Approximating Sensor Network Queries Using In-Network Summaries

Alexandra Meliou [#1], Carlos Guestrin [*2], Joseph M. Hellerstein [#3]

[#] *EECS Department, UC Berkeley*
[1] ameli@cs.berkeley.edu
[3] hellerstein@cs.berkeley.edu

[*] *SCS, Carnegie Mellon University*
[2] guestrin@cs.cmu.edu

ABSTRACT

In this work we present new in-network techniques for communication efficient approximate query processing in wireless sensornets. We use a model-based approach that constructs and maintains a spanning tree within the network, rooted at the basestation. The tree maintains compressed summary information for each link that is used to "stub out" traversal during query processing. Our work is based on a formal model of the in-network tree construction task framed as an optimization problem. We demonstrate hardness results for that problem, and develop efficient approximation algorithms for subtasks that are too expensive to compute exactly. We also propose efficient heuristics to accommodate a wider set of workloads, and empirically evaluate their performance and sensitivity to model changes.

Categories and Subject Descriptors: G2.2 [**Discrete Mathematics**]: Graph Theory-*trees*; F2.0 [**Analysis of Algorithms and Problem Complexity**]: General; E1 [**Data Structures**]-*distributed data structures, graphs and networks*.
General Terms: Algorithms
Keywords: Sensor Networks, Data Compression, Query Approximation

1. INTRODUCTION

Query processing has received significant attention in recent research on wireless sensor networks [17]. As is well known, communication is one of the most expensive operations in a sensornet [2], so various query processing techniques have been proposed to minimize it. Model-based data acquisition is a particularly promising approach to address this issue [8]. It uses historical information gathered from the network to predict rough query answers from a probabilistic model, and decides which data is worth gathering from the network at query time to augment that model sufficiently to meet a desired accuracy bound. Given a chosen set of data to gather, path planning algorithms [20] compute an efficient distributed strategy to retrieve the data needed to augment the model. This model-based approach relies upon information stored at a central location. Centralized models can make accurate predictions – assuming that the centrally maintained model is accurate – but suffer from unexpected events like node and link failures. Given the distributed nature of sensornets, a natural advance upon the early model-based work is to move this processing into the network.

In this model-based setting, the focus is on providing approximate answers to queries. Approximate queries are well suited to sensornet settings because perfect accuracy is both hard to achieve, and typically unnecessary. Physical sensors provide a spatially discrete and thus approximate view of the continuous physical phenomena they are used to monitor. Physical sensing also induces measurement errors, due to device imperfections, calibration problems, and physical stresses like dirt and heat. As a result, most users of sensornet applications understand that total accuracy of reported values is unnecessary. Approximate queries allow some uncertainty in the reported result, typically characterized by two parameters: a window w of accuracy for the answer and a confidence limit δ that is expected to be satisfied. An example of such a query is the following: "Return the temperature at each sensor node, within $\pm 1^{\circ}C$ with 95% confidence". Our focus will be on these "`SELECT *`" queries, that return a reading from each of the sensors in a field.

Our work adopts a model-based approach to approximation [8], but moves the implementation to an in-network setting, where the models can stay more easily up-to-date, preprocessing of routing paths at query runtime can be eliminated, and failures on routing paths are no longer crucial. We propose the use of a carefully designed in-network spanning tree to minimize the communication required to return a ro-

IPSN'09, April 15–18, 2009, San Francisco, California, USA.

bust estimate of the value reported by each sensor node. This tree is rooted at a base station, with Gaussian models stored at each node in the tree, one per child of the node. Queries are answered by traversing to a depth in the tree where the summaries provide sufficient information to answer a query within a specified window of accuracy.

We break this problem into individual tasks which we discuss in the corresponding sections. First, we provide a formal definition of the problem of optimal in-network summaries. Second, we present an efficient Gaussian-based compression scheme that is geared towards minimizing erroneous reporting of values, which can be optimized based on query workload. Third, we present query traversal algorithms that utilize the compression scheme to make routing decisions and give value estimates with limited communication. Our work is grounded in formal hardness results for the optimal in-network summary problem, along with approximation algorithms for a basic query processing scenario. Given this formal basis, we expand the set of scenarios we consider to a more practical setting, using a set of intuitive heuristics. We conclude with an experimental evaluation of the sensitivity of our approach to changes in the data and query workload.

1.1 Related Work

The idea of Semantic Routing Trees was proposed in [16] as an overlay index in the network, to allow for routing decisions to those leaves relevant to the query. Our in-network summaries go further, maintaining summaries of the data at different tree levels to allow for reduction in communication. The use of summaries for query processing has been examined in different settings. GHTs ([23]) propose storing and retrieving network information using Geometric Hash Tables, and Distributed Quadtrees ([7]) overlay quadtree structures over WSNs to satisfy distance sensitive spatial queries.

Cristescu et al ([5, 6]) study the relationship between data representation and data gathering, based on coding strategies. For a gathering task spanning all network locations, the goal is to minimize communication by optimizing the tree routing structure for a given coding model. The problem of jointly optimizing sensor placement and the transmission structure is further developed in [10].

Our in-network summaries aim on using models distributed in the network to make query processing more cost efficient. The same objective was tackled by centralized approaches: In [8], the BBQ system proposes a model-driven scheme to provide approximate answers to queries posed in a sensor network, satisfying some information guarantees. [20, 22] also focus on a centralized approach, where all the decisions and planning are performed at a basestation node, and heuristics are given to cope with unexpected network behavior. PRESTO ([15]) is an in-network model-driven scheme, which is however push based – deviations for the model are detected locally and in this case data is pushed to the root.

The TinyDB system [18], which is largely used for data collection in sensor networks, uses spanning trees for the data retrieval, but does not rely on any other in-network data to optimize queries. The role of in-network storage is discussed in [19] as a way to reduce energy consumption in its role in the trade-off between computation and communication. Maintaining data in the network is the focus of distributed storage ([9]). Several different coding schemes are developed for storing information in the network, but the goal in this case is to make the data resilient to failures and not to optimize query execution. A similar tactic is employed by the SPIN protocol [12, 14], which disseminates data in the network, so that a user posing a query at different locations can immediately get back results.

In terms of data gathering, directed diffusion ([13]) sets up gradients from data sources to the basestation, forming paths of information flow, which also perform aggregation. Rumor Routing ([4]), uses long lived agents that create and redirect paths to events they encounter.

2. OPTIMAL IN-NETWORK SUMMARIES

In this section we focus on the type of data summaries that we need to maintain to optimize queries in a sensornet setting and give a definition of optimality for that scheme. We then define the problem of data compression and treat it from the standpoint of the query workload.

Our workload consists of "`SELECT *`" style queries that request the approximate values of multiple sensors, without aggregation. The approximation bounds are defined by an accuracy window w and confidence limit δ specified by a query Q ($Q(w, \delta)$). The accuracy and confidence parameters specify the error tolerance that is acceptable in the answer. So a query with accuracy w, confidence δ and cardinality k, requires that on expectation δk of the reported values will fall within $\pm w$ of the actual values, i.e. the values that we would get if we sampled all locations. Depending on the values of w and δ, a query can range from being very loose to very strict in terms of accuracy. The smaller the window, and the higher the confidence, the stricter the query becomes, demanding more accurate results.

To answer queries in a sensornet setting using information residing in the network, we introduce "in-network summaries", or "spanning summary trees". An in-network summary is a spanning tree of the network, in which every node stores a model of the data in each of the subtrees it points to. As the subtrees become smaller in the lower levels of the hierarchy, the models become finer and more precise. A query using that hierarchy can explore the structure starting at the root and going only as deep as is necessary to provide answers of good quality. We want to optimize this tree structure with the objective of minimizing the communication cost of answering queries. The guarantee we want to achieve is that for n data nodes the query will produce on expectation δn answers that are within w distance of their actual value.

