Enabling ϵ-Approximate Querying in Sensor Networks*

LIU Yu, Jianzhong Li, Hong Gao, Xiaolin Fang
Data and Knowledge Engineering Research Center
Harbin Institute of Technology, Harbin, China
{pamws,lijzh,honggao,xlforu}@hit.edu.cn

ABSTRACT

Data approximation is a popular means to support energy-efficient query processing in sensor networks. Conventional data approximation methods require users to specify fixed error bounds *a prior* to address the trade-off between result accuracy and energy efficiency of queries. We argue that this can be infeasible and inefficient when, as in many real-world scenarios, users are unable to determine *in advance* what error bounds can lead to affordable cost in query processing. We envision *ϵ-approximate querying* (*EAQ*) to bridge the gap. *EAQ* is a uniform data access scheme underlying various queries in sensor networks. It allows users or query executors to incrementally 'refine' previously obtained approximate data to reach arbitrary accuracy. *EAQ* not only grants more flexibility to in-network query processing, but also minimizes energy consumption through communicating data upto a *just-sufficient* level. To enable the *EAQ* scheme, we propose a novel *data shuffling* algorithm. The algorithm converts sensed datasets into special representations called *multi-version array* (*MVA*). From prefixes of *MVA*, we can recover approximate versions of the entire dataset, where all individual data items have guaranteed error bounds. The *EAQ* scheme supports efficient and flexible processing of various queries including *spatial window* query, *value range* query, and queries with *QoS* constraints. The effectiveness and efficiency of the *EAQ* scheme are evaluated in a real sensor network testbed.

1. INTRODUCTION

Wireless sensor networks are unattained networks of battery-powered sensor motes with limited sensing, computing, storage and wireless communication capabilities. They open up new opportunities to observe the physical world, and enable us to gather data that was once difficult, expensive, or even impossible to collect [5]. Sensor networks for *data-intensive* applications [30, 31] introduce exciting challenges to researchers on data engineering. A fundamental research problem is how to process query or obtain data from sensor networks while minimizing energy consumption.

*Supported in part by the National Grand Fundamental Research 973 Program of China under grant $2006CB303000$, the National Natural Science Foundation of China (NSFC) under grant 60533110, and the Joint Research Fund of NSFC and Hong Kong Research Grant Council (RGC) under grant 60831160525.

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

In-network data approximation is a popular means to support energy-efficient query processing [28]. A number of approximation methods for sensor data have been developed [3, 5, 20, 7, 14, 12]. Conventional data approximation methods depend on *error bounds* to address the 'trade-off' between result accuracy and energy efficiency of queries. To conduct data approximation, error bounds almost always have to be specified by users before queries are submitted into the sensor network, and there is hardly any means to adjust the error bounds during runtime. The obstacle is that the relation between result accuracy and energy efficiency of queries is constantly changing over time, and, in many scenarios, the changing trend of the relation is not available to users *a prior*. In this regard, it is impractical to depend on users to predict what error bounds can lead to affordable cost in query processing. And the obstacle often results in considerable and unnecessary energy consumption, such as in the following scenarios.

First, specifying a fixed error bound for a continuous query may yield different result sizes at different time. Given a seemingly loose error bound tossed by the user, it is unpredictable whether or when the error bound can lead to affordable result sizes.

Scenario 1 *In 'botanic monitoring' [30], a continuous query collects all light readings periodically from redwood trees, so that a contour map image can be updated at the base station accordingly. The query may specify a fixed error bound, say ± 50 Lux, for in-network data approximation hoping that the result sizes are affordably small for long-running. At night, when light readings are between 0 and 100 Lux, the error bound can lead to high compression ratio, which is desirable. Unfortunately, during daytime the compression ratio may become unexpectedly low, for the deviations of light values are mostly greater than the error bound. Hence, the query incurs overwhelming energy consumption during daytime. And the long-running cost of the query is unaffordable, for it may shorten the network lifetime by large factors.*

Second, a user may have obtained a set of approximate data, and hopes that related queries can reuse them to avoid repetitive communication. However, with existing methods, approximate data generated for one query are not always reusable for other ones.

