multiple) sensors of each node at the same time). An application can request a snapshot once or periodically.

While, in a lot of applications, queries cannot dismiss any sensor readings, accuracy tends to be less critical: An approximation of the current snapshot is usually sufficient [11, 5]. At the same time, it is important to bound the error of approximations. E.g., the drug manufacturer has to document that the temperature in the production process did not exceed certain thresholds. This is not possible without error guarantees. We approximate snapshots of sensor readings under user-controlled error guarantees.

To obtain efficiency, prior work has focused on minimizing the sensing and communication costs. They dominate the power consumption by orders of magnitude, compared to computation and RAM accesses [42, 25]. While optimizing sensing costs is well understood [25], minimizing communication is challenging. A widespread method for answering queries is tree-based data collection (TDC). To reduce communication, TDC exploits the selectivity of queries by pushing data-reducing operators into the network. Then, only the data that qualifies for the result is sent along a routing tree to the base station. However, if selectivity is low, most of the data is in the result, and plain TDC is communication-intensive. A known idea for non-selective queries is approximate query processing based on models [5, 11]. Here, a model is a compact representation ("synopsis") of the measurements, and queries are processed against it. Current readings are used to update the model if error guarantees are not met. Model-based approaches suffer from the following problem: Predicting measurements is only possible with large error bounds, e.g., we query the temperature within $\pm 0.5°$C. Higher accuracy requires frequent updates. According to [11], "as epsilon gets small (less than .1 degrees), it is necessary to observe all nodes on every query...". In addition, sophisticated models require extensive training when each node has to transmit periodic readings to the base station, e.g., every 30 - 60 seconds [37].

To overcome these problems, we propose a query-processing scheme based on wavelet synopses. The wavelet transform is a

VLDB '09, August 24-28, 2009, Lyon, France

mathematical tool to decompose functions hierarchically. The idea is to transform the input data to a set of wavelet coefficients. Then, "thresholding" yields a synopsis, i.e., it discards some coefficients and retains those required to reconstruct the data within the error bounds. The use of wavelet synopses is motivated by their effectiveness in compressing data and providing accurate answers [9]. However, exploiting these capabilities in our context requires a distributed construction of synopses. Although being an area of active research, to our knowledge, no prior work has presented a distributed transform that is more efficient than a simple TDC (cf. Sections 2 and 7), let alone error guarantees in this context.

In this paper, we present SNAP ("SNapshot APproximation"), an efficient consolidation scheme for sensor readings. To achieve efficiency, SNAP incrementally constructs a wavelet synopsis during data collection. As our main contributions, we show how to distribute the transform as well as the thresholding. Regarding the distributed wavelet transform, we see two design alternatives. We could use some standard transform and map it onto the network, i.e., assign the computations to sensor nodes and set up a network overlay. The overlay has to reflect data dependencies between computations. This approach has been considered before (e.g., [43, 40]). The problem is that it sacrifices shortest paths: The routes in the overlay will exceed those of simple TDC by much (cf. Section 4). With SNAP, we explore an alternative design and integrate a wavelet transform into an optimal routing structure. We show that optimizing the integration is NP-hard and propose a polynomial algorithm that comes close to the optimal solution. Most notably, our solution is an online algorithm, i.e., the integration is not precomputed and can adapt to changes in the routing structure.

Regarding the thresholding process, discarding a subset of the detail coefficients yields a compact synopsis. However, doing so in a distributed environment leads to suboptimal synopses and to a huge communication overhead (see Section 5). Hence, we propose a distributed solution that is different and is inspired by work on image compression: Instead of discarding coefficients, we find a compact representation for them. The fundamental problem in distributing this alternative mechanism is as follows: To assign compact codes to the coefficients, we must know their frequencies in the overall synopsis. However, the synopsis is created incrementally. The frequencies cannot be known at the time of encoding. We show how to estimate the frequencies in a way that is mathematically sound. Our approach is also applicable to tuples with multiple attributes. This is an optimization problem in its own right if coefficients are discarded [9]. In summary, our contributions are:

- We present SNAP, a distributed approach to compute wavelet synopses incrementally. To our knowledge, SNAP is the first distributed wavelet compaction providing error guarantees.
- To distribute the wavelet transform, we propose to integrate it into the routing structure. SNAP is first to explore this idea.
- Finding an optimal integration is NP-hard. We provide a heuristical solution and show that it performs well by means of an optimal algorithm based on dynamic programming.
- To distribute the construction of synopses, SNAP compactly encodes coefficients, instead of discarding them.
- Compacting the coefficients requires to know their frequencies in the overall synopsis. We show how to accurately estimate these frequencies.
- We evaluate the performance of SNAP based on real-world and synthetic sensor data. Our results are that SNAP reduces the communication costs by more than a factor of five compared to state-of-the-art approaches. SNAP improves the limit in the error guarantees for which data can be efficiently consolidated by more than an order of magnitude.

2. RELATED WORK

Data-reduction mechanisms studied by the database community include sampling, histograms, and wavelets. For sensor networks, there are reduction schemes based on modeling sensor measurements. As [3, 27, 39] has demonstrated that wavelets can achieve a higher degree of accuracy of query results compared to histograms and random sampling, we focus on model-based and wavelet-based synopses in what follows. We leave aside approaches that exclusively compute aggregates, e.g., [7, 10].

Model-based Approaches. The idea behind model-based approaches is to answer queries based on a model of the current measurements, instead of querying the data from the sources. The approaches differ in the complexity of the model and their error guarantees. However, all model-based approaches estimate the measurements. If they provide error guarantees, their performance critically depends on the deviation from the actual sensor readings that an application can tolerate. For instance, estimating the temperature within $\pm 0.5^{\circ}$C works well. For more accurate estimations in turn, frequent updates of the model are required, and the communication costs quickly increase. In Section 7, we compare SNAP to model-based approaches and show that it can efficiently answer queries for error guarantees that are tighter by orders of magnitude. Subsequently, we briefly introduce these approaches. This also justifies our choice of reference points for the experiments.

The simplest model is a constant. The base station caches the most recently received readings of each node. A node sends an update whenever a reading deviates by more than a threshold t. [31] addresses the problem of choosing t such that the number of transmissions is minimal. [20] proposes to use Kalman Filters as a model. SAF [37] uses time series forecasting models with much weaker error guarantees. SNAP in turn yields strict guarantees.

The approaches mentioned exploit temporal correlations. Capturing spatial correlations requires modeling several nodes, as [5, 11] do. Both use time-varying multivariate Gaussians. BBQ [11] primarily executes at the base station, which maintains a probabilistic model for the network. If the confidence of an estimate is insufficient for a query, BBQ identifies the optimal set of attributes to observe and queries them. Ken [5] organizes nodes in clusters and maintains a model for each of them. Each cluster simulates the estimates of the base station and sends updates whenever error requirements are violated. This approach gives way to strict error guarantees. However, next to the fact that error bounds are not tight, such sophisticated models entail a very expensive learning phase.

PAQ [38] is a cluster-based approach that covers spatial correlation based on a much simpler model, at the cost of increased communication. Snapshot queries [23] follow a similar scheme. [15] shows how to regress a function to the sensor measurements in a distributed fashion. Both approaches provide no error guarantees.

Centralized Wavelet-Based Approaches. From a database perspective, there are many successful applications of wavelets as a data reduction tool. The corresponding algorithms are inherently centralized and are inappropriate for distributed environments (cf. Section 5). Due their vast number, we only present some of them as examples. Afterwards, we discuss distributed approaches in detail.

Wavelets have been used in selectivity estimation [27], for answering range-sum aggregate queries over data cubes in [39] and for general-purpose queries in [3]. [28] studies dynamic maintenance of synopses. [9] introduced extended wavelets for data sets with multiple measures. Recently, [33] also addressed the overhead due to wavelet coordinates. While most of the early work focused on the SSE (sum squared error), [12, 13] considered maximum error metrics. For streaming data, [14] proposes algorithms for building approximate synopses, and [22] introduces a greedy algorithm

for constructing synopses with maximum error guarantees.

Distributed Wavelet-based Approaches. A problem that has received a lot of attention when deploying wavelet transforms in sensor networks is mapping their regular refinement scheme onto the irregular network topology. To compute approximate query answers, a further problem has to be solved: One needs a distributed thresholding approach with error guarantees. To our knowledge, our work is the first solution to this problem.

[1] distributes the computation of a Haar wavelet if the sensor network is a 1-dimensional regular chain. [6] is similar, except that different wavelets are used (5/3-wavelets). Both approaches are confined to very specific topologies. [43] distributes the computation by means of an overlay, which is a 1-dimensional chain. [40] proposes precomputation of a 2-dimensional hierarchical refinement scheme and computation of a distributed wavelet transform on it. These approaches differ from our work in their goal as they are not interested in consolidating an approximation at the base station. [40] also shows that transforming the data and subsequently collecting the coefficients is less efficient than collecting the original data. This is because the routes in the overlay are significantly longer than a shortest path routing tree. [16, 17] integrates the wavelet transform into the routing tree. Since the routing tree is not balanced, the authors introduce zeros for missing values in the transform ("zero padding"). As shown in Section 7, this yields large detail coefficients, which is in the way of a compact representation. [8] copes with the irregular topology by assigning the nodes to clusters and transforming the data of each cluster. The problem is that only data within a cluster is decorrelated, redundancies between clusters are not removed. Thus, performance is suboptimal, as we will show in Section 7.