DEFINITION 1. *An* in-network summary *(or spanning sum-*

mary tree) over a network graph $G(V, E)$ is a spanning tree $T(V, E')$, $E' \subseteq E$, augmented with models M_v, $\forall v \in V$. M_v is stored in node p that is the parent of v, i.e., $(p, v) \in E$.

We are given a network graph $G(V, E)$ and a query workload $W = \{Q_i(w_i, \delta_i)\}$. We will use M_v to symbolize the model information kept for node v. Then the optimization problem of finding the optimal in-network summary can be defined as follows:

Given: Graph $G(V, E)$, query workload $W = \{Q_i(w_i, \delta_i)\}$
Find: Tree $T = G'(V, E')$ and models M_v, $\forall v \in V$, such that the average communication cost required to retrieve values to respond to $Q_i \in W$ is minimized

Our requirements for the models in the *in-network summaries* are that they are *compact*, *informative* and *accurate*. [1]

3. MODEL COMPRESSION

In this section we focus on one part of the in-network summary problem: the construction of models at each node. For this discussion, we assume temporarily that the structure of the tree T is given, and we want to pick the best model M_v, $\forall v \in T$. We will revisit the structure of T in Section 5.

Various models could be maintained in a spanning summary tree, but in keeping with prior work we assume a gaussian (normal) model. For the modeling of input *readings*, gaussian models are typically appropriate, since they successfully capture the nature of noisy measurements of physical phenomena ([3]). Therefore, for all leaves in T the model M_v will be a gaussian distribution based on observations of that node's measurements. Now, when a data summary needs to represent multiple sensors, a natural extension is to use gaussian mixtures, which can be of restricted size to comply with the storage limitations that sensor nodes have. A gaussian k-mixture refers to a mixture of k gaussians.

Assuming for simplicity that T has fixed fanout F, data resides at the leaves represented by the single gaussian distributions, and every internal node keeps F pairs of (childptr, gaussian k-mixture), then our "data structure" closely resembles a database index. Given a specified spanning tree T, we can imagine "bulk loading" the models M_v with a bottom-up construction, combining gaussian mixtures as we climb up the hierarchy. In this approach, mixtures high in the tree will be quite large. Since we need to keep the size restricted, we need to have a method for compressing the mixtures, otherwise called "collapsing".

In the problem of compression, our input is a mixture (set) of l gaussian distributions, and our output a k-size mixture, $k < l$. The parameter k is dictated by the amount of storage space assigned to the model on every node, and is not required to be the same on each one of them. We will first address the problem for the case of $k = 1$, assuming that the summary hierarchy keeps a single gaussian distribution as a model of all the sensors in each subtree. In Section 5.5 we revisit the compression model and explore generalizations to larger k.

During query execution, the only information that we have for a certain subtree is its collapsed distribution. In the case of a single-gaussian compression model ($k = 1$) the answers that this summary in the tree can provide is to report the mean μ of the summary as the answer for all nodes represented by the subtree. Intuitively, our goal is:

Given: A set of distributions $\mathcal{S} = \{N(\mu_i, \sigma_i^2)\}$
Find: Distribution $N(\mu, \sigma^2)$ that maximizes informativeness and accuracy over the original set $\mathcal{S}$.

To understand *informativeness and accuracy* in the above definition, some intuition is useful. A common approach in collapsing gaussian mixtures is to minimize some distance function (e.g. KL divergence) of the collapsed distribution from the original mixture. While intuitively this might seem to capture the quality of the compression, such an approach can actually lead to very bad decisions for our problem.

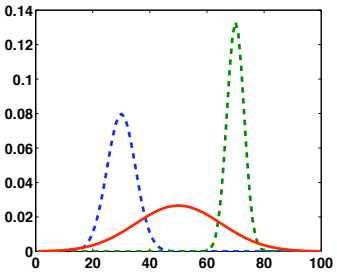

Figure 1: Two distributions (dashed lines) representing values of 2 sensor nodes, with no overlap. Collapsing using KL divergence produces a distribution (solid line) with significant mass in an interval that the original distributions contained almost none.

In the example of Figure 1 we want to collapse the 2 distributions depicted by dashed lines into one. With KL divergence as the optimization criterion, the result would be the solid line in Figure 1. The resulting distribution minimizes the distance from the original ones, but has the following pitfall: it falsely reports some significant mass on the interval $[45 - 60]$ where the original distributions contained almost none. In our setting this can in some cases lead to false query results for both of the original sensors involved, and thus loss of *accuracy*. A query with a large window may not suffer from this side effect, but one with a window small enough to fit in the problematic interval (e.g. width 10) could produce erroneous results.

The important observation here is that whether the answer would be wrong – and how wrong – depends on the specific query. This observation suggests that the collapsing be targeted to a specific query workload. This is the approach we follow below.

Alternatively, to avoid loss of accuracy, we could adjust the variance of the collapsed gaussian so that it will not contain any "fake" mass. But such an approach would result in an extremely wide and flat distribution. Such high uncertainty would be useless in query execution, as this summary could not produce answer estimates that would fall in a plausible query's accuracy window. In this case we have loss of *informativeness*.

[1] Refer to the extended version of this paper [21] for a more crisp definition.

3.1 Simple Collapsing

During compression we want to preserve as much information from the original distributions as possible. This means that the new distribution should contain as much "real mass" as possible, and this will happen if it is centered at the location that contains the most mass from the underlying distributions. This location however depends on the window in which we compute the mass. In Figure 1, if the window used is fairly large, then the location z that maximizes the total mass will be centered somewhere in between the two original distributions. On the other hand, if the window is small, then z would be centered at the narrowest of the original distributions. Therefore, compression of a given set of distributions will depend on the window assumed.

This also relates to query answers. In order to better understand the collapsing requirements, we need to look at how the collapsed distribution will be used to answer a given query. Assume a collapsed distribution $N(\mu, \sigma^2)$. $Q(w, \delta)$ is a query with error allowance inside a window w and confidence requirement δ. We consider that model distribution N can satisfy Q, if the mass of N in the interval $[\mu - w, \mu + w]$ is greater or equal to the confidence requirement δ, i.e. $M_{[\mu-w,\mu+w]} \geq \delta$. In that case the query reports value μ for both nodes. Note that the definition of query satisfaction is based on the belief that collapsed distribution N is an accurate representation of the underlying measurements. If N is a bad model, then any results based on it would simply be faulty. Based on our definition of query satisfaction, we need to construct N in such a way so that $M_{[\mu-w,\mu+w]} \geq \delta$ is true not only for N, but also for the uncompressed mixture.

As it is obvious from Figure 1, if we choose N as depicted by the solid line, then for small values of w the query may apparently pass the mass test, but the response will be wrong, as neither of the original nodes has value $\mu \pm w$ with δ confidence. However, if the window w is large enough, the distribution N may be sufficient to answer Q without problems. In order to make sure that that answer will be correct based on the query requirements we need to make sure that the mass of the collapsed distribution in the interval $[\mu - w, \mu + w]$ is the same as the total mass of the original distributions in the same interval. This observation leads directly to a simple approach for collapsing.