Scenario 2 *In 'military surveillance', a commander requires the disposition data of enemy troops to aid decision making. An overall view of the data has to be obtained first, in real-time, to learn the general situation. Detailed views of the data are then needed for analyzing the enemy troops and making decisions. The overall view is always (partially) covered by detailed views. However, with existing approximation methods, queries for the detailed views are typically unable to reuse approximate data obtained for the overall view. Hence, the repetitive information has to be transmitted multiple times, which is a substantial source of undesirable cost.*

To resolve all above problems, we envision *ϵ-approximate querying* (*EAQ*). *EAQ* is a uniform data access scheme allowing users or various query executors to efficiently 'refine' previously obtained approximate data to reach arbitrary accuracy. The intuitive idea is to split the data access process into a series of iterative sub-queries, each of which incrementally retrieves a small fraction of information and extends the outcome of previous sub-query. All intermediate outcomes of sub-queries are approximate versions of the targeted dataset with guaranteed error bounds. As for Scenario 2, *EAQ* initially obtains a sketch of the battlefield data at minimum cost. Then if time allowed, it invokes consecutive sub-queries to refine the sketch and yield better versions of approximate data. In this manner, the commander eventually obtains data that are either sufficiently accurate for making sound decisions or having the maximum possible accuracy for the time allowed. For one thing, *EAQ* overcomes the obstacle of having to toss a sometimes irrational error bound before query execution. For the other, it offers the flexibility to query network-stored sensor data at multiple accuracy levels. In addition, by selecting and communicating data upto a just-sufficient level, *EAQ* can reduce both response time and energy cost for various queries in sensor networks.

Enabling *EAQ* in sensor networks is non-trivial. As a uniform data access scheme, *EAQ* should generate refinable approximate data that are usable by various queries. In other words, the outcomes of *EAQ* may be converted, by various query executors, to any potential forms, such as statistics, histograms, contour maps, frequent patterns, etc. The diversity in data utility requires that all approximate data generated by *EAQ* should not deviate from the accurate values more than an upper bound. Namely, *EAQ* should always generate approximate datasets with guaranteed maximum absolute, or L_∞-norm, error bounds [20]. If otherwise approximate datasets only ensure cumulative error bounds, such as L_2 or L_1-norm, then the accuracy of individual sensor readings does not have a guarantee. Consequently, many important queries (such as *MAX* and *TOP-K*) and useful mining tasks [12] may yield results with unbounded errors! During *EAQ*-based data access, the same collection of data may be acquired multiple times at different accuracy levels. And the challenge arises in how to bound the accumulated communication for obtaining multiple versions of approximate data. So far as we know, no prior data approximation techniques, not even transform domain methods [2] like wavelet, can fully eliminate the redundancy among multiple versions of approximate data when subject to minimizing L_∞-norm errors. In summary, we identify two research problems in enabling *EAQ*, namely, (i) how to produce and communicate multiple versions of approximate data at an overall cost comparable to that of acquiring the finest version alone, and (ii) how to ensure that all approximate versions have guaranteed L_∞-norm error bounds.

The core technique we develop for enabling *EAQ* is a novel *data shuffling* algorithm. The algorithm converts a collection of sensed data into a special representation called *multi-version array* (*MVA*). An *MVA* is an equivalent representation of the original dataset, and it takes strictly less or equal storage. Yet, given any *MVA* prefix with three or more items, we can always recover an approximate version of the entire dataset together with an ε indicating the guaranteed bound of maximum absolute, i.e. L_∞-norm, error. To conduct *EAQ* over 'shuffled' data, each sub-query of *EAQ* only retrieves a consecutive fragment of the *MVA*, which composes an extended *MVA* prefix when concatenated with all previous fragments. Since coarser versions of approximations (or shorter prefixes) are always contained in finer ones (or longer prefixes), the accumulated cost for retrieving multiple versions of approximate data is always bounded by that of retrieving the finest version (or the longest prefix). Even in the extreme case where a user insists on obtaining an approximate version of 100% accuracy, the cost for conducting *EAQ* is still bounded by the volume of the targeted dataset, or the length of the entire *MVA*. The proposed *EAQ* scheme can support flexible and efficient processing of many important query types in sensor networks, including *spatial window* query [32] and multi-dimensional *value range* query [23, 6]. The iterative framework of *EAQ* also enables queries with various *QoS* constraints, such as *response deadline*, *energy budget*, etc.