3. PRELIMINARIES

The notion of a "sensor network" is by no means well-defined. Frequently, the application influences the hardware deployed and the network architecture. This section provides the context of our work. The architecture and some fundamental aspects of query processing are described in Sections 3.1 and 3.2. In 3.3, we specify the problem, and Section 3.4 provides a brief recap of wavelets.

3.1 Network Architecture

Our work is based on a network architecture consisting of hundreds of stationary sensor nodes. Each node is equipped with several sensors, a processor, a small RAM, a wireless radio, and is battery operated. A base station serves as the access point. Each node is aware of its neighbors, i.e., the nodes in its wireless range. It communicates with the other nodes using multi-hop routing.

To provide an example of sensor nodes, the nodes in our lab are SunSPOTs. They have a processor with a 32 bit ARM920T core which executes at 180MHz maximum clock speed. The nodes are equipped with 512KB RAM and with a 4MB Flash memory. Writing to Flash typically incurs non-negligible energy costs. We disregard these costs as SNAP exclusively uses RAM. As radio transceivers, the nodes use a CC2420 which is IEEE 802.15.4 compliant ("ZigBee"). SunSPOTs are a development platform and are shipped with a default sensor board. It contains a light sensor, a temperature sensor, and an accelerometer. (In real deployments, the sensors are usually customized to the application.) The nodes are powered with a 3.7V rechargeable 750 mAh lithium-ion battery.

3.2 Query Processing

Declarative queries are an attractive interface to collect data in sensor networks, as they hide technical details [11, 42]. To facilitate declarative queries, the network is seen as a *(sensor) relation*. One can perceive it as a relation with one attribute per sensor (e.g., temperature) and one tuple per node. As the attributes reflect the sensors of a node, the relation typically has less than ten attributes.

Queries can be one-time or continuous: A *one-time query* asks for the current snapshot. A typical usage is an interactive exploration. A *continuous query* reports snapshots periodically.

Basic tree-based query processing works as follows: A query is input at the base station. The network then disseminates the query by a simple broadcast flooding. Query results are propagated to the base station along a routing tree, with the base station as the root. A routing tree is maintained in a distributed fashion: Based on a periodic beaconing mechanism, each node maintains a parent that minimizes the distance to the base station. In a state-of-the-art implementation, the distance is measured in 'number of expected transmissions'. Intuitively, this is the hop count weighted with the link quality. This guarantees an important property of routing trees: They are shortest path trees, with respect to the distance metric. Further, routing trees are highly irregular, i.e., the degree of the nodes can vary arbitrarily (in practice, a node has about 1 to 15 children), and the tree is by no means balanced.

3.3 Problem Statement

Our goal is to efficiently approximate snapshots under user-controlled error guarantees. As the guarantees in our targeted applications refer to individual values, we need a maximum error metric. This is because for SSE (sum squared error), the error for individual values could be arbitrarily high [12]. In the following, we will sometimes refer to the maximum error e as specified in the query by "(maximum) error bound". Formally, our goal is answering queries that conform to the following structure:

```
SELECT Att_1 ± e_1, ..., Att_n ± e_n
FROM Sensors
WHERE predicates(Att_1, ..., Att_n)
{SAMPLE PERIOD x | ONCE}
```

For instance, `SELECT temp` $\pm .1°$`C FROM Sensors ONCE` queries the current temperature readings with a maximum error of $.1°$C. The `WHERE`-clause is optional. The semantics is the standard SQL semantics with extensions for temporal aspects of sensor data: We adopt the non-SQL clauses `ONCE` or `SAMPLE PERIOD` from TinyDB [24] as SNAP supports one-time as well as continuous queries: `ONCE` computes the result based on the current snapshot. Thus, `SELECT temp` $\pm .1°$`C FROM Sensors ONCE` returns one tuple from each node that belongs to `Sensors`. `SAMPLE PERIOD` yields a continuous monitoring. It defines the time interval between *independent* executions of the query. The query is executed every x seconds on the most recent snapshot.

3.4 Constructing Wavelet Synopses

Computing a wavelet synopsis consists of two subsequent tasks, wavelet transform and thresholding.

Wavelet Transform. We illustrate the principle by means of an example. Further information can be found in the standard literature, e.g., [36]. Our description is based on the Haar transform which is the basis of SNAP. We justify this choice in Section 4.

Consider the following dataset, e.g., one column of a database table: $D = [2, 6, 7, 4]$. The transform pairs neighboring values v_x and v_y and computes an *approximation coefficient* for each pair. In case of the Haar transform, the approximation coefficients are simply the averages ($\frac{v_x+v_y}{2}$): $[4, \frac{11}{2}]$. This is a "lower-resolution" representation of D. In addition, a set of *detail coefficients* is computed as pair-wise differences divided by 2 ($\frac{v_x-v_y}{2}$): $[-2, \frac{3}{2}]$. These coefficients contain the information lost by averaging: Given

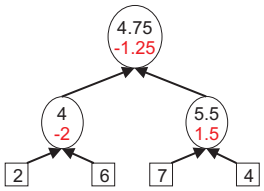

Figure 1: Structure graph for the Haar example

both the approximation and the detail coefficients, the original data can be reconstructed. Recursive application of this pairwise averaging and differencing process on the lower-resolution data (the approximation) yields the following full transform:

Resolution Level	Approximations	Detail Coefficients
0	$[\frac{19}{4}]$	$[-\frac{3}{4}]$
1	$[4, \frac{11}{2}]$	$[-2, \frac{3}{2}]$
2	$[2, 6, 7, 4]$	

The result of a Haar transform is the single overall approximation coefficient followed by the detail coefficients in the order of increasing resolution. Thus, the transform of our example data is given by $W_D = [\frac{19}{4}, -\frac{3}{4}, -2, \frac{3}{2}]$.

Each coefficient of the data transformed can be associated with a *coordinate* which identifies the coefficient by its resolution level, along with its position within this level. Intuitively, the coordinate simply corresponds to the position of the coefficient in W_D.

Thresholding. No information has been lost during the transform. Notably, the original dataset D has four values, and so does W_D. The task of creating a compact representation of the dataset (synopsis) from W_D is called *thresholding*. In a nutshell, thresholding is discarding a subset of the detail coefficients. During reconstruction, the coefficients omitted are assumed to be zero. Thus, the resulting wavelet synopsis is an approximation of D.

Why do we transform the data prior to thresholding? Wavelet transforms decorrelate the data. Most notably, if D contains similar values, the detail coefficients tend to be small. Thresholding then introduces only small errors. Thus, *in our context, the goal of the transformation step is to obtain small detail coefficients.* The subsequent thresholding problem then is an optimization problem, e.g., find a minimum number of detail coefficients to retain, given a maximum error in the reconstructed data.

Structure Graph. To distribute the wavelet transform, its logical data flow is important [17]. We capture it by means of a "structure graph". Figure 1 shows the structure graph for our example. A node represents a computation, i.e., nodes compute a function on the data obtained from their children. Edges represent data dependencies. For the Haar transform, the structure graph is a balanced binary tree. In general, structure graphs are not necessarily trees but directed acyclic graphs (e.g., Daubechies-4, etc.). Graph structures arise for transforms that use the same approximation coefficient in more than one higher-level computation. Otherwise, we speak of a "structure tree". The term "computation node" stands for nodes of the structure graph, in contrast to sensor nodes.

4. SNAPSHOT APPROXIMATION

This section presents our approach for distributing wavelet transforms. We extract design alternatives and justify our choice of integrating a transform into an unmodified routing tree (Section 4.1). Our integration approach builds on flexible structure trees (Section 4.2). We say how to optimize the integration in Section 4.3.

4.1 Distributing Wavelet Transforms

To reduce communication compared to TDC, we have to construct the synopsis during forwarding incrementally. This requires a *distributed* wavelet transform. The problem is integrating a structure graph with the network topology: Here, 'finding an integration' means finding a wavelet transform and a mapping of its computation nodes to sensor nodes. Then the edges of the structure graph imply a communication overlay.

At a high level, we see two alternatives to address the problem: (1) One could adapt the routing structure to the transform, i.e., take a *fixed* structure graph and impose it as an overlay onto the network. However, network topologies are highly irregular. Mapping a fixed structure onto the network thus has an undesirable effect on the communication costs: It leads to nodes that merely forward data without prior wavelet compaction. The routes might not even be on the shortest paths to the base station [17]. (2) One could integrate the transform into the routing structure, i.e, the routing structure is fix. Each node would receive approximation coefficients from its children in the routing structure, transform them to obtain a synopsis and forward it to its parent. This approach requires specifying a structure tree that can be mapped onto the routing tree.

