Approximate Data Collection in Sensor Networks using Probabilistic Models*

David Chu — UC Berkeley
Amol Deshpande — University of Maryland
Joseph M. Hellerstein — UC Berkeley, Intel Research Berkeley
Wei Hong † — Arched Rock Corp.

Abstract

Wireless sensor networks are proving to be useful in a variety of settings. A core challenge in these networks is to minimize energy consumption. Prior database research has proposed to achieve this by pushing data-reducing operators like aggregation and selection down into the network. This approach has proven unpopular with early adopters of sensor network technology, who typically want to extract complete "dumps" of the sensor readings, i.e., to run "`SELECT *`*" queries. Unfortunately, because these queries do no data reduction, they consume significant energy in current sensornet query processors.*

In this paper we attack the "`SELECT *`*" problem for sensor networks. We propose a robust approximate technique called* Ken *that uses* replicated dynamic probabilistic models *to minimize communication from sensor nodes to the network's PC base station. In addition to data collection, we show that Ken is well suited to anomaly- and event-detection applications.*

A key challenge in this work is to intelligently exploit spatial correlations across *sensor nodes without imposing undue sensor-to-sensor communication burdens to maintain the models. Using traces from two real-world sensor network deployments, we demonstrate that relatively simple models can provide significant communication (and hence energy) savings without undue sacrifice in result quality or frequency. Choosing optimally among even our simple models is NP-hard, but our experiments show that a greedy heuristic performs nearly as well as an exhaustive algorithm.*

1 Introduction

Sensor networks open up new opportunities to observe and interact with the physical world around us. They enable us to gather data that was until now difficult, expensive, or even impossible to collect. The Sonoma Redwoods sensor network project is a representative deployment [28]. Consisting of 72 Mica2dot motes [4] placed throughout two giant redwood trees in a grove in Sonoma County, CA, the network enabled biologists at UC Berkeley to access detailed readings of temperature, humidity, and Photo-synthetically Active Radiation (PAR) from many different positions under the large redwood canopies. This data was never before available to the plant biology community. With this information, new dynamic tree respiration and growth models are being formulated, with practical implications for both environmental management and timber cultivation.

While sensor networks can provide revolutionary new data sources for a variety of applications, it is challenging to extract this data from a sensor network because of the limited battery resources on each sensor device. In practice, sensor network deployments make sense only if they can run unattended for many months or even years. Scientists do not want to climb redwood trees to replace batteries very often, and even conveniently placed sensors like home smoke detectors need to have a long lifetime to achieve practical viability. Hence any sensor network technology has to be stingy in its energy consumption. Among the various tasks performed by a wireless sensor node, radio transmissions are by far the most expensive in terms of energy consumption. On a typical sensor node, the Telos mote, message transmission and receipt expend an order of magnitude more energy than CPU computations over an equivalent length of time [21]. Poor energy consumption can be dramatic in practice: for example, a software bug in the original Sonoma Redwoods deployment caused a third of the nodes to constantly keep their radios active; they exhausted their batteries in only a few days.

One way to significantly reduce communication cost in sensor networks is to perform in-network aggregation (e.g., `AVG` and `MIN`) [19, 11] or data reduction via wavelets or distributed regression [22, 12]. However, these techniques do not provide the fine data granularity desired by many sensor network users. For example, the biologists in the Sonoma Redwoods project would like to receive as much detailed data from the sensor network as possible, so that they can try various physical models and test various hypotheses over the data. Even though the Sonoma Redwoods deployment was based on TinyDB [20], a sensor network query processor capable of in-network aggregation and filtering, the biologists were only willing to issue the "`SELECT * FREQ` *f*" query, where the modifier `FREQ` *f* indicates the query should repeat at the given frequency. The notion that the network is condensing away detail is unattractive to them in

*This work was supported by an NSF fellowship, NSF Grant No. CNS-0509220 and a gift from Microsoft Corporation.

†Work done while at Intel Research Berkeley.

their exploratory research, so they favor transmitting more detailed data, even at the expense of a lowered frequency. Based on our first-hand experiences with early sensor network deployments, this line of reasoning appears to be quite common.

In this paper we address the problem of providing timely, efficient support for *approximate, bounded-loss* data collection in sensor networks – i.e., for "`SELECT * FREQ f WITHIN` $\pm\epsilon$" queries. For many sensornet applications, the granularity provided by such approximate data collection is more than sufficient, especially considering that the sensing devices themselves are rarely 100% accurate in measuring the underlying physical properties. We exploit the fact that physical environments frequently exhibit predictable stable states and strong attribute correlations that can assist us in inferring the state of a sensor from its past and its surroundings. For example, outdoor temperatures typically follow consistent diurnal and seasonal patterns, and at any moment in time, are unlikely to vary greatly within a local region.

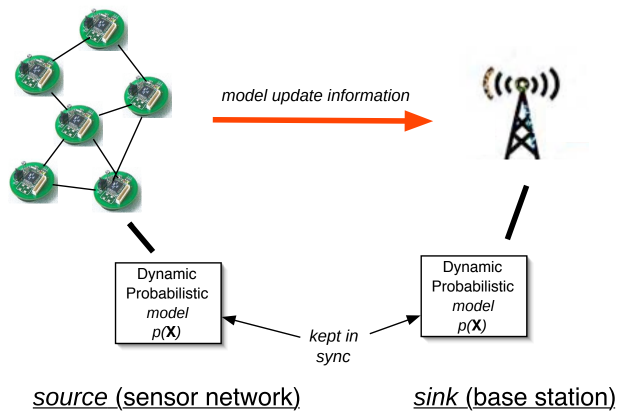
1.1 Proposed Approach

Our approach, *Ken*[1], is based on a form of compression using *replicated dynamic probabilistic models*. The basic idea is to maintain a pair of dynamic probabilistic models over the sensor network attributes, with one copy distributed in the sensor network and the other at a PC base station. At every time instance (*i.e.*, with a frequency f), the base station simply computes the expected values of the sensornet attributes according to the model[2] and uses it as the answer to the "`SELECT * FREQ f`" query. This requires no communication. The sensor nodes always possess the ground truth, and whenever they sense anomalous data – i.e., data that was not predicted by the model within the required error bound – they proactively route the data back toward the base station. As data is routed toward the base station, spatial correlations among the reported data are used to further lower communication. Using these techniques, all user-visible readings are guaranteed to be within a fixed error bound from the measured readings, even though very few readings are communicated to the base station. Figure 1 summarizes this idea.

An attractive feature of the Ken architecture is that it naturally accommodates applications that are based on event reporting or anomaly detection; these include fire-alert-and-response [16] and vehicle tracking [26]. In these scenarios, the sample rate is typically quite high, but the communication rate should remain quite low under most circumstances. The model reflects the expected "normal" state of the environment being monitored; anomalies result in reports being pushed to the base station for urgent handling by infrastructure logic. In addition to naturally supporting these applications, Ken enhances them with additional functionality: the ability to support interactive query results with well-bounded approximate answers. In essence, approximate data collection and event detection become isomorphic.

[1]ken **1:** range of perception, understanding, or knowledge
2: name of a tiny model

[2]In reality, this computation for time t can only be done at time $t + \Delta$ because of the *latency* of communicating the data from the sensor devices to the base station.


model update information
Dynamic Probabilistic model p(**X**)
kept in sync
Dynamic Probabilistic model p(**X**)
source (sensor network)
sink (base station)
Figure 1: Ken Overview: A pair of dynamic probabilistic models, one at the base station, and one distributed in the sensor network, are maintained over **X**, the sensornet attributes. Information is communicated from the sensornet to the base station only if the predictions are not within bounds.

A key technical challenge in our work is the degree to which we can exploit spatial correlations among distributed sensor nodes. A rich model of the spatial correlations can significantly improve compression of the data communicated to the base station. On the other hand, sophisticated models may require significant communication among the sensor nodes themselves for coordination and in-network model maintenance. We explore tradeoffs in this regard, considering in particular how the distributed model maintenance is mapped onto the communication topology of a sensor network. We focus on a natural class of distributed models based on spatial partitions of the network. Even within this class of models, choosing the optimum partitioning is NP-hard. We present both a dynamic programming algorithm and a greedy heuristic for partitioning, and demonstrate experimentally that the heuristic performs effectively on real-world network traces.