The main contributions of the paper are as follows.

- We envision a uniform scheme *EAQ* for accessing network-stored sensor data. *EAQ* allows for arbitrary-accuracy data access through runtime refinements. It brings more flexibility to in-network query processing, and potentially reserves energy by communicating data upto a *just-sufficient* level.
- To enable the *EAQ* scheme, a light-weight, low-order polynomial, *data shuffling* algorithm is proposed. It converts sensed datasets into representations called *multi-version array* (*MVA*). Notable features of *MVA* include (i) *MVA* is a rearrangement of the original dataset, values of data items remain unchanged, (ii) *MVA* can yield multiple approximate versions of the original dataset with guaranteed L_∞-norm error bounds, (iii) coarser versions of approximate data and their error bounds are all embedded in finer versions.
- We present *EAQ*-based query processing by examples. *EAQ* can enable a number of important types of queries, including *spatial window* query, multi-dimensional *value range* query, and queries with *QoS* constraints.
- Last but not least, we conduct experimental studies in a real sensor network testbed to evaluate the proposed techniques.

The rest of this paper is organized as follows. Section 2 formulates research objectives. In Section 3 and 4, we present techniques for enabling *EAQ*. Section 5 describes *EAQ*-based query processing. Results of empirical studies are shown in Section 6. Section 7 surveys related work. Finally, we draw conclusions in Section 8.

2. PROBLEM FORMULATION

2.1 Preliminaries

We consider a sensor network $\mathcal{N}$, in which sensor motes are deployed according to a *regular grid* in the area of interest. In the network grid, two sensor motes can communicate directly if they are in the same rectangular region of size $(r+1) \times (r+1)$, where $r \geq 1$ is the distance of *one hop*. The network performs data sampling once every δ seconds. For each sampling period, every sensor mote observes the environment and samples a data item $\mathbf{o} = (h, t, v_1, v_2, \ldots, v_d)^\intercal$, where integer h is the *identifier* of mote s_h that yielded the data item, integer t is the *time stamp* when the data item is sampled, and the list of *normalized* values $\mathcal{V} = (v_1, v_2, \ldots, v_d)$ are the attributes observed by mote s_h, such as *temperature, light, voltage, etc.*. We assume there exists a one-to-one mapping between sensor motes' identifiers h and geographic locations (x, y). The identifiers h embedded in data items are called *location stamps*. Data are stored in-network for answering queries until they become staled (i.e. time stamps older than a threshold $T \gg \delta$). Let t_{now} be the time stamp of *now*, the time stamp t in data items can be normalized as $\bar{t} = (t_{now} - t) / T$. The set of all data items sampled within one sampling period forms a *snapshot* of network $\mathcal{N}$. We define *λ-localized dataset*, or *dataset*

for short, as being any non-empty subset of a snapshot in which all data items are sampled by motes within λ hop(s) of each other.

Definition 1 (λ-Localized Dataset) *Let $\mathcal{D}$ be a non-empty set of data items collected in sensor network $\mathcal{N}$. $\mathcal{D}$ is called a λ-localized dataset, or dataset for short, if, for any two data items $\mathbf{o}_u$ and $\mathbf{o}_v$ in $\mathcal{D}$, it always holds that (i) the two time stamps t_u and t_v are within $\pm\delta$, i.e. $\left|\bar{t}_u - \bar{t}_v\right| < \delta \,/\, T$, and (ii) the two sensor motes s_u and s_v are within λ hop(s), i.e. s_u and s_v are in the same rectangular region of size $(\lambda \cdot r + 1) \times (\lambda \cdot r + 1)$ in the network $\mathcal{N}$.*

Note that, every mote samples exactly one data item during each sampling period, therefore, all location stamps in the same dataset are guaranteed to be unique. We thus formulate the *canonical form* of dataset $\mathcal{D}$ as vector $\mathcal{D} = \left(\mathbf{o}_{h_1}, \mathbf{o}_{h_2}, \ldots, \mathbf{o}_{h_k}\right)^\intercal$, where data items are subscripted with their location stamps, and are arranged in ascending subscripts order, i.e. $h_{i-1} < h_i$ $(i = 2, 3, \ldots, k)$.