Related work has mostly explored the first approach (cf. Section 2). However, there are strong arguments for the alternative design. Firstly, communication is difficult in WSNs, i.e., links are often fragile, and packet-loss rates are high. A lot of work has gone into maintaining routing trees that address these problems. It is desirable to build upon this mature technology. Second, recall that the routing tree is a shortest path tree. Deviating from it means doing suboptimal routes. SNAP therefore explores this second alternative. Note that this lets SNAP cope with node failures and the network-related problems mentioned above.

There is one approach for building wavelet histograms [16] that operates on a routing tree, i.e., each node transforms the data received. To cope with the irregularity of the routing tree, the *input data* of the transform is adjusted ("zero padding"): Whenever a node has less than 2^n children, the input is filled up with zeros. The problem is that this results in *large* detail coefficients whenever an approximation coefficient is combined with a zero. For our problem, zero padding turns out to be worse than a simple TDC (cf. Section 7). In contrast, our goal is to adjust the transform itself.

4.2 Integration Approach

This section says how a structure graph can be integrated into a routing tree. The key idea for coping with the irregularity of the routing tree is to abandon the rigid structure of classical transforms.

As a first step in developing an integration approach, we need to select a wavelet transform which will be integrated (e.g., Haar, Daubechies-4, Mexican Hat, etc.). To do so, the question is which properties its structure graph must have to enable an integration. Foremost, the structure graph must be a tree. Otherwise, it would be impossible to map it onto a routing tree. It would be optimal if the structure graph complied with the routing tree. Unfortunately, no classical wavelet transform fulfills this requirement. This is because classical transforms have a regular refinement scheme. Their structure trees are balanced, and the node degree is fixed.

However, there are mathematical foundations on Haar wavelet transforms that have arbitrary (binary) structure trees [30]. These transforms are more general than classical Haar transforms since the structure tree is not balanced any more. The problem remaining when integrating such a structure tree into a routing tree is that routing trees are not binary but have varying node degrees. The key observation to address this issue is that if a Node n_i has to transform m_i approximation coefficients, it is possible to arrange

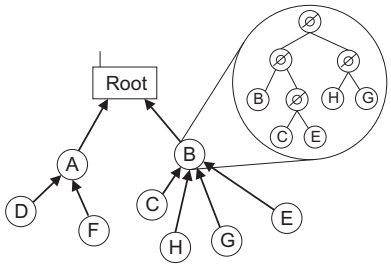

Figure 2: Integration approach

them in a binary tree and to apply the transform.

The preceding observation is the basis of our work, i.e., SNAP uses this flexible Haar transform. The flexibility of an *arbitrary* binary structure tree suffices to solve our integration problem. This is illustrated in Figure 2: Node B has set up a binary structure tree on the coefficients received from its children. This approach is pleasantly simple. At the same time, the flexibility in the structure tree leads to an optimization problem: find an optimal integration. This will be in the focus of the subsequent discussion.

4.3 Finding the Optimal Structure Tree

In the following, we present an online algorithm to integrate a binary structure tree into the routing tree. By performing the integration on-the-fly in contrast to precomputing it, the structure tree can react to changes in the network topology. Further, we avoid the communication overhead of an isolated precomputation.

We start by stating the optimization problem and establish its NP-hardness. As we cannot expect to find any exact polynomial algorithm, we devise a polynomial heuristic. We present an algorithm that finds an optimal integration based on dynamic programming (DP), to evaluate the heuristic.

The optimization problem. If a Node n_i receives m_i approximation coefficients, one per child in the routing tree, it adds its own sensor reading as a further coefficient. Then n_i transforms the data to obtain one overall approximation and m_i detail coefficients. This process reveals that the routing tree already determines the rough shape of the structure tree. Only for the $m_i + 1$ approximation coefficients at Node n_i, the structure tree is still unclear. The node needs to arrange the approximation coefficients received in a binary tree. For instance, if Node n_0 receives two approximation coefficients a_1 and a_2, there are three possible binary structure trees. n_0 can either combine a_1 and a_2 and subsequently combine the result $a_{1,2}$ with its own coefficient a_0. Alternatively, it can start combining a_0 and a_1 or a_0 and a_2. These combinations yield different detail coefficients. Recall from Section 3.4 that the sizes of the detail coefficients determine the size of the synopsis, which is created from the data transformed. Thus, by specifying the structure tree, we determine the size of the resulting synopsis. The optimization problem then is to find the best alternative for n_0.

What is an appropriate optimization function? We want to minimize the data that n_i forwards. This implies the optimization function Ω: Ω simply is the size of the synopsis (*number of bits*) built from the transformed data. We defer constructing the synopsis to Section 5. Note that Ω also works for data with multiple attributes.

Theorem 1: (Complexity of the problem) The problem of finding the optimal structure tree is NP-hard.

Proof: *(Sketch)* The problem of ordering joins can be reduced to our problem. Ordering joins in its most general form has recently been shown to be NP-hard [34]. The general problem includes considering bushy trees, i.e., there are no restrictions with respect to the ordering. It also involves cross products, i.e., every pair of relations from the query can be combined. The reduction works as follows: Every relation of the join query is mapped to an approximation coefficient. The cost function corresponds to the costs of executing the join of a particular ordering. Then, an optimal structure tree corresponds to an optimal join ordering.□

Heuristic approach. Due to the hardness of the problem, any algorithm that computes a structure tree on resource-constrained sensor nodes should not try to enforce optimality. We use a heuristical algorithm instead, which greedily minimizes the optimization function: In every step, the heuristic combines those approximation coefficients yielding the smallest detail coefficient (with respect to Ω). In other words, it is simply agglomerative clustering creating a dendrogram, where our optimization function is the distance. We omit the pseudocode as it is straightforward (see for instance [21]). We now say why this heuristic is appropriate. To do so, we present a DP-based algorithm that computes an optimal solution, and in Section 7 we will compare the heuristic to it.

Finding an optimal structure tree. The idea of DP is that whenever two partial solutions are equivalent (can be substituted in the overall solution), one only needs to consider the better one with respect to the optimization function. In our case, two partial structure trees are equivalent if they cover the same set of approximation coefficients and yield the same overall approximation coefficient. The second condition is required since the overall approximation is computed based on *binary* averages. Thus, the overall coefficient varies slightly with the order of the average computations, i.e., with the structure tree. Figure 3 shows the corresponding DP algorithm.

The algorithm incrementally constructs all partial solutions of size i. In Line 6, the partial solutions of size 1 are initialized. A partial solution consists of five fields: (1) set of input coefficients covered, (2) overall approximation coefficient, (3) the detail coefficients of the partial solution, (4) the corresponding structure tree, represented as a nested set, and (5) the costs of the partial solution (size). The actual algorithm starts in Line 8. It iterates over the size of the solutions and builds partial solutions of size i by combining pairs of partial solutions (Lines 10 - 15). Only combinations of partial solutions that do not overlap in the covered input coefficients are valid (Lines 28 - 30). In Lines 32 to 37, the new partial solution is constructed. Lines 17 to 23 implement the DP truncation.

5. CONSTRUCTING SYNOPSES

The thresholding problem is fundamentally different in a distributed setting. We demonstrate the shortcomings of a distributed discarding of coefficients in Section 5.1. To avoid these problems, we use a different mechanism as starting point: Instead of discarding coefficients, we compactly encode them (Section 5.2). To achieve a distribution of this alternative mechanism, SNAP estimates the frequencies of the detail coefficients (Section 5.3).

5.1 Thresholding in a Distributed Setting

In the following, we discuss two problems that arise if we distribute centralized solutions. (1) Discarding coefficients leads to suboptimal synopses. (2) There is a substantial communication overhead, because discarding coefficients and providing error guarantees requires the algorithm to maintain some state. This state would have to be forwarded along the tree in addition to the data.

In centralized settings, thresholding is an optimization problem under constraints. The goal either is to minimize an error given a maximum size of the synopsis or, to minimize the size given a maximum error bound. In a distributed context, the optimization problem is different. This is because we need to forward partial synopses, which incurs communication costs. Thus, we also need

```
1  OptimalStructureTree(Set {a_1, ..., a_m})
2      //a_i: approximation coefficients (from children + own measurement)
3
4      //initialization − partial solutions of size 1:
5      for j = 1 .. m
6        PartSolutions[1] = PartSolutions[1] ∪ ({a_j}, a_j, ∅, {a_j}, 0);
7
8      for i = 2 .. m //i == size of solution
9        for l = 1 .. (i − 1) //l == length of left subtree
10         for k = 1 .. |PartSolutions[l]|
11         PartSol lSubtree = PartSolutions[l].nextElement();
12           for o = 1 .. |PartSolutions[i - l]|
13           PartSol rSubtree = PartSolutions[i − l]. nextElement();
14
15           PartSol newSol = CombineStructureTrees(lSubtree, rSubtree);
16
17           if (∃ x ∈ PartSolutions[i]:
18              x.covAPPs == newSol.covAPPs && x.app == newSol.app)
19             //truncation: retain only the better one
20             if (x.costs > newSol.costs)
21               PartSolutions[i] = (PartSolutions[i] − {x}) ∪ {newSol}
22           else //no equivalet solution available
23             PartSolutions[i] = PartSolutions[i] ∪ {newSol}
24     return min_{x ∈ PartSolutions[m]}{x.costs}
25
26
27 CombineStructureTrees(PartSol lSubttree, PartSol rSubttree)
28     //disregard pairs that overlap
29     if (lSubtree.covAPPs ∩ rSubtree.covAPPs ≠ ∅)
30       return null;
31
32     Set covAPPs = lSubtree.covAPPs ∪ rSubtree.covAPPs;
33     Integer appCoeff = appCoeff(lSubtree.app, rSubtree.app);
34     Set detCoeffs = {detCoeff(lSubtree.app, rSubtree.app)}
35                     ∪ lSubtree.detCoeffs ∪ rSubtree.detCoeffs;
36     NestedSet strucTree = {lSubtree.strucTree, rSubtree.strucTree};
37     return (covAPPs, appCoeff, detCoeffs, strucTree, costs(detCoeffs));
```

Figure 3: DP for computing the optimal structure tree – a benchmark for the heuristical approach

to consider the size of intermediate results, not only the size of the *final synopsis*. Formally, the problem now is minimizing the data volume of *all the synopses*.