1.2 Overview of the Paper

In the remainder of the paper we present the design and evaluation of Ken, with the following contributions:

- We present the Ken architecture, which uses replicated dynamic probabilistic models to provide cheap `SELECT *` queries with bounded-loss guarantees (Section 3). This subsumes prior work as discussed in Section 2 [23, 15, 14] by taking advantage of predictions across both time and space.
- We investigate temporal and spatial correlations and their interplay with the network topology, and formulate the model selection problem for one natural family of models, Disjoint Cliques. After showing this model selection problem to be NP-hard, we present both exhaustive and heuristic approaches (Section 4).
- We evaluate Ken's performance on several real-world sensor network datasets and find substantial support for its use relative to previous schemes (Section 5).

- Finally, we discuss a number of extensions to the Ken approach including richer models, probabilistic error bounds, and mechanisms for handling network dynamics (Section 6).

2 Related Work

There is a large body of work on data collection from sensor networks. Directed diffusion [13] is a general purpose data collection mechanism that uses a data-centric approach to choose how to disseminate queries and gather data. Cougar and TinyDB [33, 20] provide declarative interfaces to acquiring data from sensor networks. None of this work, however, considers the problem of efficient, approximate data collection. More recently, the BBQ system [7] proposes using probabilistic modeling techniques to optimize data acquisition for sensor network queries. BBQ's approach is "pull-based", using correlation models to satisfy queries with a minimum of data acquisition. By contrast, Ken is "push-based", acquiring data at a steady rate and proactively reporting anomalies to the base station. A key weakness of BBQ's pull-based approach is that it is not well-suited for anomaly- or event-detection, whereas Ken is designed to capture any such variations. These two techniques are geared toward different application domains, and are largely complementary. We are currently exploring ways to combine them in one unified framework.

There has been much work in database literature on remote updates of continuously changing data values that is closely related to the problem we study here. Approximate data replication [23] is a query-driven replica maintenance scheme where users specify custom precision requirements along with their queries. These requirements are then used to either pro-actively "pull" just enough data from the remote sources, or to install filters at the remote sources so that they "push" the minimal amount of data to satisfy the precision requirements. Similarly, Jain et al [14] propose reducing the amount of data communicated in distributed data streams enviroment by using Kalman Filters. We will revisit these two techniques in the next section. Several works recast approximate caching under a narrow range of models. Lazaridis and Mehrotra [15] employ piecewise constant approximation schemes that provide data compression and prediction. Piecewise constant approximation, like approximate caching, offers a narrow range of predictive capabilities. More recently, Cormode et al [10] have proposed a similar approach of using replicated predictive models to solve the problem of maintaining accurate quantile summaries over distributed data sources.

Location management in *moving objects* databases is another closely related area of research [31]. Updating the server every time a moving object changes its location is prohibitively expensive, and in most cases, not very useful. Instead [31], among others, suggest keeping track of trajectories of moving objects, and only sending updates to the server if the moving object changes the trajectory, or doesn't behave according to the trajectory. In essence, this work (along with the work mentioned above) utilizes the *temporal* correlations in the data. In Ken, we present an approach that can utilize both the temporal and the *spatial* correlations (that are typically stronger in sensor networks) in the data; we expect our techniques to be useful in application domains such as location management as well.

Modeling patterns and correlations in the data, and using them to reduce data transmission rates has been a central theme in the data compression literature [25]. The best-known (though not necessarily the most practical) lossless text compression algorithms of this sort (e.g., *Prediction by Partial Matching (PPM)* [2]) combine *arithmetic encoding* [30] with Markov Models to achieve very high compression rates. Though the approach we take is abstractly similar, the distributed nature of data acquisition in the environments we consider raises interesting issues not considered in this previous work.

Distributed source coding addresses the problem of losslessly compressing *correlated* sources that are not co-located and cannot communicate with each other to minimize their joint description costs. Slepian and Wolf [27, 32], in a celebrated result, show that it is still possible to compress the data at a combined rate equal to the joint entropy of the correlated sources. This result is however non-constructive, and constructive techniques are known for a few, very specific distributions. Moreover, these techniques typically require *precise* and *perfect* knowledge of the correlations between the attributes, and will return wrong answers (without warning) if this condition is not satisfied. In our work, the bounded-loss approximation guarantees are maintained irrespective of whether the correlation model is known accurately.

Data summarization schemes, used heavily for selectivity estimation and approximate query answering in database systems [24, 29, 5] are unable to provide guarantees on individual samples (with the exception of recent work on probabilistic wavelets [9]). Also, they encounter nontrivial overheads when mapped onto a distributed setting. For example, although wavelets [29] can be used for compactly summarizing data distributions, it is unclear how they might efficiently map onto distributed nodes [12].

3 Ken Architecture

We will begin by formalizing the problem that Ken attempts to solve, and discuss how *replicated dynamic probabilistic models* can be used to solve this problem. We then provide a detailed description of the Ken architecture, and formulate the model selection problem.

Problem Definition: We are given (a) a sensor network (also called *source*) that continuously monitors a set of distributed attributes $\mathbf{X}$, and generates a data value $\mathbf{x}^t$ at every time instance t[3], and (b) a PC base station (*sink*) that requires an ϵ-loss *approximation*, $\hat{\mathbf{X}}^t$, of the *true* data values at all times, *i.e.*, $\forall i, t, |x_i^t - \hat{X}_i^t| < \epsilon$. Design a data collection protocol that optimally achieves this by utilizing known temporal and spatial correlations in the sensornet attributes.

[3]The time instances at which data is acquired depends on the application-specified frequency of data collection.

source	data producer; sensor network
sink	data consumer; base station (also node 0)
X_i	An attribute being sensed by the network. In many cases, we will use X_i to also denote the sensor node that is sensing that attribute.
X_i^t	Random variable denoting the attribute X_i at time t
$\hat{X}_i^t$	Expected value of the variable X_i^t according to the current state of model
x_i^t	Observed value of X_i^t
$\mathbf{X}$	Set of all attributes
$\hat{\mathbf{X}}^\mathbf{t}, \mathbf{x}^\mathbf{t}$	Vector equivalents of the above terms, e.g., $\mathbf{x}^\mathbf{t} = \langle x_1^t, \ldots, x_n^t \rangle$.
ϵ	The accuracy bound. At all times, Ken satisfies: $\forall t \lvert x_i^t - \hat{X}_i^t \rvert \leq \epsilon$
$p(X_1^t, \ldots, X_n^t)$	pdf over the attributes at time t
$\mathcal{N}$	Set of all sensor nodes
$\mathbf{o}^t$	observations communicated to the *sink* (and incorporated in the model through the conditioning process) at time t
$comm : \mathcal{N} \times \mathcal{N} \rightarrow R$	pair-wise communication costs

Table 1: Terminology and notation

3.1 Replicated Dynamic Probabilistic Models

The basic premise behind Ken is simple: both *source* and *sink* maintain a dynamic probabilistic model of how data evolves, and these models are always kept in sync (Figure 1). The *sink* uses the data value(s) predicted by the model as the approximation to the true data, and the *source*, who knows the predicted value by virtue of running a copy of the model, makes sure that the predicted data values satisfy the required bounded-loss approximation guarantees, by communicating some information to the consumer as required.