2.2 EAQ Data Access Model

Given a targeted dataset $\mathcal{D} = \left(\mathbf{o}_{h_1}, \mathbf{o}_{h_2}, \ldots, \mathbf{o}_{h_k}\right)^\intercal$, the basic functions of *EAQ* are (i) obtain an approximate version $\widehat{\mathcal{D}}$ of dataset $\mathcal{D}$, which conceptually contains k equivalent data items, i.e. $\widehat{\mathcal{D}} = \left(\hat{\mathbf{o}}_{h_1}, \hat{\mathbf{o}}_{h_2}, \ldots, \hat{\mathbf{o}}_{h_k}\right)^\intercal$, but has smaller actual representation, and (ii) refine $\widehat{\mathcal{D}}$ to reach higher accuracy levels, where the accuracy is measured with maximum absolute error, or L_∞-norm,

$$L_\infty\left(\widehat{\mathcal{D}}, \mathcal{D}\right) = \max_{i=1}^{k} \left\|\hat{\mathbf{o}}_{h_i} - \mathbf{o}_{h_i}\right\|_\infty = \max_{i=1}^{k} \max_{j=1}^{d+2} \left|\hat{\theta}_j - \theta_j\right| \quad (1)$$

where θ_j is the j^{th} component in each data item $\mathbf{o}_i \in \mathcal{D}$, and $\hat{\theta}_j$ is the approximate counterpart in $\hat{\mathbf{o}}_i \in \widehat{\mathcal{D}}$. To be precise, θ_j and $\hat{\theta}_j$ can be the location stamp ($j = 1$), the time stamp ($j = 2$), or any attribute value ($j = 3, 4, \ldots, d + 2$) in respective data items.

We model the data access process of *EAQ* as a series of sub-queries $\rho_1, \rho_2, \ldots, \rho_q$. Each sub-query ρ_i retrieves a fraction Δ_i of data items from the targeted dataset $\mathcal{D}$, i.e. $\Delta_i \subset \mathcal{D}$, and, for all $i > 1$, ρ_i assembles Δ_i with the outcome $\mathcal{O}_{i-1}$ of previous sub-query ρ_{i-1} to form an extended outcome $\mathcal{O}_i$, i.e. $\mathcal{O}_i = \mathcal{O}_{i-1} \cup \Delta_i$. The sub-queries of *EAQ* can be called at any mote, as well as from the base station, during query processing. Figure 1 gives an example illustrating how *EAQ* is used for in-network query processing.

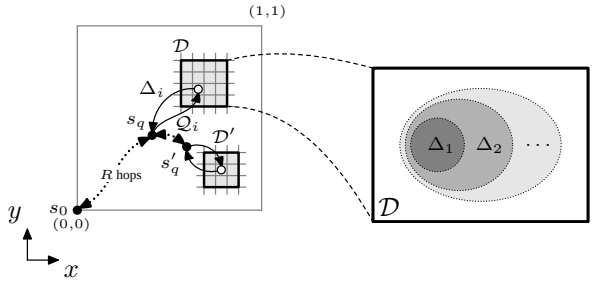


Figure 1: Modeling the *EAQ* data access scheme. An *ad hoc* query is propagated R hops from base station s_0 to sensor mote s_q, at which it discovers one targeted dataset $\mathcal{D}$ (i.e. the other is $\mathcal{D}'$). s_q invokes $\rho_1, \rho_2, \ldots$ to acquire $\mathcal{D}$ and obtains $\Delta_1, \Delta_2, \ldots$. For raw data acquisition, Δ_i's are directly forwarded to s_0. Otherwise, partial query results are generated at s_q from $\mathcal{O}_1, \mathcal{O}_2, \ldots$ and then routed to s_0.

Underlying the data access model of *EAQ*, two basic operators are needed by sub-queries for recovering approximate datasets and validating the accuracies of approximations. We draw specifications for the operators here, and present algorithms in Section 3.3.