We illustrate the intricacy of this requirement from the point of view of a single node. The problem is that the knowledge of a node does not suffice to decide which coefficients to discard: If the node discards a detail coefficient, the synopsis comes closer to the maximum error. However, there might be smaller detail coefficients to come and thus, it would have been better to save the error budget.

The second problem of discarding coefficients is metadata overhead, in two ways. First, guaranteeing a maximum error requires each node to know the error budget. Assigning fixed error budgets per node would result in very small budgets, making it difficult to discard any coefficients. [22] proposes a solution for the centralized case: One needs to keep track of the error that has already been introduced. They show that this can be done effectively by maintaining a range [min, max] for the error. If we transfer this approach to our scenario, this enlarges each intermediate wavelet that is forwarded by two additional numbers per attribute.

There is an even worse overhead: We need to know the coordinates of the remaining coefficients to reconstruct the data. This is done by storing the coefficients as <coordinate, coefficient>-tuples. Thus, for a synopsis of a single attribute the coordinates double the data volume. Reducing this overhead has been addressed in different ways, e.g., [9, 33]. A distributed approach to reduce the overhead due to the coordinates is an open problem.

5.2 Foundations for Compact Synopses

To avoid these problems, we base SNAP on a different mechanism for obtaining synopses. It is inspired by work on image compression and consists of three steps:

1. *Quantization*
2. *Integer Wavelet Transform*
3. *Entropy Coding* (Huffman Coding)

This alternative mechanism does not solve our problem of distributing the construction of synopses – this is discussed in Section 5.3 – but it is key to overcome the difficulties of discarding coefficients: Our mechanism does not do any such discarding. The error bound is exploited for the quantization which affects only a single tuple. This eliminates the need to forward error budgets. Finally, we can avoid sending coordinates along with the coefficients.

Intuition. The important step to obtain a synopsis is Step (3), the entropy coding. The idea is to use short codes for frequent symbols and longer ones for less frequent symbols. In our case, the symbols to encode are the detail coefficients. The performance of the entropy coding depends on (a) the distribution of the symbols (the more skewed the distribution is, the higher the effectiveness) and (b) the number of different symbols that appear (the more symbols there are, the more bits are required).

We can regard Steps (1) and (2) as creating the optimal conditions for an entropy coding. Step (1) gives way to a small number of different symbols. Each node quantizes its sensor readings based on the error bounds of the attributes. As a result, the infinite set of possible readings is turned into a small set. Most notably, an increase in the maximum error in the query reduces the number of different symbols. The wavelet transform (Step 2) achieves a skewed distribution. Recall that the wavelet transform is used to generate small detail coefficients. Most notably, this usually results in a distribution of the detail coefficients that is bell shaped around zero [4]. Finally, we use an integer version of the transform: While the quantization reduces the set of possible values, the regular Haar transform involves divisions by two and thus would re-enlarge this set. An integer transform avoids this problem as it yields coefficients within the input set. We will provide the details right away.

Quantization. Step (1) combines a quantization with a scaling such that each quantized value is a (positive) integer. This is required for the integer wavelet transform. Let x_i be the sensor reading of node i of an attribute, and let e be the error bound for this attribute. Finally, let *minVal* be an arbitrary lower bound for the sensor readings. Each node quantizes its measurements to obtain the approximation coefficient to start the transform with (level 1):

$$a_{1,i} = Int(\frac{x_i - \mathit{minVal}}{2 \cdot e}). \quad (1)$$

Here, $Int(\cdot)$ denotes the usual integer rounding. The base station can easily reconstruct the values as $x'_i = \mathit{minVal} + a_{1,i} \cdot 2 \cdot e$. Given $a_{1,i}$, x'_i is within $[x_i - e, x_i + e]$.

Integer Wavelet Transform. The integer version of the Haar transform is the S-transform [2], where l is the level in the structure tree and p the index of the coefficient at level l:

$$a_{l,p} = \lfloor \frac{a_{l-1,2p} + a_{l-1,2p+1}}{2} \rfloor, d_{l,p} = a_{l-1,2p+1} - a_{l-1,2p}$$

We use it with a minor modification. The S-transform either does no rounding or rounds down. Thus, the approximation coefficients systematically become smaller with the levels in the structure tree. It turns out that estimating the frequencies of the detail coefficients is much easier if we avoid this systematic deviation, see Theorem 2. As can be seen from the corresponding proof, this systematic deviation can be avoided by equally rounding up and down:

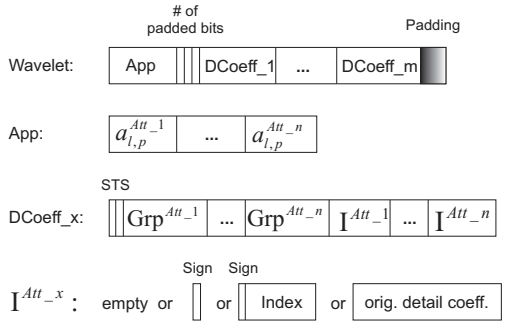

Figure 4: Format of Wavelets (Encoding)

$$a_{l,p} = \begin{cases} \lfloor \frac{a_{l',p'} + a_{l'',p''}}{2} \rfloor & \text{if } a_{l',p'} \geq a_{l'',p''} \\ \lceil \frac{a_{l',p'} + a_{l'',p''}}{2} \rceil & \text{otherwise} \end{cases} \quad (2)$$

$$d_{l,p} = a_{l'',p''} - a_{l',p'} \quad (3)$$

As the structure tree is not balanced in our case, it is not obvious how the level l is defined: We define it as the number of sensor readings that a coefficient covers. This implies that $l = l' + l''$. p still denotes the index. The inverse is given by:

$$a_{l',p'} = \begin{cases} a_{l,p} - \lfloor \frac{d_{l,p}}{2} \rfloor & \text{if } d_{l,p} \leq 0 \\ a_{l,p} - \lceil \frac{d_{l,p}}{2} \rceil & \text{otherwise} \end{cases}$$

$$a_{l'',p''} = d_{l,p} - a_{l',p'}$$

Entropy Coding. While arithmetic coding [29] usually is state-of-the-art for entropy coding, it is suboptimal in our case: It requires adding a few bits to each sequence of symbols to obtain unique encodings. Since our sequences consist of only a few symbols (usually clearly less than ten, depending on the number of attributes in the query) this is a substantial overhead. We use Huffman coding [18] which is well-known to yield better results for such short sequences and is standard in this case. In addition, the low storage requirements and its simplicity make it a good choice for resource-constrained sensor nodes. Irrespective of the coder used, there is a problem when it comes to distributing the approach. Assigning codes requires knowledge on the frequency of the symbols by the time of encoding. This is the main challenge when distributing the approach and is addressed in Section 5.3.

Implementation issues. We conclude the description of the framework with two details. First, we introduce the concept of "groups". It is standard in entropy coding to cope with large numbers of symbols. Second, we describe the format of wavelets sent.

"Groups" of symbols are used to reduce the number of symbols in entropy coding. The idea is to form groups of detail coefficients with similar frequencies. As a result, the number of groups is much smaller than the number of symbols, and each group is much more frequent. Each group is then assigned a Huffman code, i.e., only the group number is entropy-encoded. An index is used to discriminate between the members of each group. For instance, the detail coefficient 0 results in a single group. $\{-1, 1\}$ might be the second group. In particular, if a group contains x, it also contains $-x$, as our distribution is symmetric. $\{-3, -2, 2, 3\}$ might be the third group, etc. The final group contains the tail of the distribution. It contains all coefficients that are unlikely to appear.

The format used to send wavelets is shown in Figure 4. Basically, a wavelet is a bit stream which is padded at the end to align it with byte boundaries. At a high level, the wavelet starts with the approximation coefficients, one per attribute that is queried. A node then inserts three bits ("padding size") representing the size of the padding, followed by the detail coefficients. Their ordering corresponds to a depth first traversal of the structure tree. This allows to construct wavelets with ease: A node takes two wavelets as received from its children. Based on their approximation coefficients, it computes a new approximation and a detail coefficient. The latter is first in depth-first ordering. The node then simply appends the detail coefficients from the left child in the structure tree as received, followed by those from the right child. Also, it removes the padding from its children in this append step. Given the padding size, this can be done without decoding the detail coefficients.