Example 3.1 *One very simple prediction model assumes that the data value remains constant over time:*

$$\hat{\mathbf{X}}^{t+1} = \hat{\mathbf{X}}^t \;\; (i.e., \hat{X}_i^{t+1} = \hat{X}_i^t, \, \forall i)$$

In that case, the predicted value according to the model is same as the predicted value at the last time instant. The source*, by virtue of "running" a copy of the model, knows the value that the* sink *uses, and sends an update to the* sink *if the accuracy bound is not satisfied. Since this model is naturally distributed, each sensor node can decide and if required, send such an update independently of the other sensor nodes.*

Example 3.2 *Because of the strong temporal correlations present in sensor data, a better model would be a linear prediction model:*

$$\hat{X_i}^{t+1} = \alpha_i \hat{X_i}^t + \beta_i$$

where α_i and β_i are constants, and $\hat{X_i}^t$ denotes the approximation computed at time t (Table 1 summarizes the notation we use in this paper). In that case, the source *at time t checks whether $\hat{X_i}^{t+1}$ computed in this manner is sufficiently close to x_i^{t+1} it observes, and sends an update to the* sink *if it is not. The best-known lossless* text compression *algorithms essentially use generalizations of this basic idea [1]. This is also similar to the approach taken by Jain et al [14] in applying Kalman Filters to distributed streams.*

The second model, though an improvement over the first model, only utilizes the temporal correlations, and ignores the spatial correlations across sensors that tend to be very strong in many sensornet deployments. To unify treatment of both kinds of correlations, we let the the *prediction model* used by Ken be a *dynamic probabilistic model* that can be used to compute a *probability density function (pdf)*, $p(X_1^t, X_2^t, \ldots, X_n^t)$, over the possible assignments of values to the variables $X_i, i = 1, \ldots, n$ at time t. For simplicity, we restrict our presentation to *Markovian* models, where given the values of all attributes at time t, the values of attributes at time $t+1$ are independent of those for any time earlier than t. This assumption leads to a very simple model for representing such a dynamic system consisting of:

- A probability distribution function (pdf) for the initial state of the system, $p(X_1^{t=0}, \ldots, X_n^{t=0})$, and
- A *transition model*, $p(X_1^{t+1}, \ldots, X_n^{t+1} \mid X_1^t, \ldots, X_n^t)$, that can be used to compute the pdf for time $t+1$ from the pdf at time t[4].

As noted before, the *sink* uses the expected values of the variables according to the pdf computed at time t as the approximation to the true variable values. However, depending on the approximation errors and the specified accuracy bounds, the *source* may need to communicate some information to the *sink* that is incorporated in the model so that the accuracy bounds are met. For the Markovian dynamic models, this information takes the form of observed variables values, $X_i^t = x_i^t$, at time t, and the process by which this information is incorporated in the model is by *conditioning*. If we denote by $\mathbf{o}^t$, the observations reported to the *sink* at time t, the pdf at time $t+1$ can be computed given *all* the observations communicated to the *sink* till time t:

$$p(X_1^{t+1}, \ldots, X_n^{t+1} \mid \mathbf{o}^{1\ldots t}) = \int p(X_1^{t+1}, \ldots, X_n^{t+1} \mid X_1^t, \ldots, X_n^t) \, p(X_1^t, \ldots, X_n^t \mid \mathbf{o}^{1\ldots t}) dX_1^t \ldots dX_n^t.$$

Example 3.3 *Figure 2 shows an example of this process for a system with two variables, X_1, and X_2, where the data evolution is modeled using a 2-dimensional linear Gaussian. Figure 2 (i) denotes the model at time t, $p(X_1^t, X_2^t)$, with expected values $\hat{X}_1^t$, and $\hat{X}_1^t$ respectively. When transitioning to the next time instance, $t+1$, both the* source *and the* sink *use the transition model $p(X_1^{t+1}, X_2^{t+1} | X_1^t, X_2^t)$ to obtain a model $p(X_1^{t+1}, X_2^{t+1})$ over the two variables at*

[4]We currently assume that the parameters of the model are computed using historical data; we plan to address the issue of adapting these parameters over time in future work.

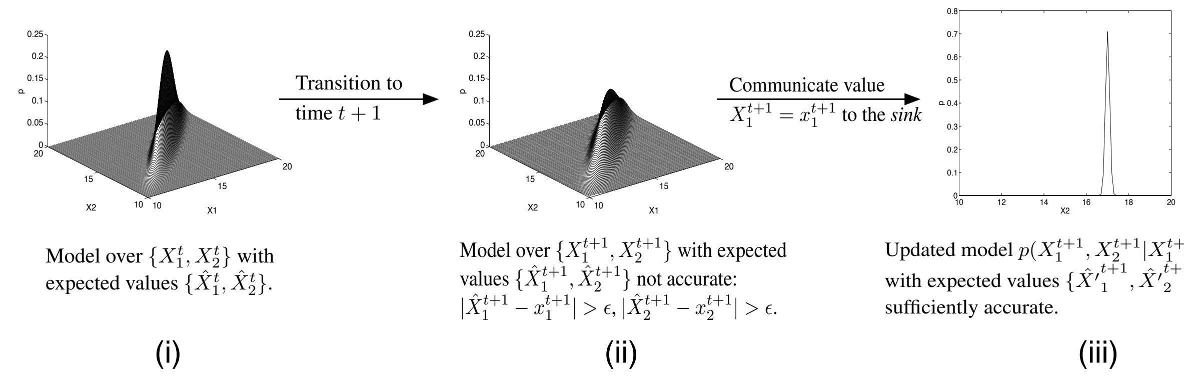


Figure 2: Ken in action

time $t+1$, with expected values $\hat{X}_1^{t+1}, \hat{X}_2^{t+1}$ (Figure 2(ii)). Now, the source *checks whether these expected values (that will be used as the approximation to the true values by the* sink*) are sufficiently accurate. If yes, no data needs to be transmitted to the* sink*. On the other hand, if the predictions are not accurate (as shown in the example), the* source *communicates a subset of the values (in this example, the observed value of X_1^{t+1}) to the* sink*, and both the* source *and* sink *update the model using this information to obtain $p(X_1^{t+1}, X_2^{t+1} | X_1^{t+1} = x)$. The expected values of the variables will now satisfy the required accuracy guarantees.*

Note that even though both variables violated the accuracy guarantees, because of the strong spatial correlations between them, the value of only one of them was needed to be communicated to the sink.

3.2 Ken Normal Operation

In Ken, the *source* performs the following steps (Figure 1) at time t:

1. Using the transition model, compute the probability distribution function $p = p(X_1^{t+1}, \ldots, X_n^{t+1})$ over attributes $X_i^{t+1}, i = 1, \ldots, n$. As discussed above, this pdf depends on all observations that have been *communicated to the* sink *so far*; other values are ignored since they did not change the model at the *sink*.
2. Compute the expected values of the attributes according to the pdf:
$$\hat{X}_i^{t+1} = \int X_i^{t+1} p(X_1^{t+1}, \ldots, X_n^{t+1}) dX_1^{t+1} \ldots dX_n^{t+1}$$
3. If the expected values are sufficiently accurate, *i.e.*, $|\hat{X}_i^{t+1} - x_i^{t+1}| < \epsilon, \forall i = 1, \ldots, n$, then *stop*. There is no need to send anything to the *sink*.
4. Otherwise:
(a) Find the smallest subset of attributes such that conditioning on it makes the predictions accurate. In other words, find the smallest $Y = \{X_{i_1}, \ldots, X_{i_k}\}$ such that the expected values according to the pdf:
$$p_Y = p(X_1^{t+1}, \ldots, X_n^{t+1} | X_{i_1}^{t+1} = x_{i_1}^{t+1} \ldots X_{i_k}^{t+1} = x_{i_k}^{t+1})$$
satisfy the accuracy guarantees. Note that the set of all attributes ($Y = \{X_1, \ldots, X_n\}$) always satisfies this condition.
(b) Send the values of attributes in X to the *sink*.

Most of the work during the normal operation of the Ken system is done by the *source*. The steps followed by the *sink* are as follows:

1. Use the transition model to compute $p = p(X_1^{t+1}, \ldots, X_n^{t+1})$ (identical to Step 1 above).
2. If the *sink* received from the *source* values of attributes in $X = \{X_{i_1}, \ldots, X_{i_k}\}$, then condition p using these values as described in Step 4 (a) above.
3. Compute the expected values of the attributes X_i^{t+1}, and use them as the approximation to the true values.