Dataset Recovery (α) The *dataset recovery* operator α recovers an approximate version $\widehat{\mathcal{D}}$ of dataset $\mathcal{D}$ from the outcome $\mathcal{O}$ of respective sub-query, i.e. $\widehat{\mathcal{D}} = \alpha\left(\mathcal{O}\right)$.

Error Validation (ϵ) The *error validation* operator ϵ validates the accuracy of an approximate version $\widehat{\mathcal{D}} = \alpha\left(\mathcal{O}\right)$, In particular, $\epsilon\left(\mathcal{O}\right)$ should yield the same result as $L_\infty\left(\widehat{\mathcal{D}}, \mathcal{D}\right)$, but ϵ *must only* base on $\mathcal{O}$, and *nothing more* from dataset $\mathcal{D}$.

The cost for conducting *EAQ* over a particular dataset is the accumulated communication of all iterations of sub-queries. If it takes q iterations to obtain approximate version $\widehat{\mathcal{D}}$, i.e. $\widehat{\mathcal{D}} = \alpha\left(\mathcal{O}_q\right)$, then the accumulated cost can be expressed as follows.

$$\mathcal{C}\left(\widehat{\mathcal{D}}\right) = \sum_{i=1}^{q} \mathcal{C}\left(\mathcal{O}_i\right) = \sum_{i=1}^{q} c \cdot \left|\Delta_i\right| \quad (2)$$

where $\Delta_i \subset \mathcal{D}$ is the fraction of data items retrieved by the i^{th} iteration ρ_i, and c is the average cost (i.e. in the unit of *Joule per data item*) for accessing one data item from dataset $\mathcal{D}$.

It deserves our notice that, when $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$, the cost model in Equation 2 has a minimum,

$$\mathcal{C}^*\left(\widehat{\mathcal{D}}\right) = \min\left\{\sum_{i=1}^{q} c \cdot \left|\Delta_i\right|\right\} = c \cdot \left|\bigcup_{i=1}^{q} \Delta_i\right| \quad (3)$$

Equation 2 suggests that different iterations of sub-queries should retrieve non-overlapping fractions of the targeted dataset.

2.3 Research Objectives

In order for the *EAQ* scheme to be effective, we require that (i) it should always be possible to improve the accuracy of an approximate version through more iterations of sub-queries, and (ii) it must be viable to reach arbitrary accuracy with limited iterations of sub-queries. In other words, sub-queries of *EAQ* should always ensure *error convergence*, as defined in Definition 2.

Definition 2 (Error Convergence) *Let ρ_i and ρ_j be, respectively, the i^{th} and j^{th} iteration of sub-query ρ for the same data access process. ρ is said to ensure 'error convergence', if (i) the outcome $\mathcal{O}_i$ of ρ_i yields more accurate approximate version than the outcome $\mathcal{O}_j$ of ρ_j when and only when $i > j$, i.e. $i > j \iff \epsilon\left(\mathcal{O}_i\right) < \epsilon\left(\mathcal{O}_j\right)$, and (ii) given any $\varepsilon > 0$, there always exists an integer $K > 0$ such that, for all integers $k > K$, $\epsilon\left(\mathcal{O}_k\right) < \varepsilon$, where $\mathcal{O}_k$ is the outcome of ρ_k in the same data access process.*

On the energy efficiency of *EAQ*, we concern that the accuracy of an approximate version should worth the cost for obtaining it. Namely, the cost for obtaining an approximate version should be bounded, on both ends, by those for obtaining the pair of versions having slightly looser and slightly tighter accuracies. Moreover, as Equation 3 suggests, different iterations of sub-queries should retrieve non-overlapping fractions of the targeted dataset. Formally, sub-queries of *EAQ* should always exhibit *full data reuse*.

Definition 3 (Full Data Reuse) *Let $\mathcal{O}_q = \bigcup_{i=1}^{q} \Delta_i$ be the outcome of the q^{th} iteration ρ_q of sub-query ρ, where $\Delta_i \neq \emptyset$ $(1 \leq i \leq q)$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$ $(1 \leq i, j \leq q)$. The respective sub-query ρ is said to ensure 'full data reuse', if for all $\varepsilon \in \left(\epsilon\left(\mathcal{O}_q\right), \epsilon\left(\mathcal{O}_1\right)\right]$ there exists outcome $\mathcal{O}_b$ $(1 < b \leq q)$ in the same data access process, such that $\varepsilon \in \left(\epsilon\left(\mathcal{O}_b\right), \epsilon\left(\mathcal{O}_{b-1}\right)\right]$.*

In summary, we identify the research objectives of this paper as (i) to realize the two basic operators α and ϵ as specified in Section 2.2, and (ii) to guarantee both *error convergence* and *full data reuse* for sub-queries of *EAQ* according to Definition 2 and 3.