Figure 4 further shows the encoding of the detail coefficients: Each one starts with two bits called Subsequent Tree Structure (STS). As our structure tree is irregular, this is required to reconstruct the tree at the base station. STS encodes whether a node has two children, one left child, one right child, or no children. Thus, the purpose is similar to the coordinates of coefficients. Following the STS bits, the Huffman code of the group numbers are listed, one for each attribute. Finally, the indices follow. In case of the zero coefficient, no index is required. Otherwise, the indices start with a single bit which encodes the sign of the coefficient. For groups of size greater than 2, an index number follows that identifies the unsigned member. This number cannot be compressed since the group members are equally frequent. However, it makes sense to have groups whose size is a power of 2 to fully exploit this index number. The group that encodes the tail of the distribution is special: Here, the original detail coefficient is used.

5.3 Distribution by Estimating Frequencies

The mechanism described so far requires to know the frequencies of the symbols in the overall synopsis. This is the main challenge when distributing this approach. As the synopsis is now created incrementally, the frequencies are unknown by the time of encoding. Also, they strongly depend on the error bound e.

A common approach in entropy coding is to initially assume that each symbol is equally frequent and to adjust the frequencies continuously with each symbol. This approach is not applicable in our case. Each leaf node would start with a frequency distribution that is far from the actual one. This results in a bad compression. This distribution would only become better close to the root. Additionally, a node at a higher level of the tree receives encoded coefficients. To get to a count of the coefficients seen so far, the node would have to decode them. In the following, we propose an approach that lets each node *estimate* the frequency of a coefficient in the (unknown) final synopsis. Given this estimation approach, the previous framework can be executed in a distributed environment.

5.3.1 Estimating Frequencies

Our approach is based on two ideas. In the style of selectivity estimation, the base station continuously keeps track of the frequency distribution of the detail coefficients. I.e., the first idea is to estimate the frequencies based on experience from previous queries. The main problem is that the frequencies strongly depend on the maximum error e. The second idea addresses this issue: The distribution is maintained for an error $e = 0$. As illustrated in the following, this corresponds to a *continuous* distribution of detail coefficients. *Given that the nodes know this distribution, it is now possible for each node to estimate the frequency of a detail coefficient for $e \neq 0$ by discretizing the continuous distribution according to e.*

We start by illustrating the idea of discretizing a continuous distribution. Suppose that we know the distribution $pdf(x)$ of a physical quantity, e.g., temperature. Figure 5(a) serves as an illustration.

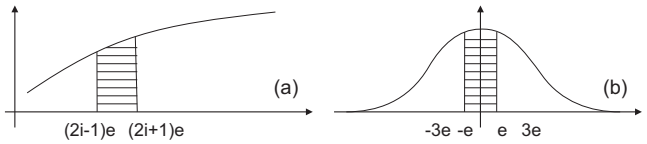

Figure 5: Estimating frequencies

Formally, $pdf(x)$ is a probability distribution function. Discretizing the quantity corresponds to mapping all measurements of an interval $[(2i-1)\cdot e, (2i+1)\cdot e]$ to $2i\cdot e$ (or i after scaling). Thus, the relative frequency of i is the probability that a sensor reading is in this interval. Formally, let n be the number of sensor readings. The absolute frequency of i is:

$$freq(i) = n \cdot \int_{(2i-1)\cdot e}^{(2i+1)\cdot e} pdf(x)dx \quad (4)$$

To apply this idea of quantizing a continuous distribution, we need to define a "*continuous* distribution of the detail coefficients $pdf_0(x)$". $pdf_0(x)$ is the distribution of the detail coefficients of a slightly modified transform ($e = 0$): It omits the quantization as there is no error budget. In addition, as it is exact, there is no rounding in the wavelet transform:

$$\hat{a}_{1,i} = x_i - \text{minVal}$$
$$\hat{a}_{l,p} = \frac{\hat{a}_{l',p'} + \hat{a}_{l'',p''}}{2}$$
$$\hat{d}_{l,p} = \hat{a}_{l'',p''} - \hat{a}_{l',p'}$$

Given the distribution $pdf_0(x)$ of $\hat{d}_{l,p}$, we estimate the frequency of $d_{l,p}$ by integration as in Equation 4 (cf. Figure 5b).

5.3.2 Requirements

The previous approach is a valid estimation if the following Assumption A1 and Proposition P1 hold. We justify them in Section 5.3.4. As the estimation is based on experience from previous executions, we assume that:

A1: $pdf_0(x)$ is largely unchanged since its last update.

As we base the estimation on a slightly modified transform, we have to prove the following proposition:

P1: The distribution of the unscaled detail coefficients $d_{l,p}\cdot 2e$ of our actual transform corresponds to the one of the continuous detail coefficients $\hat{d}_{l,p}$.

$d_{l,p}\cdot 2e$ are the quantized detail coefficients except for scaling to an integer. The scaling depends on e and therefore is not captured in the distribution $pdf_0(x)$, but in the integration – we integrate over $[(2d_{l,p}-1)\cdot e, (2d_{l,p}+1)\cdot e]$ (cf. Equation 4).

5.3.3 Maintaining Frequency Distributions

We have already described how each node estimates the frequency of $d_{l,p}$ from $pdf_0(x)$. We now discuss how to maintain $pdf_0(x)$. As a starting point for the maintenance, suppose that we have collected the continuous detail coefficients empirically. The idea is to obtain $pdf_0(x)$ by fitting a curve to their distribution.

Fitting a curve to a set of detail coefficients requires to specify a family of functions (polynomial, Gaussian, etc.) to be used. Bell curves often are well suited to describe the detail coefficients of a wavelet transform, in particular generalized Gaussian distributions, e.g., [4]. However, they are not a good choice in our case: They are computationally complex, and there is no antiderivative. The nodes would have to use numerical integration over a computationally complex function to arrive at an estimate. We found that the following simple bell curve describes the distribution well: $pdf_0(x) = \frac{1}{\pi}\frac{a_0}{1+(a_0x)^2}$. The parameter a_0 describes the spread of the coefficients around 0 and has to be determined by curve fitting.

Note that, while $pdf_0(x) = \frac{1}{\pi}\frac{a_0}{1+(a_0x)^2}$ is well-suited for our data, our technique is orthogonal to the curve function used.

Fitting $pdf_0(x)$ to the empirically determined detail coefficients is difficult due to outliers, i.e., isolated large coefficients. They yield a spread parameter a_0 which is too large. Thus, the tail of the distribution is too heavy, and the estimate of the number of frequent coefficients is too low. A simple extension would be to estimate the distribution on an α-quantile, i.e., to discard the outliers. But in this case the distribution is highly sensitive to the choice of α.

Instead of using $pdf_0(x)$, we estimate its cumulative distribution $cdf_0(x) = \frac{1}{2} + \frac{1}{\Pi}arctan(a_0x)$. This has the following advantages over using $pdf_0(x)$: (a) The nodes do not have to solve an integral. Given $cdf_0(x)$, the frequency of $a_{1,i}$ is simply $freq(a_{1,i}) = n[cdf_0([2a_{1,i}+1]\cdot e) - cdf_0([2a_{1,i}-1]\cdot e)]$. (b) Estimating $cdf_0(x)$ can be done robust to outliers by using an α-quantile. This is now insensitive to the choice of α: By first accumulating the frequencies and then discarding the tails, we keep the information that there were further coefficients. In a way, we only discard their "wrong" position. (c) $cdf_0(x)$ can also be estimated from the quantized detail coefficients. This is key to react to changes in the distribution: We update $cdf_0(x)$ every time the attribute has been queried.

In summary, our approach works as follows:

1. **Initialization.** Prior to executing any query, the base station once collects the continuous detail coefficients for each attribute, determines the cumulative frequencies, and calculates a_0. In our implementation, we use a standard curve fitting algorithm (Levenberg-Marquardt, [26]).
2. **Usage.** Given a query, for each of the attributes the corresponding a_0 and e are distributed in the network along with the query. Thus, every node arrives at the same estimates of the frequencies and thus at the same encoding.
3. **Update.** After execution, the base station re-estimates a_0. A moving average continuously adapts the distribution.

5.3.4 Justification

We now justify A1 and prove P1. Assumption A1 states that the distribution of the detail coefficients $pdf_0()$ is largely unchanged since its last update. If it does not hold, the effectiveness of the coding degrades, the more the current distribution of the detail coefficients deviates from $pdf_0()$. That distribution may change due to changes in the data being transformed. More precisely, (major) changes in the *relative differences* of the data might affect $pdf_0()$. In contrast, spatial and temporal correlation of the sensor readings lead to stable differences. Changes in $pdf_0()$ can also stem from changes in the structure tree. But this is even less critical. First, recall that the underlying routing tree dictates its coarse structure. While routing trees change from time to time due to links going down, etc., such changes affect the tree only locally. Given that the differences of the approximations that a node receives are similar, the resulting structure tree is roughly the same as well.