Our two requirements for collapsing are that it should not introduce "fake mass" (high accuracy), and it should retain as much "real mass" as possible (high informativeness). This makes sense for a specific choice of window, and it guarantees that queries of the same window will get accurate response. Formally, the problem we want to solve is as follows:

Given: One dimensional function f representing a probability distribution (in our case a gaussian mixture)

Find: z s.t. the mass of f in $[z - w, z + w]$ is maximized.

$$\max_z \int_{z-w}^{z+w} f(x)dx \qquad (1)$$

Once the optimal location for z is chosen, this becomes the mean of the collapsed distribution. The variance of the new distribution will be calculated based on the mass of the original distributions in the same interval: $\int_{\mu-w}^{\mu+w} N(\mu, \sigma^2)dx = \sum_i \int_{\mu-w}^{\mu+w} N_i(\mu_i, \sigma_i^2)dx$

The solution to the maximization of equation (1) cannot always be determined analytically. One approach is to numerically evaluate it through the use of sliding windows: given window w, "slide" the interval $[z_i - w, z_i + w]$ across the x-axis calculating the mass of the distribution for every z_i and pick the one with the maximum value. Another approach is to use gradient ascent with the means of the original distributions as starting points. Experiments on those approaches showed that the computationally efficient sliding windows technique provides accurate results for moderate discretization ([21]).

3.2 Tail-aware Collapsing

Even though simple collapsing does guarantee that the mass inside window w of its mean ($[\mu - w, \mu + w]$) is accurate (i.e. corresponds to real mass from the original distributions), there is no guarantee for parts of the interval $I \subset [\mu - w, \mu + w]$, or the tails of the distribution $[-\infty, \mu - w] \cup [\mu + w, \infty]$.

Tail-aware collapsing is a more conservative version of collapsing that disregards the mass in the tails of the distributions in the calculation of the location of maximum mass. This avoids inaccuracies that could be introduced through recursive collapsing of distributions, but leads to more conservative models ([21]).

4. QUERY TRAVERSAL

In the previous section we discussed optimal compression of a set of input distributions based on an expected query workload. Sensor nodes organized in a tree structure can create an in-network summary by recursively performing compression bottom up, from leaves to root. In this section, we focus on the problem of routing queries using an in-network summary, making routing decisions at each node based on the local model M_v. First we discuss what the optimal traversal would be for query $Q(w, \delta)$ on tree $T = G(V, E)$. A traversal is a connected component of G that contains the basestation. Since G is a tree, every connected component in it is also a tree. The optimal traversal is that of minimum total cost that *satisfies* Q on expectation (Def. 2).

DEFINITION 2 (QUERY SATISFACTION). *In a network of n nodes, a response $R = \{r_1 \dots r_n\}$ to a query $Q(w, \delta)$ is said to satisfy Q if on expectation the actual values of at least δn nodes fall in their respective interval $[r_i - w, r_i + w]$.*

Due to linearity of expectations, assuming that the underlying distributions are correct, then the query will be satisfied if $\sum_i \int_{r_i-w}^{r_i+w} f_i(x)dx \geq \delta n$. So the optimal traversal problem can be defined as follows:

Given: tree $T = G(V, E)$ and node models $M_v, \forall v \in V$

Find: $G'(V', E')$, $E' \subseteq E$, such that $\sum_e Mass(M_u, w) \geq \delta n$, where $e = (u, v)$ with $u \in V'$ and $v \in V \backslash V'$

In the above problem statement, $Mass(M_u, w)$ is the maximum mass that can be contained inside window w from model M_u. If the mixture model is compressed to a single gaussian, $Mass(M_u, w) = \int_{\mu-w}^{\mu+w} f(x)dx$, where f is the normal probability density function.

4.1 DP Traversal

We will solve the optimal traversal problem using a dynamic programming algorithm. Every node will keep a DP table (in our case it's just a vector) which will hold information on forwarding decisions based on assigned budget. For example, if v is the root of subtree T_v and C_{T_v} is the cost of traversing the entire subtree T_v, then v will keep a vector of length C_{T_v} where every entry will be the maximum mass that can be collected by assigning the corresponding budget to that subtree. For a tree of fanout F, children $u_1, \ldots, u_F$ of node v and a budget assignment $B = \{b_1, \ldots, b_F\}$ among the F children, the DP function for computing the entry $J_v(c)$ will be:

$$J_v(c) = \max_B \sum_{i=1, b_i>0}^{F} J_{u_i}(b_i) + \sum_{i=1, b_i=0}^{F} Mass(M_v, w)|T_{u_i}| \quad (2)$$

where $|T_{u_i}|$ corresponds to the number of nodes represented by the subtree of node u_i. The second summation in (2) gives a mass estimate for the unvisited children ($b_i = 0$) based on the model of node v. Also, $\sum_i b_i = c - \sum_{i\ s.t.\ b_i>0} w_{(v,u_i)}$, where $w_{(v,u_i)}$ the weight of the edge (v, u_i), and the sum subtracted in this formula adjusts the available budget by the cost of reaching those children of v with non-zero budget assignments. The weight of each (v, u_i) edge can be simply defined as the number of hops needed to reach that child. Along with the DP vector, the choice of best budget assignment for each cell should also be kept. At the leaves of the tree, the DP vector is a single element, $J(0) = Mass(M_v, w)$.

Algorithm 1 outlines this dynamic program for the case of a binary tree ($F = 1$), and unit edge weights. The code can be easily extended to the more general case of fanout F or even unrestricted fanout.

Algorithm 1 DPConstruct(v,w,B)

```
1: if B < 0 then
2:    return 0
3: if B == 0 then
4:    J(B) = Mass(M_v, w)
5: else
6:    for k = 0 ... B do
7:       lMass(k) = DPConstruct(u_1, w, k − 1)
8:       rMass(k) = DPConstruct(u_1, w, B − k − 1)
9:    J(B) = max_k(lMass(k) + rMass(k))
10:   leftBudget(B) =argmax_k(lMass(k) + rMass(k)) − 1
11:   rightBudget(B) = B − leftBudget(B) − 1
```

With the DP in place, routing decisions can easily be made depending on the δ parameter of the query. Given δ we can compute the desirable mass as δn, where n is the total number of nodes in the network. Using the DP table at the root, we can find the smallest budget that achieves that mass, say b_i, and the values of $leftBudget(b_i)$ and $rightBudget(b_i)$ will give the budget assignments for the left and right child respectively. The traversal descends until the budget is exhausted. The DP vector for each node is equal to the number of nodes in that node's subtree, so if n are all the nodes in the tree, the space needed would be $O(n)$. Constructing the DP vector for each node has complexity $O(n^2)$, so the whole algorithm has complexity $O(n^3)$.

The described DP approach can compute the optimal traversal solution for a query of window w, on a tree model compressed using the same window. One problem is that the DP tables can become large at nodes high in the hierarchy, which could violate our limited storage principle.

As an alternative, we proceed to propose a simple greedy traversal algorithm that makes decisions locally at every node, without the requirement of keeping extra information, like the DP tables.

4.2 Greedy Algorithm

Our greedy descent algorithm is quite straightforward. The query is initiated at the root of the tree, and every node decides whether to descend or not based on the satisfiability of the query by the local model:

$$\int_{\mu-w}^{\mu+w} f(x)dx \geq \delta \quad (3)$$

If the model at the current node satisfies (3), then no descent is necessary. Otherwise the query is forwarded to the node's children. In this simple version of the algorithm, if a decision is made to forward, then all of the children will receive the query. More elaborate schemes can give different priorities to children and perform selective forwarding. This however would require extra communication, as decisions to forward might depend on traversal results on other subtrees, and cannot be made only locally.