3.3 Ken: Choosing the Prediction Model

The model used for prediction in Ken is clearly a central piece of the system. Regardless of the model, the correctness of data collection is never compromised, *i.e.*, the accuracy guarantees at the *sink* are always maintained. However, the effectiveness of Ken in reducing the data transmission rates is highly dependent on the model chosen. This would suggest choosing a sophisticated and complex model that captures all the correlations and the patterns in the data. Unfortunately, because of the distributed nature in which data is generated in a sensor network, this is not always wise, and in fact, complex models can sometimes result in higher overall communication costs than relatively unsophisticated models such as the one used in Example 3.1 above. This is because in order to perform the steps outlined in Section 3.2, nodes may need to communicate with each other to bring correlated data together in one place, and this communication can be very significant.

The total communication cost incurred during the normal operation of Ken consists of two components:

- **intra-source:** cost incurred in the process of collecting the data generated at each time step to check if the predictions are accurate (*source* operation, Step 3).
- **source-sink:** cost incurred while sending a set of values to the *sink* (*source* operation, Step 4(b)).

Consider using a multi-dimensional model over all attributes of the sensor network that incorporates all spatial and temporal correlations (Figure 3(i)). As shown in previous work [7], such models work extremely well in sensor networks and can be used for probabilistic querying with great benefits. But if we were to use such a model in Ken , the *cost of checking whether the prediction is accurate* (intra-source cost) becomes prohibitively high. Such a model will require collecting values of all attributes at *one location* at each time step, and the cost of doing so will most likely dwarf any savings in source-sink communication that might result. On the other extreme is the approach of using a set of one-variable models, one for each attribute. In that case, we can distribute the task of checking whether predictions are accurate to the sensor nodes that generate those attributes, and hence, the intra-source cost will be 0. But, such a model will not be able to reap the benefits of spatial correlations in the data, which tend to be much stronger than temporal correlations in many cases.

This leads us to pose one of the key challenges in performing approximate data collection with accuracy guarantees:

Given a sensor network that consists of a set of nodes $\mathcal{N}$, a set of attributes $\mathbf{X}$ observed by these nodes, and a communication topology over these nodes[5]*, find a dynamic probabilistic model $\mathcal{M}$, and a mapping from $f : \mathbf{X} \rightarrow \mathcal{N}$ specifying where attribute predictions should be checked, such that the total expected communication cost is minimized, where that cost is the sum of:*

1. **intra-source :** *The total communication cost incurred in sending the value of attribute X_i to $f(X_i)$ at every time step.*
2. **source-sink :** *The total communication cost incurred while sending a set of values to the* sink *as required.*

Note that our optimization goal does not include the CPU cost of doing the inference itself, *i.e.*, the cost of checking whether predictions are accurate, and computing the minimal subset of values to send to the *sink*. This is because the communication cost is typically much higher than the computational cost of doing inference. We note that it would be fairly easy to extend the formulation above, and the algorithms we propose later, to include this cost as well.

A large variety of models can be used in Ken for the prediction purposes.

Example 3.4 Disjoint-Cliques Models: *A natural choice of $\mathcal{M}$ to reduce the intra-source cost, but also utilize spatial correlations between attributes, is to partition the sensor attributes in multiple localized clusters, and use a multi-dimensional model for each of these clusters. We will term these clusters of attributes* cliques[6]*, and the sensor node at which the inference is done the* clique root*. Note that the clique root is not required to be a part of the clique. Figure 3 (ii) shows an example of this, where the 6 attributes $X_1, \ldots, X_6$ are partitioned in two cliques $\{X_1, X_2, X_3\}$ and $\{X_4, X_5, X_6\}$ with clique roots X_2 and X_4 respectively. We will revisit this class of models in detail in Section 4.*

[5]We assume a static known network topology in the body of the paper; we revisit this issue in Section 6.

[6]We use the terminology from the *graphical models* literature; the clique attributes don't, in general, form a clique in the network topology graph.

Example 3.5 Average Model: *Figure 4 shows another possible class of models which could potentially work very well if the average of all the variables, $\bar{X} = \frac{\Sigma_{i=1}^{n} X_i}{n}$ can predict the individual values very well. In this case, $f : \mathbf{X} \rightarrow \mathcal{N}$ maps every attribute to the node that produces it, whereas $\mathcal{M}$ consists of n models, $\mathcal{M}_i$, each over two variables $X_i, \bar{X}$. Though this model might seem to incur a very high cost in computing $\bar{X}$ and communicating it to all the nodes in the sensor network, as shown in Figure 4, the computation of $\bar{X}$ can be done efficiently using* in-network aggregation*, requiring only $O(n)$ messages in the process. Disseminating $\bar{X}$ to the sensor nodes also takes at most $O(n)$ messages. Hypothetically, if the source-sink cost is zero (i.e., $\bar{X}$ always predicts X_i within the approximation bounds), this model can reduce the total communication cost by a factor of $O(n)$ over the naive approach of communicating all values generated by the sensor network to the base station.*

The latter model illustrates the large range of possible models that can be used in Ken making the problem of choosing the *best* model is extremely hard. In general no class of models will work well for all kinds of sensor networks. An interesting goal for future work is to build a system that allows different classes of prediction models to be used flexibly and dynamically; we will revisit the issue in Section 6. However, many of the sensor networks typically exhibit correlations that are inversely proportional to the distance between sensors. For example, a sensor network that monitors *temperature* or *humidity* exhibits this pattern. Fortunately for us, the Disjoint-Cliques models naturally exploit precisely this correlation pattern, and hence, we will focus on this class of models in the rest of the paper (though we also evaluate performance of the **average** model in our experimental study).

4 Disjoint-Cliques Models

Disjoint-Cliques models naturally localize and distribute the in-network computation required by Ken, and may be suitable for use in Ken in many different sensornet environments. In this section, we will address the problem of finding the optimal Disjoint-Cliques model. Not surprisingly the problem is NP-hard, and we will present both exhaustive and heuristic algorithms for finding good Disjoint-Cliques solutions. We compare the performance of these algorithms in Section 5.

4.1 Problem Analysis

We will denote a Disjoint-Cliques model as $\mathcal{M} = \mathcal{M}_1(C_1), \ldots, \mathcal{M}_k(C_k)$, where k is number of partitions, and $C_i \subseteq X, i \in \{1, k\}$ is a partitioning of X. In other words we will be running k separate multi-dimensional models, one for each clique C_i. Furthermore, we define the *data reduction factor* for C_i, m_i, to be the expected number of communicated values for C_i. We will address the issue of how to compute m_i in Section 4.4 as the computation depends on the exact nature of the individual models.

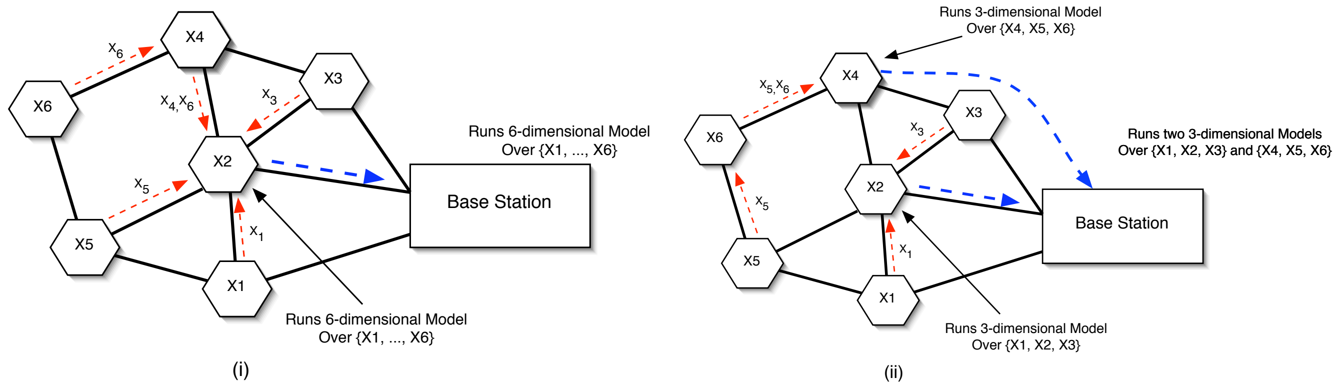

X4
X3
X6
X2
X5
X1
Base Station
Runs 6-dimensional Model Over {X1, ..., X6}
Runs 6-dimensional Model Over {X1, ..., X6}
(i)
Runs 3-dimensional Model Over {X4, X5, X6}
Runs two 3-dimensional Models Over {X1, X2, X3} and {X4, X5, X6}
Runs 3-dimensional Model Over {X1, X2, X3}
(ii)