3. DATA SHUFFLING ALGORITHM

3.1 Revisiting Sensor Data

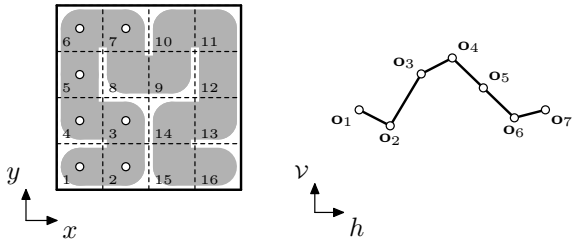
A well-known property of sensor data is the strong spatial correlation [18]. The spatial correlation among sensor data are inclined to be *localized*, i.e. nearby sensors within a few hops, or a distance of tens to hundreds of meters, are likely to yield similar readings, and vise versa. For example, in *intrusion detection* [1], the emergence of an intruder (i.e. human or vehicle) typically only triggers the number of sensors sufficiently close to it. Back in Section 2.1, we presumed a one-to-one mapping between sensor motes' identifiers and coordinates. Now we propose to employ a mapping that implicitly clusters nearby data which potentially exhibit high spatial correlation. In particular, nearby sensor motes (or data items) should be mapped to have closer identifiers (or location stamps), and vise versa. Viable choices for the mapping include various space-filling curves [26], like *Hilbert* curve, *Z*-curve, *Peano* curve, etc. We choose *Hilbert* curve as an example in the following discussions. In practice, the mapping can be computed using the methods in [19]. When the identifiers of sensor motes are assigned according to an l-order *Hilbert* curve, i.e. filling a $2^l \times 2^l$ grid in the network area, certain sized groups of motes with consecutive identifiers always yield λ-localized datasets (Cf. Definition 2.1).

Claim 1 *Let $\mathcal{S}$ be a set of k motes having consecutive identifiers $\mathcal{S} = \{s_u, s_{u+1}, \ldots, s_v\}$. If $k \leq \sum_{i=1}^{\lfloor \log_2 (\lambda \cdot r + 1) \rfloor} 4^{i-1} + 1$, where $\lambda \cdot r = 2^j - 1$ ($j = 1, 2, \ldots, l$) and r is the distance of one hop, then $\mathcal{S}$ always yields λ-localized datasets, otherwise it is not guaranteed. We may denote a dataset yielded by $\mathcal{S}$ with $\mathcal{D}_u^v$, where u and v are respectively the smallest and largest identifiers of sensor motes in $\mathcal{S}$, and the size of dataset $\mathcal{D}_u^v$ is $k = v - u + 1$.*

Proof *The proof of the claim is due to properties of Hilbert curves, and falls beyond the topic of this paper. We thus omit the details.* □

Here after, we default all datasets as being produced according to Claim 1 in which all data items have consecutive location stamps.

In Figure 2(a), we draw the south-west 4×4 grid of the network area, which is filled with a 2-order *Hilbert* curve (i.e. the shadowed stroke). The current snapshot of the network (i.e. $t = t_{now}$) contains a dataset $\mathcal{D} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_7)^\intercal$, where $|\mathcal{D}| = 7$. The data items are sampled and stored respectively by the set of sensor motes $\mathcal{S} = \{s_1, s_2, \ldots, s_7\}$. The scatter plot of data items in Figure 2(b) captures the relation between location stamps h and attribute values $\mathcal{V}$. Note that, for clarity of presentation, Figure 2(b) presumes the attribute values in $\mathcal{V}$ as one dimensional. The techniques to be presented are capable for handling multi-dimensional data.