Proposition P1 is that the distribution of the unscaled detail coefficients $d_{l,p}\cdot 2e$ corresponds to the one of the continuous coefficients $\hat{d}_{l,p}$. This will be shown as follows: Starting with a continuous distribution we examine the influence of (a) the quantization and (b) the rounding on the continuous coefficients. We show that:

1. The expected value of the coefficients remains unchanged.
2. The variance of the detail coefficients is small (usually much smaller than e).

(2) implies that the expected value represents the detail coefficients well. Given (1), Proposition P1 follows.

What is difficult in proving the statements is to model the influence of the quantization and the rounding appropriately. We will present this in detail. Given the idea, some proofs are obvious and will only be sketched. In order to calculate the expected value and variance of the detail coefficients we will introduce the random variables $A_{l,p}, D_{l,p}, R_{1,i}, T_{l,p}$. We start by defining a random variable $R_{1,i}$ modeling the quantization (cf. Equation 1). Without the scaling the coefficients are quantized to a multiple of $2e$. As each value in $[-e, e]$ is equally likely, $R_{1,i}$ is uniformly distributed:

Definition 1: $R_{1,i}$ is a random variable that is uniformly distributed in $[-e, e]$.

It follows that $R_{1,i}$ has mean $\mu = 0$ and variance $\sigma^2 = \frac{e^2}{3}$.

Definition 2: Let $A_{1,i} := \hat{a}_{1,i} + R_{1,i}$ be the random variable that models the quantized $\hat{a}_{1,i}$.

Lemma 1: $E(A_{1,i}) = \hat{a}_{1,i}$ and $V(A_{1,i}) = \frac{e^2}{3}$ where $E(\cdot)$ is the expected value and $V(\cdot)$ is the variance.

Proof: *(Sketch)* This can be seen by simply substituting the definition of $A_{1,i}$ and regarding Definition 1.□

Definition 3: For $l' + l'' = l, l > 1$, let $A_{l,p} := \frac{A_{l',p'} + A_{l'',p''}}{2} + T_{l,p} \cdot e$. Here, $T_{l,p}$ is a random variable $\in \{-1, 0, 1\}$. Further, let $R_{l,p} := \hat{a}_{l,p} - A_{l,p}$ for $l > 1$.

$T_{l,p} \cdot e$ models the effect of rounding in the integer transform on the unscaled coefficients.

Lemma 2: $E(T_{l,p}) = 0$ and $V(T_{l,p}) = \frac{1}{2}$.

Proof: *(Sketch)* For the rounding, we distinguish between four cases (cf. Equation 2): If $a_{l',p'}$ and $a_{l'',p''}$ are both even or uneven, there is no rounding. That is, in two out of four cases, $T_{l,p} = 0$. Otherwise, we either round up ($T_{l,p} = 1$) if $a_{l',p'} > a_{l'',p''}$ or down ($T_{l,p} = -1$) if $a_{l',p'} < a_{l'',p''}$. Both cases are equally likely. The lemma then follows from the definition of E and V.□

Definition 4: Let $D_{l,p} := A_{l'',p''} - A_{l',p'}$.

Theorem 2: $E(A_{l,p}) = \hat{a}_{l,p}$. $E(D_{l,p}) = \hat{d}_{l,p}$.

Remark: As $D_{l,p}$ incorporates the quantization and the rounding, the second statement in the theorem is actually our first claim (1) for justifying P1.

Proof: The first statement can be proven by induction on l. For $l = 1$, the statement is true due to Lemma 1. Assuming $E(A_{l',p'}) = \hat{a}_{l',p'}$ and $E(A_{l'',p''}) = \hat{a}_{l'',p''}$ we get: $E(A_{l,p}) = E(\frac{A_{l',p'} + A_{l'',p''}}{2} + T_{l,p} \cdot e) = \frac{1}{2}E(A_{l',p'}) + \frac{1}{2}E(A_{l'',p''}) + E(T_{l,p}) \cdot e = \frac{1}{2}\hat{a}_{l',p'} + \frac{1}{2}\hat{a}_{l'',p''} + 0e = \hat{a}_{l,p}$. The second statement can be obtained directly: $E(D_{l,p}) = E(A_{l'',p''} - A_{l',p'}) = E(A_{l'',p''}) - E(A_{l',p'}) = \hat{a}_{l'',p''} - \hat{a}_{l',p'} = \hat{d}_{l,p}$.□

Lemma 3: $V(A_{l,p}) < e^2$.

Proof: (Induction) Note that $V(A_{l,p}) = V(R_{l,p})$ by definition. For $l = 1$, Lemma 3 is true due to Lemma 1.
If $V(A_{l',p'}) = V(R_{l',p'}) < e^2$, $V(A_{l'',p''}) = V(R_{l'',p''}) < e^2$ we get: $V(A_{l,p}) = V(\frac{A_{l',p'} + A_{l'',p''}}{2} + T_{l,p} \cdot e)$
$= V(\frac{\hat{a}_{l',p'} + R_{l',p'} + \hat{a}_{l'',p''} + R_{l'',p''}}{2} + T_{l,p} \cdot e)$
$= \frac{1}{4}V(R_{l',p'}) + \frac{1}{4}V(R_{l'',p''}) + e^2 \cdot V(T_{l,p}) < \frac{1}{4}e^2 + \frac{1}{4}e^2 + e^2 \cdot \frac{1}{2}$
$= e^2$.□
The equality in the second to last line holds as R and T model roundings and can be assumed to be stochastically independent.

Remark: In this general case, e^2 is the smallest upper bound that can be proven for the variance. It is possible to show via induction that for a perfectly balanced structure tree that covers l readings, $V(A_{l,p}) = e^2 - \frac{1}{l}\frac{2}{3}e^2$ which quickly converges to e^2.

Theorem 3: $V(D_{l,p}) < 2e^2$.

Proof: $V(D_{l,p}) = V(A_{l'',p''} - A_{l',p'}) = V(\hat{a}_{l'',p''} + R_{l'',p''} - \hat{a}_{l',p'} - R_{l',p'}) = V(R_{l'',p''}) + V(R_{l',p'}) < e^2 + e^2 = 2e^2$.□

Theorem 2 is the second statement (2) for justifying Proposition P1 and thus completes our discussion.

6. SENDING APPROXIMATIONS

We conclude the description of SNAP with an optimization regarding sending of approximation coefficients. It is based on the observation that the quantization not only leads to a small set of detail coefficients – this is exploited by the entropy coding. It also restricts the set of approximation coefficients to a small number.

Consider a system that collects temperature values with $e = 0.1°$C. Suppose that the measurements within the network range between 18.5274°C and 23.3883°C. Then the approximation coefficients must also be within this range. Most notably, there are only $\lceil \frac{23.3883 - 18.5274}{2 \cdot 0.1} \rceil = 25$ possible values. In this case, sending an approximation should not require more than $\lceil log_2(25) \rceil = 5$ bits.

Underlying idea. Intuitively, the approximation coefficients will mostly vary in a small range around the overall average $avg_{\texttt{net}}$. Assume for now that $avg_{\texttt{net}}$ is known. If Node n_i computes an approximation coefficient $a_{l,p}$ and then computes its difference from the quantized overall average $\bar{a}_{l,p} = Int(\frac{avg_{\texttt{net}} - minVal}{2e}) - a_{l,p}$, this difference $\bar{a}_{l,p}$ will be a small number. Thus its binary representation will be of one of the following forms: $0...01X_0...X_r$, $1...10X_0...X_r$, or $0...0$ for $a_{l,p} <, >$ or $= avg_{\texttt{net}}$ (where $X_i = \{0|1\}$). If we know the remainder ($01X_0...X_r$, $10X_0...X_r$, or $0...0$), we can restore $\bar{a}_{l,p}$ by prefixing 0's or 1's depending on the first bit. I.e., there is no information in the beginning of $\bar{a}_{l,p}$. *The idea is to only send this remainder.* Technically, this requires knowing the length of the remainder, which depends on the quantization.

Approach. In general, our optimization works as follows: Assume that each node knows $avg_{\texttt{net}}$ and the range r. The latter is defined as the difference of the max and the min value measured. To send the information contained in $a_{l,p}$, n_i computes $\bar{a}_{l,p}$. It then truncates $\bar{a}_{l,p}$ to the last $\lceil log_2(\lceil \frac{r}{2e} \rceil) \rceil$ bits. The parent of n_i reconstructs $a_{l,p}$ by filling up the remainder and adding it to $avg_{\texttt{net}}$.

Note that $\lceil log_2(\lceil \frac{r}{2e} \rceil) \rceil$ bits are only sufficient if both values $avg_{\texttt{net}}$ and r are accurate. Therefore, the encoding actually used by SNAP is as follows: If $\lceil log_2(\lceil \frac{r}{2e} \rceil) \rceil$ bits are sufficient, a node starts the encoding with a '0'-bit followed by the remainder. 'sufficient' means that at least $01X_0...X_r$ or $10X_0...X_r$ can be captured to achieve uniqueness. If this condition is not fulfilled, the node starts the encoding with '10' followed by $\lceil log_2(\lceil \frac{r}{2e} \rceil) \rceil + 1$ bits. Note that this effectively doubles the range. In case of any errors, the encoding can start with '11' followed by the original $a_{l,p}$.