Figure 3: (i) Using a single 6-dimensional model incurs very high data collection cost; (ii) An example Disjoint-Cliques model with cliques $\{X_1, X_2, X_3\}$, and $\{X_4, X_5, X_6\}$

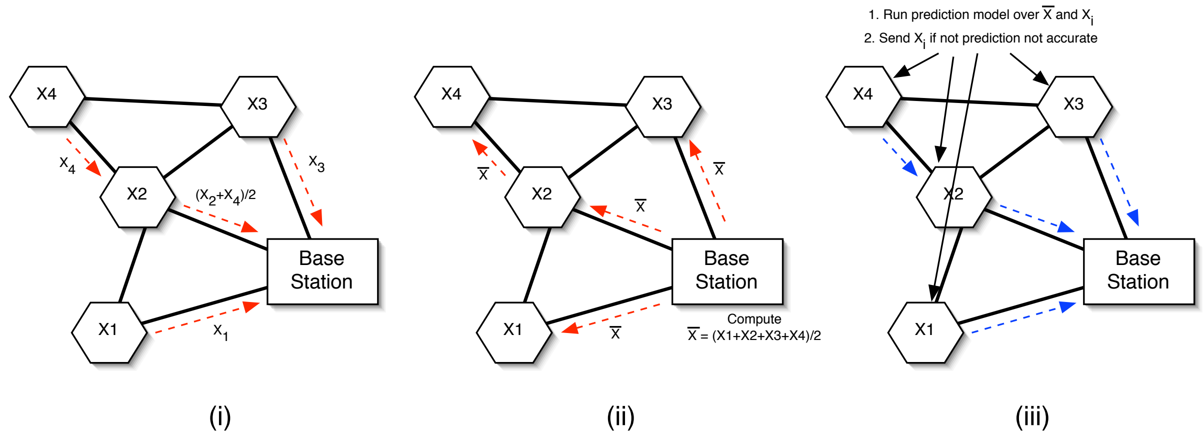

1. Run prediction model over $\overline{X}$ and X_i
2. Send X_i if not prediction not accurate
$(X_2+X_4)/2$
Compute $\overline{X} = (X1+X2+X3+X4)/2$
Base Station
(i)
(ii)
(iii)

Figure 4: The Average Model uses the average of all variables to predict each variable: (i) In-network aggregation is used to compute *average* efficiently; (ii) *average* is disseminated back into the network; (iii) Each node decides if it needs to send its value up to the base station given that *average* is already known.

Given this, we observe that the *expected cost* incurred by the group of sensor nodes C_i is independent of the rest of sensor network, *i.e.*, the total communication cost of (a) collecting the values of variables in C_i at one node, (b) checking whether the prediction is accurate, and (c) communicating a subset of values to the base station as needed, depends only on the members of C_i and the communication topology of the network. More precisely, if C_i^{root} were chosen to be the sensor node at which the data corresponding to C_i is collected, then:

- **intra-source** $= \Sigma_{x \in C_i} comm(x, C_i^{root})$
- **source-sink** $= m_i \times comm(C_i^{root}, 0)$

Moreover, C_i^{root} itself can be computed simply as:

$$C_i^{root} = \operatorname*{argmin}_{root \in \mathbf{X}} \Sigma_{x \in C_i} comm(x, root) + m_i\, comm(root, 0)$$

Note that C_i^{root} is not required to be in C_i, and we frequently observe otherwise in our experiments.

This independence between costs of different cliques allows us to effectively develop algorithms for finding effective Disjoint-Cliques model solutions. The problem of finding the *optimal* Disjoint-Cliques model, unfortunately, is NP-hard, even if an oracle that instantaneously computes the data reduction factor for any clique is provided (reduction from *minimum 3-dimensional assignment*).

4.2 Exhaustive Algorithm for finding Optimal Solution

Figure 5 illustrates a dynamic programming based algorithm that finds the optimal solution for a given instance of the problem. The algorithm proceeds by finding the optimal solution for each subset of attributes in a bottom-up fashion, utilizing the *principle of optimality* to reduce unnecessary computation. The complexity of this algorithm is $O(n4^n + M2^n)$ (where M is the cost of computing the data reduction factor for a given clique C_i) which makes it prohibitively expensive except in simplest of sensor networks.

4.3 Greedy Heuristic Algorithm

Figure 6 shows a simplified version of a greedy algorithm we employed for finding a Disjoint-Cliques model. The algorithm begins by choosing an arbitrary attribute X_x from the set of all attributes, finds the clique containing X_x that has the largest per-attribute data reduction factor, and greed-

```
EXHAUSTIVE($\mathbf{X}$, $comm : \mathcal{N} \times \mathcal{N} \rightarrow R$)
 FOR $i$ := 1 TO $|\mathbf{X}|$ DO
   FOR EACH $C \subseteq \mathbf{X}$ SUCH THAT $|C| = i$ DO
     $m_C$ $\leftarrow$ DATA REDUCTION FACTOR FOR CLIQUE $C$ (SECTION 4.4)
     $costC \leftarrow \min_{rootC \in \mathbf{X}} \{ \Sigma_{x \in C} comm(x, rootC) + m_C\ comm(rootC, 0)\}$
     FOR EACH $C_1 \subset C$ DO
       $c \leftarrow cost(C_1) + cost(C - C_1)$
       IF $c < costC$ THEN
         $costC \leftarrow c$
         $solutionC \leftarrow \{C_1, C_2\}$
       FI
     DONE
     $cost(C) \leftarrow costC, solution(C) \leftarrow solutionC$
   DONE
 DONE
```

Figure 5: Dynamic programming based optimal algorithm

```
GREEDY-K($\mathbf{X}$, $comm : \mathbf{X} \times \mathbf{X} \rightarrow \mathbf{R}, \mathbf{k}$)
 LET $covered \leftarrow \{\}, solution \leftarrow \{\}$
 WHILE $covered \neq \mathbf{X}$ DO
   LET $X_x \notin covered$ BE AN UNCOVERED ATTRIBUTE CHOSEN ARBITRARILY.
   LET $\mathcal{C}_X$ BE THE SET OF ALL CLIQUES $C$ SUCH THAT $C \subseteq (\mathbf{X} - \mathbf{covered})$, $X_x \in C$ AND $|C| = k$.
   FOR EACH $C \in \mathcal{C}_X$
     IF $C$ CONTAINS TWO ELEMENTS $a$ AND $b$ SUCH THAT:
       $comm(a, b) \geq \frac{1}{4} max_{u,v \in \mathbf{X}} comm(u, v)$
       $\mathcal{C}_X \leftarrow \mathcal{C}_X - C$
     ELSE
       COMPUTE $m_C \leftarrow$ DATA REDUCTION FACTOR FOR CLIQUE $C$ (SECTION 4.4)
     FI
   DONE
   LET $C' \in \mathcal{C}_X$ BE THE CLIQUE WITH LARGEST $\frac{m_{C'}}{|C'|}$
   $solution \leftarrow solution \cup \{C'\}, covered \leftarrow covered \cup C'$
 DONE
```

Figure 6: Our Greedy-k algorithm

ily chooses this clique to be in the final solution. It then removes the attributes contained in this clique, including X_x, from the set of all attributes, and repeats the procedure till all attributes are included in some clique in the solution. The algorithm also avoids computing data reduction factors for cliques that contain attributes too far from each other in the sensor network. Because of the high intra-source communication required to collect such attributes at one location, and also because spatial correlations tend to be inversely proportional to distance, such cliques are unlikely to be part of the optimal solution. Finally, we further control the running cost of this algorithm by restricting the sizes of cliques considered in the final solution through the parameter k.