(a) network grid (b) dataset plot

Figure 2: (a) 4×4 grid in the network area filled by Hilbert curve (the shadowed stroke), white circles represent data items in dataset $\mathcal{D}$; (b) scatter plot of dataset $\mathcal{D}$ in h-$\mathcal{V}$ space, attribute values in $\mathcal{V}$ are presumed one-dimensional for clarity.

We observe two interesting properties in the *canonical form* of datasets (Cf. Section 2.1). First, location stamps establish a *full order* among all data items in the same dataset.

Observation 1 (Order Insensitivity) *Given any permutation of a dataset $\mathcal{D}$, it is always possible to resume the canonical form of $\mathcal{D}$ through sorting data items by the location stamps. With radix sort [21], the resuming completes in $O(k)$ time, where $k = |\mathcal{D}|$.*

Second, it is possible to obtain an approximate version of dataset $\mathcal{D}$ from any subset of $\mathcal{D}$ having two or more data items.

Observation 2 (Partial Recoverability) *Given an arbitrary subset $\mathcal{O}$ of dataset $\mathcal{D}$, where $|\mathcal{O}| \geq 2$, we can obtain an approximate version $\widehat{\mathcal{D}}$ of dataset $\mathcal{D}$ using pairwise linear approximation, i.e. for all $\mathbf{o}_h \in \mathcal{D}$ ($h = h_1, h_2, \ldots, h_k$, $k = |\mathcal{D}|$), we can have $\hat{\mathbf{o}}_h \in \widehat{\mathcal{D}}$,*

$$\hat{\mathbf{o}}_h = \begin{cases} \dfrac{h-v}{u-v}\,\mathbf{o}_u + \dfrac{u-h}{u-v}\,\mathbf{o}_v & \mathbf{o}_h \notin \mathcal{O} \\ \mathbf{o}_h & otherwise \end{cases} \tag{4}$$

in which $\mathbf{o}_u$ and $\mathbf{o}_v$ are two data items in $\mathcal{O}$ having the first and second closest location stamps u and v to h. Where it does not cause ambiguity, we do not distinguish between an approximate version $\widehat{\mathcal{D}}$ and the subset $\mathcal{O} \subseteq \mathcal{D}$ from which $\widehat{\mathcal{D}}$ is constructed.

The approximation error of Equation 4 can be formulated as $\varepsilon_h = \|\hat{\mathbf{o}}_h - \mathbf{o}_h\|_\infty = \max_{j=1}^{d+2} |\hat{\theta}_j - \theta_j|$, where θ_j and $\hat{\theta}_j$ are the j^{th} components in $\mathbf{o}_h$ and $\hat{\mathbf{o}}_h$ respectively. We notice that the contributions of location stamps (θ_1 or $\hat{\theta}_1$) and time stamps (θ_2 or $\hat{\theta}_2$) to the approximation error ε_h are always negligible, namely,

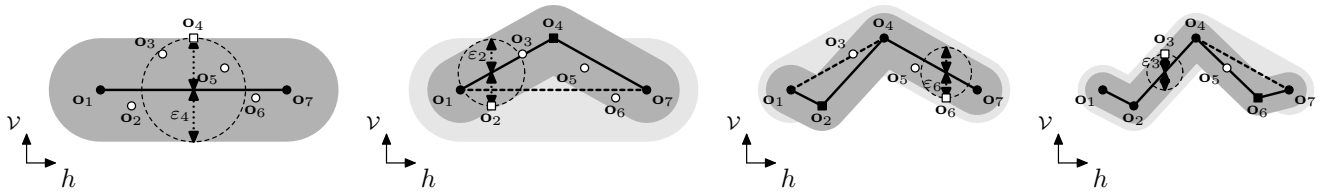
Claim 2 *Let $\mathbf{o}_h = (h, t_h, v_1, \ldots, v_d)^\intercal$ be a data item in dataset $\mathcal{D}$ and $\hat{\mathbf{o}}_h = (\hat{h}, \hat{t}_h, \hat{v}_1, \ldots, \hat{v}_d)^\intercal$ be the approximate version of $\mathbf{o}_h$ built from subset $\mathcal{O}$ of dataset $\mathcal{D}$ using Equation 4. It always holds that $h = \hat{h}$ and $t_h \approx \hat{t}_h$, hence, $\varepsilon_h = \max_{j=1}^{d} |\hat{v}_j - v_j|$. Intuitively, ε_h is equal to the L_∞-norm distance from $\mathbf{o}_h$ to line segment $\overline{\mathbf{o}_u\,\mathbf{o}_v}$ in h-$\mathcal{V}$ space projected to the $\mathcal{V}$-axis(es).*