Algorithm 2 GreedyDescend(node$_i$,w,δ)

```
1: Compute I = ∫_{μ−w}^{μ+w} f(x)dx
2: if I ≥ δ then
3:    return μ_i
4: else
5:    GreedyDescend(children_i,w,δ)
```

Algorithm 2 is more conservative than the DP approach, as it applies the satisfiability definition on every subtree and not just the global tree. The algorithm will terminate the recursion at a set of nodes each of which satisfy the query according to Definition 2 in their local subtree. If k_i is the number of nodes represented by every subtree i where the recursion terminated, then on expectation $\delta \sum_i k_i = \delta n$ have accurate within w values, which in turn means that globally the query is satisfied. So every solution of the greedy algorithm is guaranteed to satisfy query Q, but the satisfiability definition only

requires it to hold globally and not for every subtree, making Algorithm 2 conservative.

We evaluate the performance of the greedy algorithm comparing it with the DP solution for different sizes of the window w, and against different values of the confidence parameter δ depicted on the x-axis. Both algorithms in this case are executed on the same binary tree with data gathered from the Intel Berkeley Lab deployment [1]. To construct the initial leaf distributions, data from one hour is analyzed, and nodes are grouped together in the tree based on spatial proximity. The results are given in Figures 2 and 3. The greedy algorithm demonstrates a more conservative behavior in terms of cost, which also results in a smaller number of reported errors, but stays a competitive alternative, without requiring maintaining state in the nodes.

In this experiment, the choice of range of window sizes for the query workload ([0.5,2]) relates to the variance of the underlying data. Queries with windows smaller than 0.5 would always sample all, or most, locations, as the queries are now too strict even for the leaf level models. Also, experiments with very big windows provide no useful information, as the queries become so loose relative to the data that even the rough root level model is enough to satisfy them.

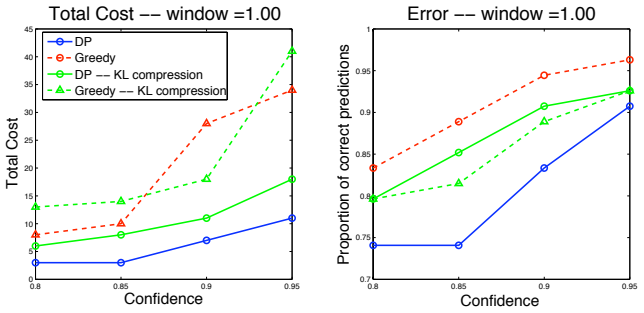

Figure 4: Comparison of our compression method with KL divergence based compression, using DP and greedy traversal

In Figure 4 we demonstrate the benefits of our compression scheme, based on mass maximization, with KL divergence based compression. Depicted are the results for a specific query window size, but the rest behave similarly. The comparison is done using both the DP and greedy traversal algorithms, and in both cases our compression results in plans of lower cost. An important note is that the temperature data used for our experiments actually gives KL divergence some advantage, and yet it still loses. With data points whose underlying distributions differ a lot, the problems of the KL based compression would become more exaggerated.

5. THE TREE CONSTRUCTION PROBLEM

In the previous sections we discussed methods for performing compression and query traversal. However, the algorithms were applied over a predefined tree, and even though compression is optimally done given a specific design workload (i.e. a set window), the way the nodes are grouped affects the quality of compression. Grouping of dissimilar nodes into the same subtree will inevitably lead to bad compression, and therefore bad performance. In this section we will discuss the problem of finding the optimal tree for a specific query workload.

We focus on the case of graphs with unit edge weights. From a practical perspective, a communication link between two nodes is considered to exist if the nodes have a packet loss rate bounded by a threshold.

The metric to minimize is communication cost during query execution. Assume T_{OPT} is an optimal tree that produces the minimum possible cost when traversed by query $Q(w, \delta)$. A traversal of T_{OPT} using the greedy algorithm (Alg. 2) will stop at nodes on various levels that all satisfy inequality (3). Define the cut C as the set of nodes at which Alg. 2 stops. Note that there is no path from root to leaf in tree T_{OPT} that does not "hit" the cut C. The structure of the tree below the cut, i.e. all nodes that have an ancestor in C, is irrelevant to the cost of query Q, as the query traversal will never descend that far.

Also, in the case of trees with constant fanout, the structure above the cut is irrelevant as well as shown in Theorem 1:

THEOREM 1. [2] *In a tree with fixed fanout F, the cost to traverse from the root to cut, i.e. the cost to reach all nodes in the cut from the root, is $\frac{F}{F-1}(|C| - 1)$, where $|C|$ is the size of the cut.*

5.1 Optimal Tree Problem

Theorem 1 shows that if C is the cut that Alg. 2 picks for query $Q(w, \delta)$, then the structure of the hierarchy above and below the cut are irrelevant to the cost of answering the query, and therefore only the size of the cut determines query cost. A node in the cut acts as a representative for the whole subtree, so the cut is a set of connected components each of which contains a representative node with model M_v that satisfies Q.

The Optimal Tree Problem with respect to query $Q(w, \delta)$, is the problem of finding a tree that satisfies Q using Alg. 2 with minimum communication cost, and is formally defined as follows:

DEFINITION 3 (OPTIMAL TREE PROBLEM).
Given: *Graph $G(V, E)$ with the cost function $c : E \rightarrow \mathcal{R}_+$ and query $Q(w, \delta)$*
Find: *Set of components $S = \{C_1 \dots C_k\}$ such that*

- $\forall i$ *C_i is connected in G.*
- $\forall i \ \frac{1}{|C_i|} \sum_{j \in C_i} Mass_j(C_i) \geq \delta$.
- $\cup C_i = V$ *and* $C_i \cap C_j = \emptyset$ *for* $i \neq j$
- *With the objective to minimize the cost of the minimum cost subtree T of G that contains at least one vertex from each component C_i.*

In the above definition $Mass_j(C_i)$ represents the total mass that node j contributes to component C_i.

[2]For proofs refer to the extended version of this paper [21]

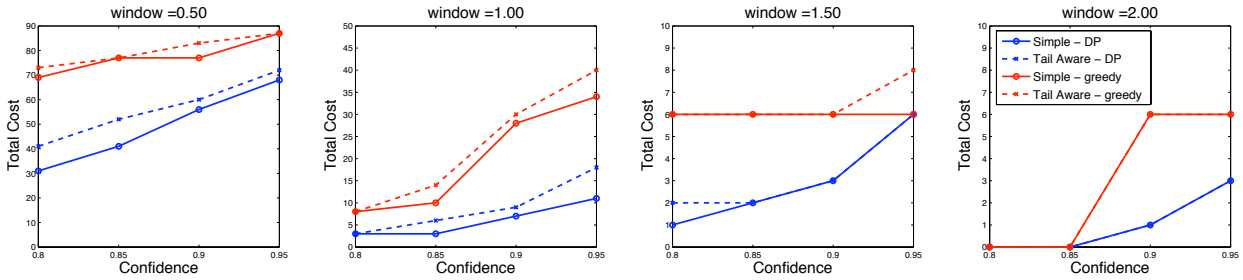

Figure 2: Evaluation of cost of Greedy against optimal cost found by the DP algorithm. The "Simple" and "Tail-aware" schemes refer to the type of compression deployed (Section 3)