Complexity: The complexity of this greedy algorithm is upper-bounded by $O(\binom{n}{k}M)$ where M is the cost of finding data reduction factor for a given clique, but is typically much less than that because we aggressively prune away cliques that are unlikely to be in the final solution.

4.4 Computing data reduction factors

Given a set of attributes C, the computation of the expected data reduction factor if using a probabilistic model $\mathcal{M}_C$ for predicting these attributes depends on the exact form of the model used. As an example of how this computation proceeds, consider a clique of size 1 containing attribute X_1, at time 1, and let X_1 be modeled using a linear Gaussian. The expected data reduction factor for such a clique can be written as:

$$E(\textit{data reduction factor}) = 1/E(num\ steps\ before\ a\ prediction\ error)$$

where the latter term can be computed as:

$$\begin{aligned} E(num\ steps\ before\ a\ prediction\ error) &= \\ 1 \times p(X_1^1 \notin [\mu_1 - \epsilon, \mu_1 + \epsilon]) \\ +2 \times p(X_1^1 \in [\mu_1 - \epsilon, \mu_1 + \epsilon]\ \&\ X_1^2 \notin [\mu_2 - \epsilon, \mu_2 + \epsilon]) \\ +3 \times p(X_1^1 \in [\mu_1 - \epsilon, \mu_1 + \epsilon]\ \&\ X_1^2 \in [\mu_2 - \epsilon, \mu_2 + \epsilon] \\ \&X_1^3 \notin [\mu_3 - \epsilon, \mu_3 - \epsilon]) + 4 \times \ldots\ldots \end{aligned}$$

where μ_i denotes the mean of the Gaussian at time i (the precise values of the means are irrelevant in this expression; only the variances of the Gaussians matter).

Even for this simplest possible model, there does not exist a closed form expression for computing the expected data reduction factor. We instead use the standard approach of Monte Carlo integration to compute this value numerically. We omit the details of the process for brevity.

5 Evaluation

In this section, we present an extensive evaluation of Ken over traces from two real-world sensor network deployments. Our results demonstrate the effectiveness of Ken in reducing the communication cost and the energy consumption of the system without undue sacrifice in result quality or frequency, even with a relatively simple class of models such as Disjoint-Cliques models. We begin with a discussion of the experimental setup, the traces we use, and the specific form of models that we chose for these deployments.

5.1 Experimental setup

We have implemented Ken in Matlab on traditional PC hardware, and we are in the process of implementing the on-sensor component of Ken. We evaluate Ken's performance by using traces from two real-world sensor network deployments: (1) *Lab:* A deployment at the Intel Research Lab in Berkeley [17] consisting of 49 mica2 motes and (2) *Garden:* A deployment at the UC Berkeley Botanical Gardens consisting of 11 mica2 motes. We limit our discussion to three of the attributes sensed by these motes, *temperature*, *humidity* and *voltage*. Figures 7 and 8 plot several representative days of *temperature* and *humidity* data for both deployments. All of these fluctuate cyclically (with a period of 24 hours) to a first approximation. However, we qualitatively observe several secondary characteristics, especially in the lab dataset. For example, the midday temperature rise on the first day (minutes 400 - 700) is smooth whereas the rise on the third day (minutes 3100 - 3600) is abrupt and jagged. The garden

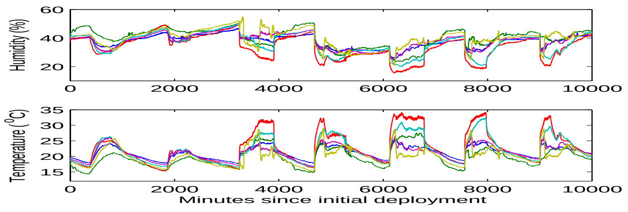


Figure 7: Lab data overview

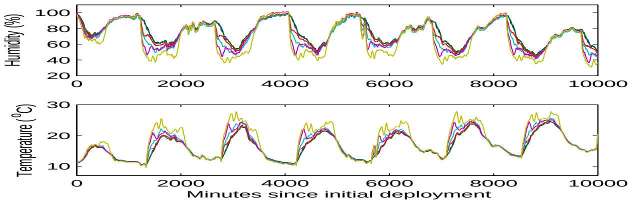


Figure 8: Garden data overview

dataset tends to exhibit more consistency. We will see that these datasets reveal different facets of Ken.

For both these deployments, we chose to model the attributes as time-varying multivariate Gaussians (Cf. Figure 2). As demonstrated in previous work on these datasets [7, 8], multivariate Gaussians fit this data quite well, especially the data from the *garden* deployment. We estimated the model parameters using the first 100 hours of data (*training* data), and used traces from the next 5000 hours (*test* data) for evaluating Ken.

Unless otherwise specified, we used error bounds of 0.5^oC for *temperature*, 2% for *humidity* and 0.1V for *battery voltage*; in other words, the user would like the difference between the reported value and the actual observed value to be less than these error bounds at all times.[7] Additionally, the user would like the sensors to sample hourly. We also experimented with other various sampling rates and bounds, and observed very similar performance trends.

5.2 Comparison Schemes

We evaluate and compare the following schemes on these datasets:

- **TinyDB [18]**: This scheme always reports all sensor values to the base station. The guarantees provided by this scheme are the strongest as the data errors are always zero.
- **Approximate Caching [23]**: This scheme caches the last reported reading at the *sink* and *source* , and *source*s do not report if the cached reading is within the threshold of the current reading. In modeling terms, this is a degenerate Markov model. We set approximate caching's reporting threshold to match that of Ken (e.g., 0.5^oC for temperature).
- **Ken with *Disjoint-Cliques (DjC)* and *Average (Avg)* models**: We study both the Disjoint-Cliques (DjC) models (Section 4) and the Average (Avg) model (Example 3.5) within the Ken framework. The Disjoint-Cliques models permit a finely-tuned study of the impact of spatial correlations via restricting the maximum possible clique size. Unless otherwise specified, we use the Greedy-k heuristic algorithm to find the Disjoint-Cliques model used in Ken; we denote by DjCk the model found by Greedy-k.
- **Single node dual models [14]**: This technique is equivalent to Ken with DjC1 (maximum clique size restricted to one), and hence, the comparison to this technique will often be implicit.

[7]These were the bounds deemed acceptable for biological research purposes by our biology collaborators.

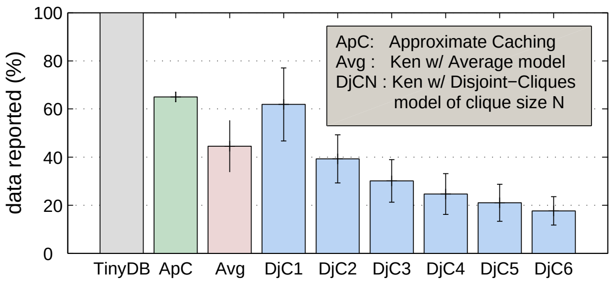


Figure 9: % of data reported under various schemes for the garden dataset.

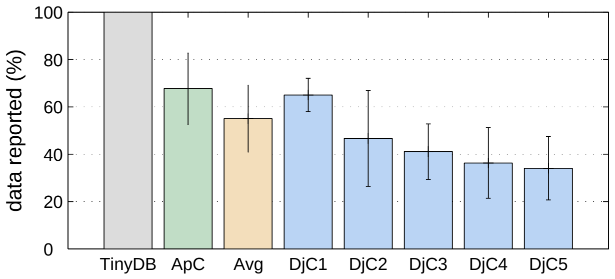


Figure 10: % of data reported under various schemes for the lab dataset.

Also, we choose to permit one data unit per message. This implies we do not aggregate data as it flows up towards the root, and data sent is equal to messages sent. This choice allows us to fairly compare against TinyDB, which likewise sends one data unit in each message. A count of messages sent also serves as a fair proxy for energy expended; on typical sensor network platforms, sending and receiving messages dominates energy consumption [21].

5.3 Topology independent characteristics

We begin with investigating the effectiveness of these schemes at reducing the amount of data reported to the base station, while ignoring any network topology dependent characteristics. In other words, we compare the average number of data values reported to the base station for these schemes.