Finally, we say how the nodes get to know $avg_{\texttt{net}}$ and r. We do not require each node to know the max and min value but only the range which is much more stable in time. The base station simply maintains these parameters as part of its catalog. In detail, the base station keeps moving averages for $avg_{\texttt{net}}$ as well as for the min and max value per attribute. $avg_{\texttt{net}}$ or r are distributed along with the query whenever its deviation from the value currently used exceeds a threshold. Determining these thresholds is straightforward. The question is when the length of the remainder changes. If $avg_{\texttt{net}}$ is correct, the length changes if the range doubles or halves. If r is valid, we need to update $avg_{\texttt{net}}$ if it deviates by more than r. Overall, the combination of both parameters must not deviate by more than r from the values currently used. This budget should not be fully exploited since the encoding gets worse if the deviations come close to these upper bounds. In our implementation, we update r if the range halves or becomes larger by $\frac{1}{3}$. We update $avg_{\texttt{net}}$ whenever it changes by more than half of the range currently used.

7. EVALUATION

In this section, we demonstrate the performance of SNAP based on real-world and synthetic sensor data. We will show that SNAP efficiently consolidates sensor relations even for tight error bounds.

7.1 Experimental Setup

We implemented a prototype of SNAP in the ns-2 network simulator. It is a widely used simulator that allows for a controlled environment of our experiments and ensures repeatability.

Query Workload. Our evaluation is based on queries that match the pattern from our problem specification (cf. Section 3.3). There are two degrees of freedom in this pattern: (1) the number of attributes in the SELECT-clause and (2) the error budget for each of them. Both will serve as parameters in our experiments.

Comparison Schemes. In line with the discussion in Section 2, we compare SNAP to the following schemes:

(1) Wavelet-based approaches. We want to highlight the benefit of our integration approach of building upon an unmodified routing tree and using an irregular transform. Two alternatives have been proposed: Using zero padding to account for the irregular routing tree [16] and subdividing the network into clusters, each of which applies a wavelet transform [8]. However, these approaches provide no error guarantees and are not designed to cope with multiple attributes. We extended them based on state-of-the-art approaches for dealing with tuples [9] and for thresholding with error guarantees [22]. In our experiments, we do not account for any overheads due to these extensions like sending coordinates of the coefficients or forwarding the remaining error budgets. Thus, we heavily underestimate the costs of the related approaches.

(2) Model-based approaches. We compare SNAP to approaches based on multivariate Gaussians. While these sophisticated models have high training costs, they are best when it comes to normal operation. (We do not account for training costs in our experiments.) [5] has shown that Ken is superior to Caching and Kalman Filters. Among Ken and BBQ [11], we decided to compare SNAP to Ken since Ken provides the same error guarantees as SNAP. Note that the problem of model-based approaches are false predictions for tight error bounds, which is the same for BBQ and Ken.

(3) Tree-based Data Collection. We compare SNAP to simple tree-based data collection (TDC). As we will show, for the related approaches, TDC is among the best for tight error bounds.

Data sets and Setting. Our goal is to evaluate SNAP on real-world data sets, i.e., traces from WSNs. We used two data sets in our experiments: The LUCE data [35] and the Intel lab data [19]. SNAP performs equally well on both of them. For brevity, we report on results from LUCE as it covers more attributes. The data is from a network consisting of 81 nodes when the data was acquired (January 2007). For the experiments based on real data, we set the size of the network and the relative positions of the nodes to reflect the setting during the original data collection. We also use some synthetically generated data sets. Synthetic data is necessary to evaluate SNAP on large networks and allows us to study extreme cases. In these experiments we distribute the nodes randomly.

Metrics. Our intention is to capture the communication costs, as they dominate the power consumption. In the literature, there are two common metrics: Number of bytes transferred [24] and number of transmissions (networking packets) [5]. We use both. For the second metric we set the maximum packet size to 127 bytes as used by SunSPOTs. Most of the graphs in this text are based on the other metric: We have conducted our experiments on real data where the size of the network is 81 nodes. For such sizes, SNAP builds synopses that fit into a single networking packet. While this is nice, the problem is that varying parameters has no effect on the number of packets sent and is pointless. In contrast, the number-of-bytes metric is well suited to show the data reduction that SNAP achieves. This carries over to the number-of-transmissions metric, as we will show as well. Note that the number-of-bytes metric is independent of the maximum packet size.

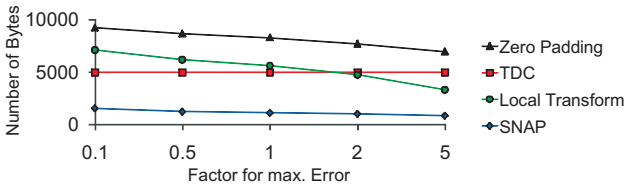

Figure 6: Comparison of integration approaches (real data)

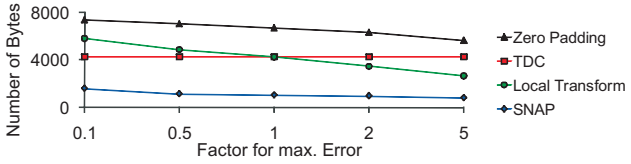

Figure 7: Comparison of integration approaches (synth. data)

Default setting. In each experiment we vary one parameter. If a parameter is not varied we use the following default value: Among the attributes of the data set, we query the node ID (no error), ambient temperature ($e = 0.1^\circ$C), surface temperature ($e = 0.1^\circ$C) and relative humidity ($e = 0.5\%$). These attributes were available at all nodes in the data set. Note that querying for multiple attributes without involving the IDs is rare in practice.

Varying error bounds. As our queries cover multiple attributes, if we want to vary the maximum errors, we have to consider all of the attributes. We simply multiply their default settings with a factor, ranging from 0.1 to 5. E.g., for the ambient temperature (default setting $e = 0.1^\circ$C), the errors range from 0.01°C to 0.5°C.

7.2 Comparative Experiments

Wavelet-based approaches. In a first set of experiments we examine building upon an irregular transform. We compare SNAP to zero padding and cluster-based local transforms. The objective is to give evidence for the following claims: (1) Zero padding results in *many, large* detail coefficients. (2) A cluster-based local transform also suffers from inherent problems and thus cannot result in small synopses: Only the data within each cluster is decorrelated – redundancies between clusters are not removed. In addition, as the number of nodes within a cluster is rarely a power of two, we again need zero padding to perform local wavelet transforms, though less than in the tree-based approach.

We compare the approaches for different maximum errors. We expect all of them to perform better for larger error bounds. Next to the two wavelet-based approaches and SNAP, we also measure the performance of TDC, which is independent of the error. The results are shown in Figure 6. They confirm our expectation. The related approaches perform worse than a simple TDC. Recall that the costs of the wavelet-based approaches are underestimated substantially. In contrast, SNAP outperforms TDC by a factor of about five.

We repeated these experiments on our synthetic data, see Figure 7. They are in line with those on real data. All approaches including TDC are slightly better due to differences in the routing tree. For the synthetic data we placed the nodes randomly.

Model-based approaches. [11] has pointed out that model-based approaches degrade with tighter error bounds. We want to show that the performance of SNAP does not degrade as much. Note that model-based approaches need updates of the model from time to time, as otherwise all estimations would be out of bounds after a while. Thus, they only make sense if the network is queried

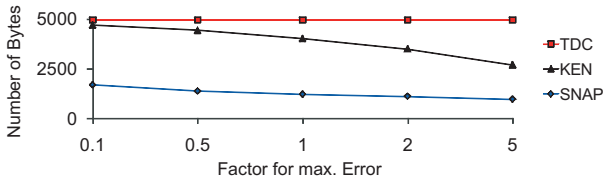

Figure 8: Comparison to Ken (real data)

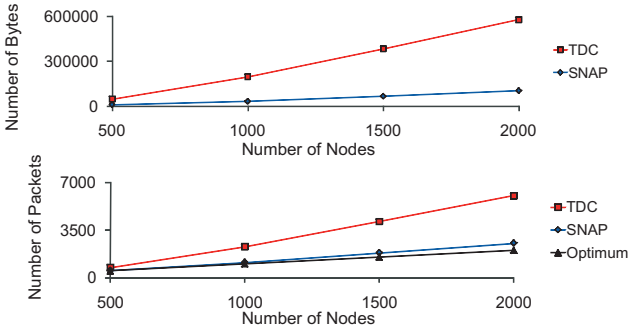

Figure 9: Scalability of SNAP (synthetic data)

continuously. We measure the performance of Ken by using a continuous query and averaging the costs. We query 40 consecutive snapshots from the real world data set after an extensive training phase. SNAP is executed on the same 40 snapshots, and we also average the costs. The results are shown in Figure 8. Note that our results for Ken are in line with those presented in [5]. Most notably, if the error bounds become less than our default setting, Ken performs similar to TDC. In contrast, even for bounds an order of magnitude tighter (factor $\frac{1}{10}$), SNAP still achieves a reduction of the communication costs by factor three.