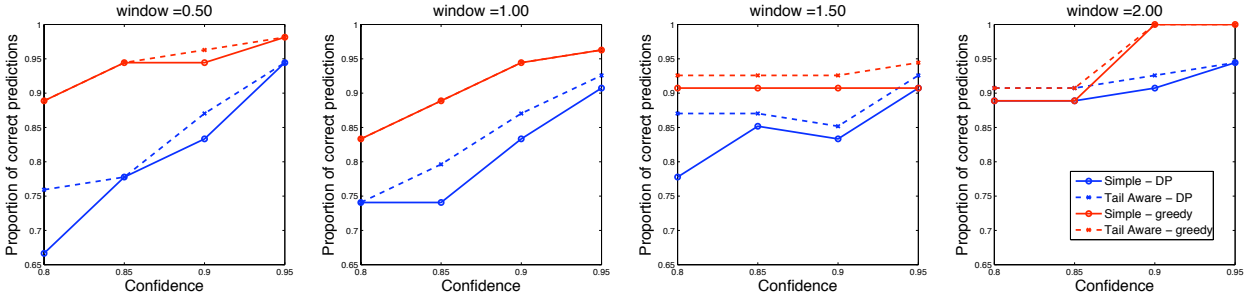

Figure 3: Comparison of the proportion of correct responses for the greedy and the optimal cost traversal chosen by the DP algorithm. The "Simple" and "Tail-aware" schemes refer to the type of compression deployed (Section 3)

The objective of the Optimal Tree Problem (4th bullet) is the Group Steiner Tree Problem: given a graph $G(V, E)$, cost function $c : E \rightarrow \mathcal{R}_+$ and sets of vertices $g_1, g_2, \ldots, g_k \subset V$, find the minimum cost subtree T of G, that contains at least one vertex from each set g_i.

Since the Group Steiner Tree Problem is NP-hard, the Optimal Tree Problem is also hard. The GST problem has a polylogarithmic approximation [11], so we will attempt to address the selection of components C_i as a separate problem.

5.2 Optimal Clustering

From Definition 3, the division of nodes into the components C_i can be viewed as a clustering problem. Each cluster corresponds to a subtree rooted at a node chosen by the query cut. Each cluster is of limited diameter (Def. 3, 2nd bullet) and, according to Theorem 1, we want to find the minimum size cut for the specific query, so equivalently the optimal clustering should minimize the number of clusters.

This subproblem is also hard, but we provide a greedy algorithm that approximates the optimal solution by a logarithmic factor. The pseudocode is given in Alg. 3.

PROPOSITION 1. *Algorithm 3 provides a factor* $\log(n)$ *approximation to the optimal clustering, which minimizes the number of clusters.*

Proposition 1 is easily derived when one notices that Algorithm 3 is equivalent to greedy Set Cover. In practice the algorithm behaves a lot better than this guarantee, with results always close to the best solution. The graphs from those experiments were omitted due to space constraints.

It is important to note that Algorithm 3 may violate one of the conditions of Definition 3, that each cluster has to be connected. Greedy Clustering greedily adds nodes to each

Algorithm 3 GreedyClustering(V, w, δ)

```
Clusters = ∅
Pick discretization D = {z_1, ..., z_k}
repeat
  for z ∈ D do
    S_z = ∅
    for v_i ∈ V do
      Mass_z(v_i) = ∫_{z-w}^{z+w} f_i(x)dx //f_i the model of node v_i
    while Σ_{v_i ∈ S_z} Mass_z(v_i) ≥ δ|S_z| do
      v* = argmax_i Mass_z(v_i)
      remove v* from Mass_z
      if Σ_{v_i ∈ S_z ∪ {v*}} Mass_z(v_i) ≥ δ|S_z| then
        S_z = S_z ∪ {v*}
      else
        break;
  S*_z = argmax_i |S_{z_i}|
  Clusters = Clusters ∪ S*_z
  V = V \ S*_z
until V = ∅
```

fixed cluster center from the discretization. For a fixed center, the weight of a vertex is constant and independent of which other vertices join the same cluster. Without the connectivity requirement, the greedy addition of vertices in decreasing weight will indeed create the cluster with the largest cardinality for that interval. However, if connectivity is enforced, the problem becomes NP-hard as we proceed to show.

Limited Diameter Max Connected Subgraph:

Given: Graph $G(V, E)$, vertex weight W_u for vertex u, maximum diameter D. [3]

Find: $G'(V', E')$ s.t: G' is connected, $\frac{1}{|V'|}\sum_{u \in V'} W_u \leq D$, with the objective to maximize $|V'|$

[3]With vertex weight W_u being the mass of u's distribution outside the window interval, then D would be $1 - \delta$

THEOREM 2. *The Maximum Connected Subgraph of Limited Diameter Problem (MCSLD) is NP-hard.*

The connectivity requirement ensures that the participating nodes can form a tree. To enforce connectivity we can either: (a) augment the clusters chosen by Algorithm 3 with extra communication nodes to force connectedness, or (b) change the greedy algorithm so it only augments the clusters with nodes accessible by those already in the cluster. With option (b) the algorithm will no longer have the logarithmic guarantee. Option (a) may be more appealing in most cases, as the extra cost for intra-cluster communication is only setup cost for the cluster formation and determination of the common model, and does not inflict extra cost during query time. Figure 5 presents a comparison of communication cost between clusters constructed by the different approaches. The numbers for Greedy with Intra-Cluster cost encode the model setup cost; during query execution communication cost is equivalent to Greedy and therefore beats the Connected Greedy approach.

5.3 Distributed Clustering

Algorithm 3 provides a centralized approach in approximating the minimum number of clusters. In this section we will give a distributed clustering algorithm, and experimentally compare it with the centralized one.

Algorithm 4 Distributed Expanded Neighborhood

```
1: Basestation broadcasts clustering msg
2: Each node picks a random wait time.
3: if wait time passes without cluster requests then
4:    node initiates clustering
5: repeat
6:    node randomly selects v from neighbor table
7:    if d ≤ D_max then
8:       node sends cluster request to v
9:       Augment neighbors table with neighbors of v
10:   else
11:      Remove v from neighbors
12: until no more neighbors
```

A distributed approach avoids the overhead of communicating all the data to a centralized location, and is more suitable for performing selective reclustering in case some part of the hierarchy needs to be updated. Algorithm 4 basically initiates at a single node level where a cluster of size 1 is created. The node looks up neighbors on its neighborhood table and attempts to augment its cluster size by inviting them to join the cluster. Once a new neighbor joins, the current neighborhood is augmented by the newcomer's neighbors. The cluster stops growing when no more neighbors can be added without exceeding the maximum allowed cluster diameter D_{max}. Note that the algorithm requires some bookkeeping and messages to communicate neighborhood tables. A simpler version of distributed greedy clustering uses a random walk to augment the cluster, with a node making a pick only from its own neighbors and not the complete neighborhood defined by the cluster. The cluster stops growing when the random walk cannot proceed without violating the cluster diameter.

In Figure 6 we compare the 2 centralized approaches, greedy and connected greedy, and the 2 distributed approaches, expanded neighborhood and random walk. The simulation experiments were performed in Matlab, with real data from the Intel Berkeley Lab deployment. The algorithms are compared on the cardinality of the clusterings that they create. Out of all the approaches, greedy is the only one that does not enforce the connectivity requirement, which does not ensure that it will give the best result as it is not optimal, but has a logarithmic approximation guarantee.

From these experiments we observe that the results of distributed clustering are very comparable to the centralized ones, especially to connected greedy. On average the distributed expanded neighborhood and random walk heuristics performed within a factor of 1.4 and 2.8 respectively of the centralized greedy algorithm which has a guaranteed logarithmic factor approximation of the optimal solution.

5.4 Building Trees for Varied Workload

In Section 5 we discussed the problem of constructing the optimal tree for a specific query. We connected it with a clustering problem, we showed hardness results and gave approximation algorithms and heuristics. In this section we will extend our approach to the setting of a more varied query workload. We will assume a baseline confidence δ, and query workload that includes various different windows $\{w_1, \ldots, w_k\}$.