Figure 9 and Figure 10 plot the percentage of nodes reported for the garden data and lab data respectively. These figures immediately distinguish several facets. First, Ken and Approximate Caching both achieve significant savings over TinyDB. This occurs because the user relaxes her error tolerance in exchange for communications savings. While the magnitude of these savings will clearly vary with the desired error bounds, the figures indicates that even for a modest bound of 0.5^oC, 65% data reduction is readily attainable; TinyDB pays a hefty premium for exactness.

Second, Average reports at a higher rate than Disjoint-Cliques with max clique size restricted to 2 (DjC2). Given that Average also incurs a high *fixed* aggregation and dissemination cost, we don't expect Average to outperform DjCk, $k \geq 2$ (though it could conceivably outperform ApC and DjC1). Hence, we do not consider the Average model further in this section.

Third, Disjoint-Cliques with a max clique size of 1 (DjC1) and Approximate Caching miss at comparable rates of about 65% in these datasets. This suggests that capturing and modeling temporal correlations alone may not be sufficient to outperform caching. However, we see the definitive advantage of larger cliques: the data reported falls rapidly as Ken is able to utilize spatial correlations. Ken often has the opportunity to select and report those few nodes which serve to strongly indicate the readings of other nodes. Approximate caching fundamentally cannot take advantage of spatial correlations, and as a result, suffers in such environments.

Lastly, we begin to witness several underlying dataset differences between Lab and Garden: The garden dataset's stronger spatial and temporal correlations yield more data reduction (21% data reported at cliques of size 5, DjC5) compared to that of Lab's (36% data reported).

5.4 Topology dependent costs

Next we evaluate the effect of intra-source and source-sink communication costs on the performance of these schemes. Recall that for Disjoint-Cliques the intra-source cost consists of the cost of collecting data values of all clique members at the clique root, and the source-sink cost consists of the cost of reporting a (possibly empty) subset of the data values to the base station (Section 4). The relative importance of these costs in the total communication cost strongly influences the cliques chosen for this class of models. It is more worthwhile to reduce data early in the network if the cost to communicate to the sink is high. Conversely, the higher (fixed) intra-source costs may make large cliques unattractive if the cost to communicate to the sink is low.

Approaching this evaluation in two phases, we first construct synthetic topologies on the garden dataset. This lets us quantify the merit of various clique sizes while controlling the cost to the base station. Afterward, we study actual topologies of the lab data.

Through both of these topology studies, we compared the use of the optimal dynamic programming solution (Section 4.2) with the greedy heuristic (Section 4.3). While the dynamic programming's optimality is appealing, we ultimately ran most experiments with the greedy heuristic because of its computing time speedup. Figure 11 shows a comparison of the two algorithms. The greedy algorithm and the optimal algorithm perform very comparably, with the greedy algorithm very often within 12% of the optimal.

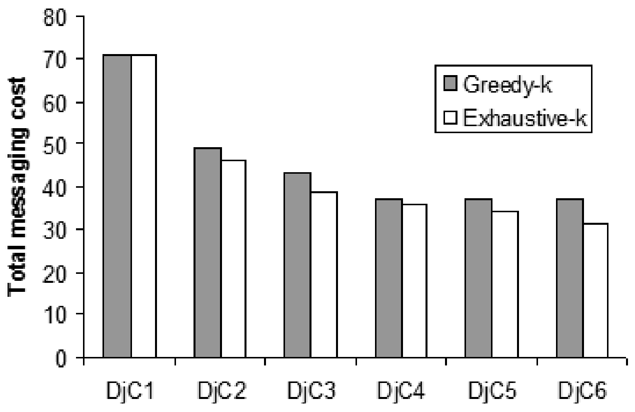


Figure 11: Comparing *Greedy-k* and *Exhaustive-k* for various k (*Exhaustive-k* is *Exhaustive* algorithm restricted to cliques of size k).

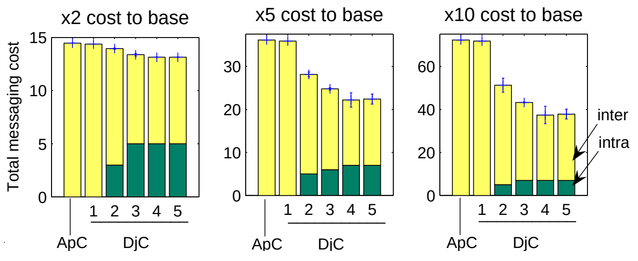


Figure 12: Total communication cost for the *garden* dataset for the various schemes under different network topologies.

Garden Dataset:
The garden dataset contains 11 nodes, with equivalent path costs between every pair of these 11 nodes. To create varied topologies, we manipulate the path cost to the base as a multiple of the path cost between pairs of nodes. By applying the greedy heuristic algorithm outlined in Section 4.3, we derive the clique partitions and observe the resulting communication costs. Recall that the greedy heuristic can be instrumented with a limiting maximum clique size k, effectively limiting the use of spatial correlations. We use this max clique size to study total messaging costs given artificial limits. Figure 12 illustrates these costs, decomposed into intra-source and source-sink costs (pointed at by the "inter" and "intra" arrows respectively).

As the cost to the base station increases to 5 and 10 times the cost to communicate within the clique, larger clique sizes become increasingly attractive. This behavior follows our expectations, since data reduction is a priority when transmitting data is more expensive. Larger cliques make far fewer gains when the cost of communicating with the base station is comparable to the cost of communicating with neighbors.

We begin to see that larger clique sizes (5 nodes) do not offer limitless gains, even when the source-sink cost is relatively high (Figure 12, right panel). The leveling off of the messaging cost (e.g. from DjC4 to DjC5) means that the greedy heuristic is not using larger clique in its solution, even though the heuristic is permitted to use larger cliques. This makes sense because nodes distant from one another in the

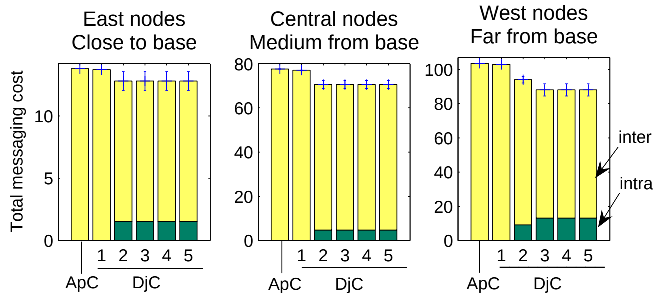


Figure 13: Total communication cost for the *Lab* deployment partitioned into three node groups, *east*, *central* and *west*. Note that the three plots have different y-axes.

physical deployment may not have sufficiently strong spatial correlations incentive to form into a larger clique. In addition, the increased intra-source costs may start dominating the source-sink costs.

Lab Dataset:
We next investigate the lab data and its accompanying connectivity data. In this deployment, the network has been instrumented such that the base station resides at the east end of the network. This leads to consider and evaluate three regions of the network separately: East, Central and West. Since link quality is roughly proportional to geographic distance, these regions correspond to excellent (x1.5 cost to base), good (x3 cost to base) and moderate (x6 cost to base) costs to base respectively. Figure 13 plots the total messaging costs resulting from application of the greedy heuristic (Section 4.3).

We can see that the areas closer to the base station do not benefit from larger cliques. This follows our previous findings: nodes in this area generally do not experience a net benefit after accounting for the fixed intra-source costs. On the other hand, the area further from the base station incur modest data reduction gains from larger cliques. The diameter of the network is about 6 hops, yet we do not see as much gain for larger cliques as in the garden data. This is largely because multivariate Gaussians are not a perfect fit for this dataset. Human intervention (in particular, turning the air conditioning on and off) results in this data being much harder to predict than the garden data.