7.3 Analysis of SNAP

We now study the performance of SNAP. As the performance of TDC is close to Ken for our default setting, and TDC is relatively easy to handle, we use TDC as a point of comparison.

Scalability. We are interested in the performance of SNAP for larger networks. To determine the influence of the network size we vary the number of nodes from 500 to 2000. At the same time we vary the area of the network to keep the node density constant. The experiments are conducted on synthetic data as we do not have data sets for larger networks. Intuitively, the relative savings of SNAP over TDC should be independent of the network size. This is because our synopsis achieves a constant compaction factor of the data. The results in Figure 9(a) confirm this expectation. To confirm that the data reduction carries over to the number of transmissions, Figure 9(b) graphs the results for this metric. Interestingly, even for 2000 nodes, SNAP sends only about 2500 packets. That is, SNAP reduces the data such that most of the nodes send one packet. This is the optimum for the number-of-transmissions metric, indicated by the black line.

Number of attributes. In this set of experiments we determine the influence of the number of attributes in the query on SNAP. If the query exceeds three attributes (plus ID), we additionally query for solar radiation ($e = 0.5\frac{W}{m^2}$) and wind speed ($e = 0.1\frac{m}{s}$). We conduct the corresponding measurements on synthetic data because not all of the nodes in our data set have available the complete set of sensors. Intuitively, the number of attributes should not have a major influence on the relative performance of SNAP over TDC.

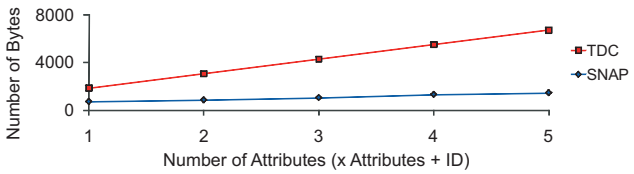

Figure 10: Influence of the number of attributes (synth. data)

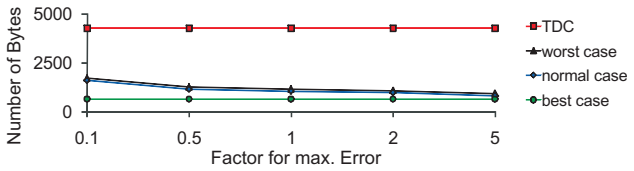

Figure 11: SNAP: performance on extreme data (synth. data)

On the one hand, more attributes better amortize the overhead in the detail coefficients for coding the tree structure (two STS bits). On the other hand, this might require tradeoffs in building the structure tree – different attributes might prefer different tree structures. Figure 10 confirms that both influences are minor. The data reduction becomes slightly better with the number of attributes as the ID, which is included in all queries, is difficult to compress.

Heuristical vs. optimal structure tree. The following experiment highlights the appropriateness of our heuristic to devise a structure tree. We compare SNAP (which incorporates the heuristic) to a modified version of SNAP that uses the optimal algorithm. Both approaches integrate the structure tree into the routing tree. Thus, the rough structure is the same. Our result is that the heuristic performs within 5% of the optimum.

Extreme data sets. Finally, we are interested in how much the performance of SNAP depends on spatial correlation in the data. Therefore, we evaluate SNAP on two extreme data sets. The first one simulates perfect correlation, i.e., each node measures the same value on corresponding attributes. This is supposed to be the best case for SNAP. The second data set simulates uncorrelated data as a worst case. We set the ranges of the attributes as observed in our real data and let each node observe a random value from the ranges for each of the attributes. The results are shown in Figure 11. As expected, the performance of SNAP degrades with less correlation. However, the sensitivity is limited. SNAP performs well even on the uncorrelated data. This also explains why we observe a similar performance of SNAP on both real-world data sets.

8. CONCLUSIONS

This paper studies the evaluation of queries with low selectivity in sensor networks. Prior work has addressed non-selective queries by approximating results based on models. The solutions work well if the accuracy requirements are loose. For more accuracy, communication costs increase quickly. This paper has presented SNAP, a wavelet-based approach for efficient consolidation of sensor relations. SNAP constructs the synopsis during data collection incrementally. As a core contribution, we have shown how to distribute the wavelet transform and the thresholding step. We have done so by integrating the transform into an unmodified routing tree. To obtain a synopsis we have explored a design that encodes coefficients compactly instead of discarding them. To distribute this mechanism, we have proposed an approach to estimate the frequencies of coefficients. SNAP is the first distributed solution to the threshold-

ing problem with error guarantees. It achieves a data reduction by more than a factor of five and improves the accuracy for which data can be efficiently consolidated by more than an order of magnitude.

Acknowledgements. This work was partially supported by the German Research Foundation (DFG) within the Research Training Group GRK 1194 "Self-organizing Sensor-Actuator Networks" (GRK1194) and by the Landesstiftung Baden-Württemberg as part of Project "ZeuS". We are grateful to Björn Reuber for much help.

9. REFERENCES

[1] J. Acimovic, R. Cristescu, and B. Beferull-Lozano. Efficient distributed multiresolution processing for data gathering in sensor networks. In *ICASSP*, 2005.

[2] A. R. Calderbank, I. Daubechies, W. Sweldens, and B. lock Yeo. Wavelet transforms that map integers to integers. *J. Appl. Comput. Harmonic Anal.*, 5:332–369, 1998.

[3] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB*, 2000.

[4] S. G. Chang and M. Vetterli. Adaptive wavelet thresholding for image denoising and compression. *IEEE Transactions on Image Processing*, 9(9):1532–1546, 2000.

[5] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, 2006.

[6] A. Ciancio and A. Ortega. A distributed wavelet compression algorithm for wireless multihop sensor networks using lifting. In *ICASSP*, 2005.

[7] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[8] X. T. Dang, N. Bulusu, and W. chi Feng. Rida: A robust information-driven data compression architecture for irregular wireless sensor networks. In *EWSN*, 2007.

[9] A. Deligiannakis, M. Garofalakis, and N. Roussopoulos. Extended wavelets for multiple measures. *ACM Transactions on Database Systems (TODS)*, 32(2):10, 2007.

[10] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *EDBT*, 2004.

[11] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.

[12] M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *ACM SIGMOD*, 2002.

[13] M. Garofalakis and A. Kumar. Deterministic wavelet thresholding for maximum-error metrics. In *PODS*, 2004.

[14] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, 2001.

[15] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, 2004.

[16] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. In *IPSN*, 2003.

[17] J. M. Hellerstein and W. Wang. Optimization of in-network data reduction. In *DMSN Workshop*, 2004.

[18] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. Inst. Radio Eng.*, 40(9):1098–1101, 1952.

[19] http://db.csail.mit.edu/labdata/labdata.html.

[20] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD*, 2004.

[21] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[22] P. Karras and N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2005.

[23] Y. Kotidis. Snapshot queries: Towards data-centric sensor networks. In *ICDE*, 2005.

[24] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *OSDI*, 2002.

[25] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[26] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, June 1963.

[27] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, 1998.

[28] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, 2000.

[29] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. *ACM Trans. Inf. Syst.*, 16(3):256–294, 1998.

[30] F. Murtagh. The haar wavelet transform of a dendrogram. *Journal of Classification*, 24(1):3–32, 2007.

[31] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *SIGMOD*, 2001.

[32] Sensinet enables fda-compliant temperature monitoring and data collection. http://www.sensicast.com/uploadedFiles/CS-Pharma.10.06_casestudy.pdf.

[33] D. Sacharidis, A. Deligiannakis, and T. K. Sellis. Hierarchically compressed wavelet synopses. *VLDBJ*, 18:203–231, 2009.

[34] W. Scheufele and G. Moerkotte. On the complexity of generating optimal plans with cross products (extended abstract). In *ACM PODS*, 1997.

[35] http://sensorscope.epfl.ch/index.php/Environmental_Data.

[36] E. J. Stollnitz, T. D. Derose, and D. H. Salesin. *Wavelets for computer graphics: theory and applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[37] D. Tulone and S. Madden. An energy-efficient querying framework in sensor networks for detecting node similarities. In *ACM MSWiM symposium*, 2006.

[38] D. Tulone and S. Madden. Paq: time series forecasting for approximate query answering in sensor networks. In *EWSN*, 2006.

[39] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *ACM SIGMOD*, 1999.

[40] R. S. Wagner, R. G. Baraniuk, S. Du, D. B. Johnson, and A. Cohen. An architecture for distributed wavelet analysis and processing in sensor networks. In *IPSN*, 2006.

[41] J. Werb. Making sense of the sensor network value chain. http://www.sensicast.com/uploadedFiles/Resource_Center/Making_Sense_of_the_Sensor_Network_Value_Chain.pdf.

[42] Y. Yao and J. Gehrke. Query processing for sensor networks. In *CIDR*, 2003.

[43] S. Zhou, Y. Lin, J. Wang, J. Zhang, and J. Ouyang. Compressing spatial and temporal correlated data in wireless sensor networks based on ring topology. In *WAIM*, 2006.