Following the intuition that models high in the hierarchy present a coarse view of the data, while moving into deeper levels provides more detail, it seems natural to address varied workload by recursively clustering in decreasing order of window size. Clustering will start with the largest window size which represents the less strict query in the workload. The clusters produced are further divided into smaller clusters using the next largest window in the query workload, and the process continues in that fashion until the smallest window size. This heuristic, sketched in Algorithm 5 produces a tree in which every level corresponds to a different window size, from larger in the higher parts of the hierarchy, to smaller in the deeper levels.

Algorithm 5 TreeConstruction(G,wRange)

```
1: sort(wRange)
2: G(k+1) = G;
3: for i = k downto 1 do
4:    w = wRange(i)
5:    G(i) = cluster(G(i+1),w)
6:    Connect G(i) with G(i + 1)
```

We will experimentally evaluate the performance of our heuristic by comparing its communication cost for queries of the different window sizes, versus a tree that was geared only towards the specific window at hand. We design a single tree T for window sizes 0.5, 1, 1.5 and 2, and confidence 0.9, as well as 4 other trees each one geared towards a single one

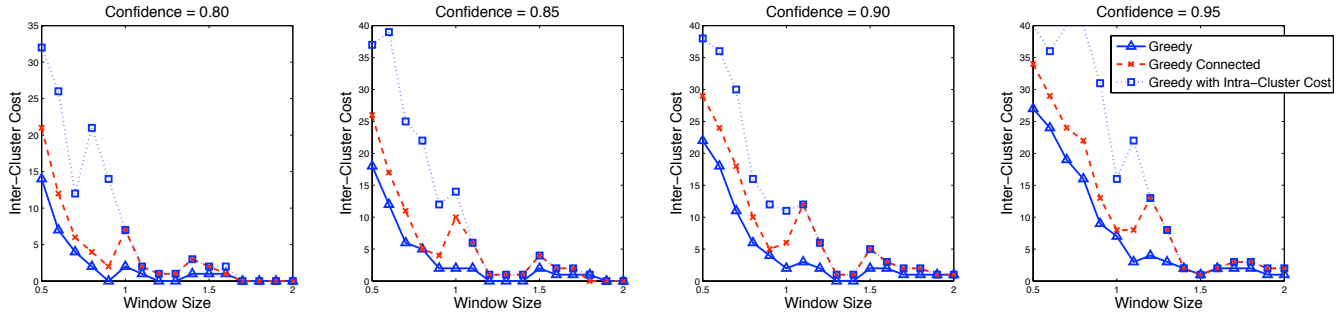

Figure 5: Comparing the different clustering approaches, based on the communication cost for varied parameters of window size and confidence for the query workload.

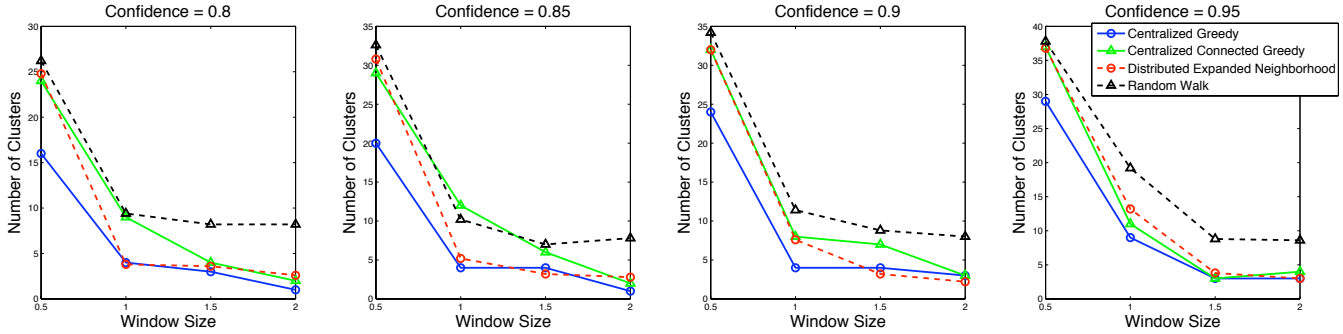

Figure 6: Comparison of the distributed and centralized clustering algorithms.

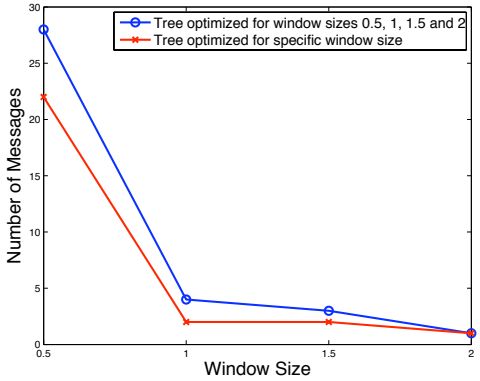

Figure 7: Comparing the performance of a tree designed over workload W vs a tree clustered over a single window

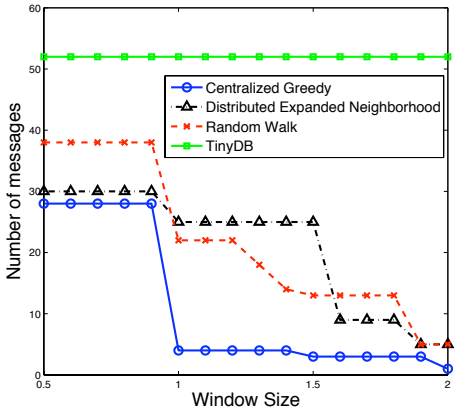

Figure 8: Query experiments on trees constructed by different clustering algorithms

of the previous window values. The choice of window sizes is again dictated by the variance of the underlying data which we are modeling, as explained in Section 4.2. Compression with window sizes that are too small or too big, relative to the underlying data variance, would produce models of too high variance that would not be useful in query answering. The communication cost of the query workload over T is evaluated against the corresponding cost of the optimal tree for each window size.

Figure 7 shows that Algorithm 5 approximates well the best single clustering solution. Note that the tree construction is not specific to the type of clustering used, and any of the clustering algorithms that we proposed –distributed or centralized– can be used for that step.

Figure 8 compares the performance of hierarchies constructed using the tree construction heuristic (Alg. 5) for a design workload of window sizes 0.5, 1, 1.5 and 2, and confidence of 0.9, with different methods for the clustering step: centralized connected greedy (Alg. 3), distributed expanded neighborhood (Alg. 4) and random walk. The hierarchies are evaluated on the communication cost of a query workload of a fine range of window sizes. The distributed algorithms are somewhat outperformed by the centralized approach, but they still beat traditional data gathering approaches like TinyDB ([18]) which gathers data along a spanning tree. The TinyDB cost is constant and independent of the query parameters.

5.5 Enriched Models

Our analysis up to this point focused on Single Gaussian Model (SGM) compression schemes, which have minimal requirements of storage space from the nodes. In this section

we will examine more complex models and evaluate their performance against SGMs. Note that the tradeoff is not only between informativeness and storage space, but also communication cost, as larger models will be more expensive to transmit and update. The size of each model is represented by the parameter k (for SGMs $k = 1$), and the amount of space that each model uses is proportional to k.

We compare 3 different types of models against SGMs:

k-mixture: A k-size mixture is maintained instead of a size 1 mixture. The compression of a l-size mixture to size k is done by clustering the l distributions of the original mixture into k sets using a modified k-means algorithm. Then each set is compressed to a SGM as described in Section 3.1.