5.5 Multiple Attributes per Node

Correlations among attributes offer another opportunity to employ Ken's model-based compression. For example, a sensor node's battery *voltage* often peaks when the *temperature* is high. Abstractly, we can think of multiple attributes per physical node as multiple logical nodes with zero communication cost among them. We then can apply the same Ken Disjoint-Cliques model here. More significantly, larger cliques always outperform smaller cliques in this scenario since the intra-source communication cost is 0. Figure 14 plots Ken's data reduction savings for a single node with three attributes: *temperature*, *humidity* and *voltage*. These are combined in various configurations as shown in the figure. We can see that while any combination of compression

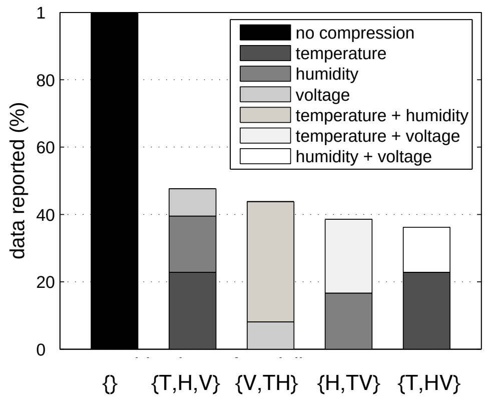


Figure 14: Compression using correlations among *temperature*, *humidity* and *voltage* in Garden deployment: *e.g.*, $\{V, TH\}$ denotes that $temperature$ and $humidity$ were modeled as a size-two clique, whereas $voltage$ was compressed as a single-node clique.

far exceeds no compression, utilizing inter-attribute correlations further improves the compression ratio. Ken-based models incur no cost while providing immediate data reduction.

6 Extensions

Our approach to approximate data collection in Ken has many interesting extensions that we plan to explore.

Richer Probabilistic Models: The distributed nature of data acquisition leads to interesting tradeoffs in model selection that have not been explored before. We already saw some of these tradeoffs with Disjoint-Cliques models where simpler models were preferred over more sophisticated models even though the latter class of models was a better fit for the data. We plan to explore other such families of models that lend themselves to easy localization and distribution, and evaluate their effectiveness in approximate data collection.

Probabilistic Reporting: Although Ken's deterministic error guarantee is desirable in many applications, there are applications that can accept non-deterministic errors with a strong probabilistic guarantee in exchange for further reduction in communication cost. Ken can be easily extended to do probabilistic reporting, wherein each node determines whether to report a value to the base station based on a probabilistic function. Our preliminary experiments indicate that *relaxed step functions* and *logistic functions* can be used very effectively for this purpose.

Application to Caching, Distributed Streams: Approximate caching and distributed streams are two application domains where our probabilistic modeling-based approach would extend naturally. For example, instead of storing just the last cached value at a client [23], we can instead use probabilistic models to predict how the cached value changes, and thus use patterns in data evolution to reduce the data communication between a server and a client. Data collection from distributed streams [3] can also be made significantly more efficient by using the approach we advocate in this paper.

Dynamic Network Topologies: So far, we have assumed a static network topology. As we know first-hand, real sensor networks are dynamic and topologies evolve over time. For Ken to handle network topology changes efficiently, it requires Ken to interact closely with the underlying network layer. TinyOS exposes an interface for applications to trigger network topology updates. We plan to take advantage of this interface when we implement Ken on TinyOS motes for Ken to control the network topology updates and incrementally update its network link costs as they change.

Detection of Node Failures: When the basestation does not hear from a node for an extended period of time, Ken needs to be able to distinguish between node failure and the case that the data is always within the error bound. Ken can make this distinction based on its knowledge of the expected miss rate for the node. Once again, use of robust probabilistic modeling techniques allows us to take a systematic approach to this problem.

Robustness to Message Loss: Though it is possible to use reliability protocols such as end-to-end ack plus retransmissions to eliminate message losses, such protocols incur very high overheads and may not be suitable in all cases. However, the Markovian nature of the probabilistic models we use provides us with another, more systematic, approach to dealing with this problem. We observe that, in such models the *future* is independent of the *past* given the *present*, and we can use this to control the inconsistencies in the reported data. As an example, periodic heartbeats can be used to "re-synchronize" models even in presence of arbitrary message losses, leading to only transient data inconsistencies.

7 Conclusions

Database query processing ideas can play an important role in sensor networks. However, our experience with early adopters of sensornet query engines shows that their typical workloads are not well-served by the energy optimizations developed in the early-phase research. Our work here on Ken revisits the design of sensornet query processing in the light of a standard practical workload, "`SELECT *`" data collection queries. It also applies to another standard workload, anomaly detection, in a natural and efficient way.

Ken, like other work in recent years (e.g., BBQ [7] and Jain, et al. [14]), focuses on using probabilistic models to provide approximate answers efficiently. This is a rich area for database research in general, and is particularly well suited to sensor networks, since sensor data is by nature noisy and uncertain, but often drawn from fairly smooth distributions [6]. Ken and BBQ present two complementary points in the design space for approximate sensornet queries using probabilistic models. The advantage of Ken is it assures the user of the faithfulness of the approximate answer to the actual sampled value, without assuming model correctness, while significantly reducing the volume of data transmitted. This area warrants further research in order to chart the space and understand the applicability of various designs to different problems.

References

[1] T. Bell, I. H. Witten, and J. G. Cleary. Modeling for text compression. *ACM Comput. Surv.*, 21(4):557–591, 1989.

[2] J. G. Cleary and I. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans on Communications*, 1984.

[3] O. Cooper, A. Edakkunni, M. Franklin, W. Hong, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, and E. Wu. Hifi: A unified architecture for high fan-in systems. In *Proceedings of VLDB*, 2004. Demo.

[4] Crossbow, Inc. Wireless sensor networks (mica motes). `http://www.xbow.com`.

[5] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence is Good: Dependency-Based Histogram Synopses for High-Dimensional Data. In *SIGMOD*, May 2001.

[6] A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR*, 2005.

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

[8] A. Deshpande, C. Guestrin, S. Madden, and W. Hong. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, 2005.

[9] M. Garofalakis and P. Gibbons. Probabilistic wavelet synopses. In *SIGMOD*, 2002.

[10] S. M. Graham Cormode, Minos Garofalakis and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD*, 2005.

[11] C. Guestrin, P. Bodik, T. R., P. Mark, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, 2004.

[12] J. Hellerstein and W. Wang. Optimization of in-network data reduction. In *DMSN*, 2002.

[13] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of ACM MOBICOM*, Boston, MA, August 2000.

[14] A. Jain, E. Chang, and Y.-F. Wang. Adaptive stream resource management using Kalman Filters. In *SIGMOD*, 2004.

[15] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *ICDE*, 2003.

[16] P. Levis and et al. Fire alert-and-response system prototype, 2004. http://www.cs.berkeley.edu/ pal/cs199.

[17] S. Madden. Intel lab data, 2003. http://berkeley.intel-research.net/labdata.

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

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

[20] S. Madden, W. Hong, J. M. Hellerstein, and M. Franklin. TinyDB web page. `http://telegraph.cs.berkeley.edu/tinydb`.

[21] Moteiv. Telos Revb datasheet, December 2004. http://www.moteiv.com/pr/2004-12-09-telosb.php.

[22] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys*, 2004.

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

[24] V. Poosala and Y. E. Ioannidis. "Selectivity Estimation Without the Attribute Value Independence Assumption". In *VLDB*, 1997.

[25] K. Sayood. *Introduction to data compression*. Morgan Kaufmann Publishers Inc., 1996.

[26] C. Sharp, S. Schaffert, A. Woo, N. Sastry, C. Karlof, S. Sastry, and D. Culler. Design and implementation of a sensor network system for vehicle tracking and autonomous interception. In *EWSN*, 2005.

[27] D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4), 1973.

[28] G. Tolle. Sonoma redwoods data, 2005. http://www.cs.berkeley.edu/ get/sonoma.

[29] J. S. Vitter, M. Wang, and B. Iyer. "Data Cube Approximation and Histograms via Wavelets". In *CIKM*, 1998.

[30] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, June 1987.

[31] O. Wolfson, S. Chamberlain, S. Dao, and L. Jiang. Location management in moving objects databases. In *WoSBIS*, 1997.

[32] A. D. Wyner and J. Ziv. The rate-distortion function for source coding with side information at the decoder. *IEEE Transactions on Information Theory*, 1976.

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