Virtual nodes: As in k-mixtures, an l-size mixture is divided into k sets and a SGM is computed for each one. The difference is that in the previous approach, a single k-mixture represents all of the nodes in the subtree, whereas here each separate SGM represents only that portion of the nodes used to construct it. This is equivalent to "splitting" each node into k virtual nodes and using simple SGMs. Note that this approach requires some extra bookkeeping space to record the groupings of sensor nodes into the virtual nodes.

SGMs on multiple windows: The extra space is used to maintain additional SGMs for different window sizes. Depending on the query window the appropriate model is used at every node.

The generalized models described are evaluated against simple SGM compression on a tree built for a workload of window sizes $[0.5, 1, 1.5, 2]$ and confidence of 0.9. The models are then tested on a query workload of a finer range (Figure 9). The results shown are for enriched models of size $k = 4$. Experiments with larger k were also performed, with no change in the results.

An initially surprising observation is that enriched models demonstrate minimal to no performance gains. Specifically, k-size mixtures are not any better than a size 1 mixture, and the same goes for virtual nodes. Even though this may seem counterintuitive, it is justified by the criterion used during tree construction. Nodes are divided into clusters ensuring that the mass in the interval $[c - w, c + w]$ for each cluster, where c the center of the cluster, is enough to satisfy the confidence that the tree was designed for. Also, the design of SGMs preserves this mass in the resulting compressed model. Therefore, a cluster designed on that criterion can be sufficiently represented by a SGM, making more elaborate models unnecessary for queries of equal or less confidence than the tree was designed for. We observe some minimal gains on queries of higher confidence – we can now receive some benefit from the additional information – but still the gains are not significant. These results are actually evidence of the quality of our tree construction method. Queries have requirements for normal error bounds and thus a normal distribution is the appropriate model when the underlying nodes are clustered based on its properties.

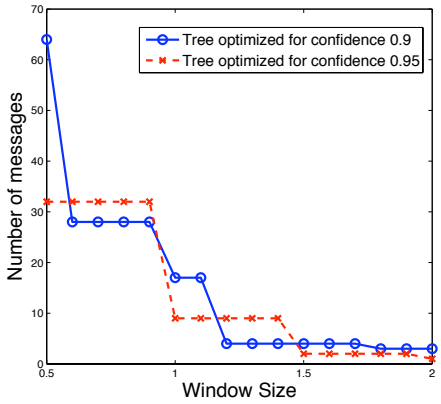

Figure 10: Comparison of hierarchies built on different confidence. The query workload is of confidence 0.95.

The 3rd design, SGMs for multiple windows, shows some performance gains, but note that these are not due to modeling improvements, i.e. better representation of the underlying data, but better tuning to the design workload. A SGM hierarchy attempts to minimize the average communication cost for the design workload. The benefit in multi-window SGMs arises from keeping models of several window sizes on higher levels, thus giving the opportunity to queries to terminate higher in the tree than they otherwise would.

Our analysis demonstrates that a SGM is sufficient for our data representation, making in-network summaries a very simple and effective approach for modeling sensornet data.

6. SENSITIVITY ANALYSIS

In this section we evaluate the sensitivity of our tree construction algorithms to different parameters using simulation experiments. As with all the previous experiments, these are also performed over real temperature data from the Intel Berkeley Lab deployment.

First we examine the effects of the confidence parameter in tree construction. Two trees built on different confidence limits, 0.9 and 0.95, are evaluated over the same query workload of confidence 0.95. The results, shown in Figure 10, demonstrate that the choice of confidence for tree construction does not have a big impact on performance.

In our second set of experiments we want to explore the effects of the choice of design workload on communication performance during query time. Query workload W is evaluated over a tree T built with W as the design workload, and another one T' built with $W' \subset W$ (Fig. 11). Interestingly enough T is not always the winner. This is because T is forced to have more levels than T', and for some window sizes (e.g. 0.5), it makes queries traverse more levels. As a result, including the full workload in the tree design is not necessarily the best choice. It would be interesting to explore how to optimize the window range for the tree design, and incorporating that range as a parameter in the optimization may require revisiting our tree construction heuristic.

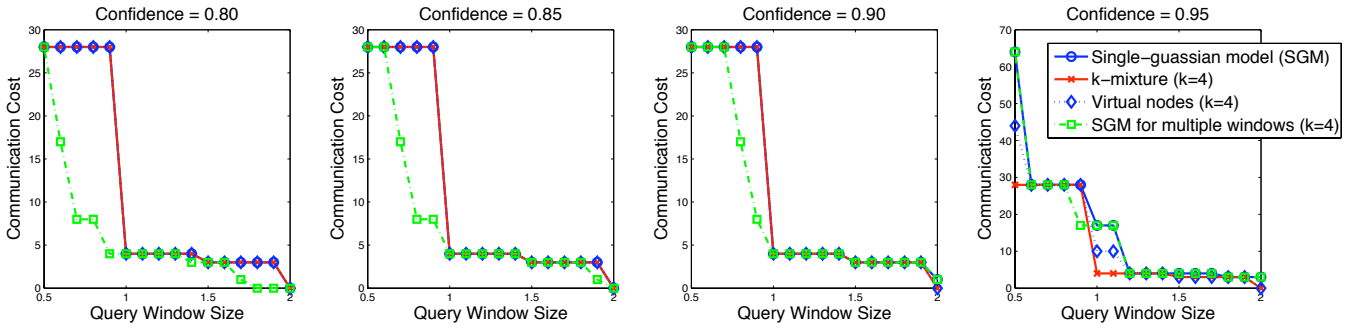

Figure 9: Evaluation of SGMs and enriched models

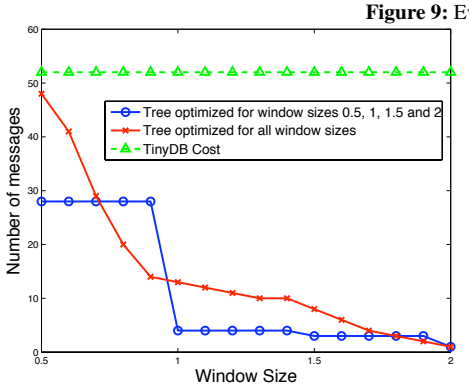

Figure 11: Comparing tree construction with a few vs a broader range of windows

In our final performance experiments, we evaluate the performance of in-network summaries over time. Data is taken for 48 hours, starting around midday of the first day. A summary hierarchy is built based on data from the first hour, and a workload W of window sizes $[0.5, 1, 1.5, 2]$ and confidence 0.9. When models at the leaves change due to temperature variations throughout the day, those get propagated up the tree either at query time, or with a background process. Models then get updated at various levels, but the structure of the tree remains the same throughout the experiment.

The hierarchy is tested using workload W every hour over a period of 48 hours, and compared with the performance of a tree that is completely rebuilt every hour (full restructuring). The results are given in Fig. 12.

We observe that even with full reconstruction we get bad performance when variance of the underlying data is high, as happens around timesteps 20 and 40. In the case of model updates without restructuring of the tree, even though for large windows the results are reasonable, for small windows they are very disappointing. Note however that performance deteriorates at different times for different windows. From the experimental results, the higher levels of the hierarchy corresponding to larger windows remain consistent until about 40 hours later, mid-levels (window 1) seem to become unusable after about 6 hours, and the lowest levels (smallest window) seem to not be at all reusable.

This behavior can be addressed with *escalated* restructuring. Different levels of the tree get restructured with different frequencies: clusters corresponding to small windows will get reclustered more frequently, whereas those of larger window size much less often. In Figure 13 escalated restructuring is performed every hour, 6 hours and 24 hours, for clusters of windows 0.5, 1, and 1.5 respectively. The choice of restructuring frequencies at this point is ad hoc and based on observations of the behavior in Fig. 12, but the purpose of this experiment is to demonstrate that escalated restructuring provides big performance benefits. An interesting part of future work would be to discuss how to automatically derive the appropriate frequencies or determine on the fly when reclustering is necessary.

In-network summaries with escalated restructuring have performance comparable to the best result. After time 42 however, the model deteriorates, because coincidentally the size 1 clusters get updated on a moment of high variance of the underlying data, and thus the resulting clusters are of low quality. A simple fix would be to detect high variance in the data, and avoid restructuring during those times.

Using escalated reclustering, in-network summaries are able to follow changes of the data that happen at a relatively slow rate. This makes our scheme suitable for various monitoring applications that deal with slow changing phenomena, like environment observation and forecasting systems, various types of habitat monitoring applications, landslide detections, water quality monitoring. It is not suitable for emergency response situations that involve sudden changes in the underlying data, like fire alert systems. In order to deal with cases that require some type of outlier detection, in future work, in-network summaries could potentially be augmented with a push scheme that locally detects sudden data changes and reports them to higher levels.

7. CONCLUSIONS

In this paper we introduced in-network summaries to improve the efficiency of approximate query processing in sensornets. Summaries can be used to make routing decisions and provide answers to queries without paying the communication cost to access the whole network, and without requiring centralized planning and model maintenance. We formally defined the problem of optimal summary construction, and broke it into several different components that we addressed. We presented efficient compression schemes for the summaries which are optimized based on query workload, and provided traversal algorithms that utilize the summary

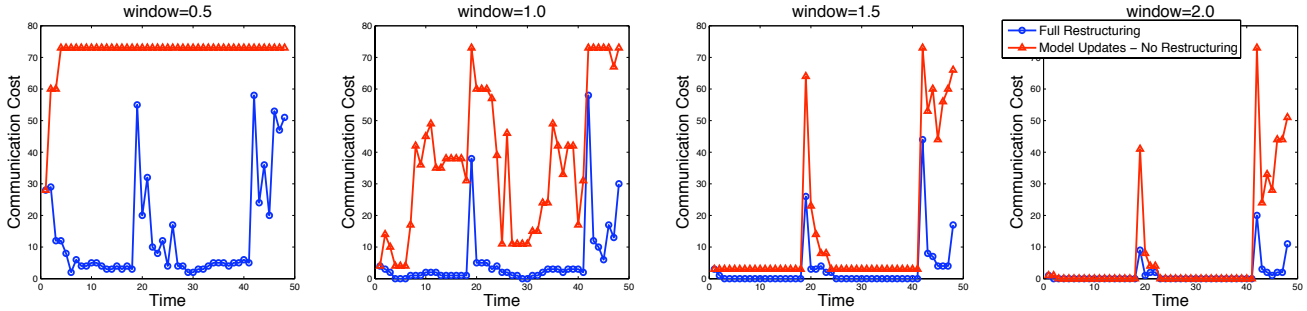

Figure 12: Time progression of in-network summaries with model updates.

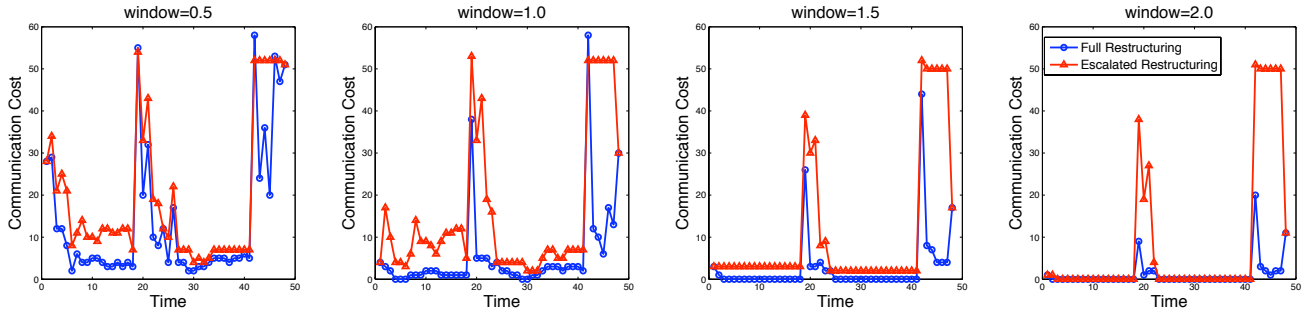

Figure 13: Time progression of in-network summaries with model updates and escalated restructuring.

structures to produce query results. We related the problem of tree optimization to a clustering problem which proved to be NP-hard, and gave a centralized approximation algorithm and distributed heuristics. We experimentally evaluated our algorithms using data from a real world deployment and performed comparisons across multiple parameters. Finally we tested the sensitivity of our scheme to variations of the design decisions and the data, and we showed that it is quite robust to changes.

Acknowledgements: This work was supported by NSF Grants IIS-0803333, NeTS-NBD CNS-0721591 and IIS-0713661.

8. REFERENCES

[1] https://mirage.berkeley.intel-research.net/.
[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 2002.
[3] Analog Devices. Data sheet: Small, low power 3-axis ±3g imems accelerometer adxl 330.
[4] D. Braginsky and D. Estrin. Rumor routing algorithm for sensor networks. In *WSNA*. ACM Press, 2002.
[5] R. Cristescu, B. Beferull-Lozano, and M. Vetterli. On network correlated data gathering, 2004.
[6] R. Cristescu, B. Beferull-Lozano, M. Vetterli, D. Ganesan, and J. Acimovic. On the interaction of data representation and routing in sensor networks. In *ICASSP*, 2005.
[7] M. Demirbas and X. Lu. Distributed quad-tree for spatial querying in wireless sensor networks. *ICC*, June 2007.
[8] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, 2004.
[9] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *IPSN*, page 15, Piscataway, NJ, USA, 2005. IEEE Press.
[10] D. Ganesan, R. Cristescu, and B. Beferull-Lozano. Power-efficient sensor placement and transmission structure for data gathering under distortion constraints. *ACM Trans. Sen. Netw.*, 2(2):155–181, 2006.
[11] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000.
[12] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks, 1999.
[13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom*, 2000.
[14] J. Kulik, W. R. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 2002.
[15] M. Li, D. Ganesan, and P. Shenoy. Presto: feedback-driven data management in sensor networks. In *NSDI*, 2006.
[16] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD*, pages 491–502, New York, NY, USA, 2003. ACM.
[17] S. Madden and J. Gehrke. Query processing in sensor networks. *Pervasive Computing*, 3(1), 2004.
[18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 2005.
[19] G. Mathur, P. Desnoyers, D. Ganesan, and P. Shenoy. Ultra-low power data storage for sensor networks. In *IPSN*, 2006.
[20] A. Meliou, D. Chu, C. Guestrin, J. Hellerstein, and W. Hong. Data gathering tours in sensor networks. In *IPSN*, 2006.
[21] A. Meliou, C. Guestrin, and J. M. Hellerstein. Approximating sensor network queries with in-network summaries. Technical Report UCB/EECS-2008-137, EECS Department, UC Berkeley, Oct 2008.
[22] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein. Nonmyopic informative path planning in spatio-temporal models. In *AAAI*, 2007.
[23] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: a geographic hash table for data-centric storage. In *WSNA*, 2002.