$$\varepsilon_h = \epsilon_{\mathcal{V}}\big(\mathbf{o}_h, \overline{\mathbf{o}_u\,\mathbf{o}_v}\big) = \max_{j=1}^{d} |\hat{v}_j - v_j| \tag{5}$$

Proof *If $\mathbf{o}_h \in \mathcal{O}$ then $\hat{\mathbf{o}}_h$ and $\mathbf{o}_h$ are always identical, i.e. $\hat{h} = h$ and $\hat{t} = t$, otherwise, $\hat{\mathbf{o}}_h$ is built from $\mathbf{o}_u, \mathbf{o}_v \in \mathcal{O}$, thus $\hat{h} = \frac{h-v}{u-v}\,u + \frac{u-h}{u-v}\,v = h$, and $\hat{t}_h = \frac{h-v}{u-v}\,t_u + \frac{u-h}{u-v}\,t_v$. As such, $\hat{t}_h - t_h = \frac{u}{u-v}\,(t_v - t_h) + \frac{h}{u-v}\,(t_u - t_v) + \frac{v}{u-v}\,(t_h - t_u)$. Note that $\mathbf{o}_u$, $\mathbf{o}_v$, and $\mathbf{o}_h$ are in the same dataset $\mathcal{D}$, hence $|\hat{t}_h - t_h| < \frac{|u|+|v|+|h|}{|u-v|}\,\delta \,/\, T$. Since $T \gg \delta$, $|\hat{t}_h - t_h| \approx 0$, thus $t_h \approx \hat{t}_h$.* □

3.2 Data Shuffling Algorithm

According to Observation 1 and 2, any subset $\mathcal{O}$ of dataset $\mathcal{D}$ having two or more data items can be viewed as an approximate version of $\mathcal{D}$; and by Equation 5, the error of approximation is the maximum of all distances from data items in $\mathcal{D} \setminus \mathcal{O}$ to respective h-$\mathcal{V}$ space line segments $\overline{\mathbf{o}_u\,\mathbf{o}_v}$. This enlightens our thought to *rank* and *reorder* data items in $\mathcal{D}$ according to their importance, or errors incurred if otherwise omitted, to approximating the entire dataset $\mathcal{D}$. In doing so, we can obtain a special permutation $\pi(\mathcal{D})$ of $\mathcal{D}$ in which data items with higher ranks tend to precede those with lower ones. Thus, any prefix of $\pi(\mathcal{D})$ can yield an approximate version of $\mathcal{D}$, and the error of approximation is equal to the rank of the first data item missing from the prefix. We next describe the *data shuffling* procedure for constructing such permutation $\pi(\mathcal{D})$, which we call *multi-version array* or *MVA*.



(a) $\mathcal{O}_2 = \big(\mathbf{o}_1, \mathbf{o}_7\big)^\intercal$ (b) $\mathcal{O}_3 = \big(\mathcal{O}_2^\intercal, \mathbf{o}_4\big)^\intercal$ (c) $\mathcal{O}_4 = \big(\mathcal{O}_3^\intercal, \mathbf{o}_2\big)^\intercal$ (d) $\mathcal{O}_5 = \big(\mathcal{O}_4^\intercal, \mathbf{o}_6\big)^\intercal$

Figure 3: Approximating dataset $\mathcal{D}$ with $\mathcal{O}_2$, $\mathcal{O}_3$, $\mathcal{O}_4$ and $\mathcal{O}_5$ respectively. Black dots are data items included in respective approximate versions; white squares are current farthest data items to be included in subsequent approximate versions; and white circles are the remaining data items in $\mathcal{D} \setminus \mathcal{O}_b$ ($b = 2, 3, 4, 5$). The shadowed background strokes indicate the errors of approximation.

Consider the dataset $\mathcal{D}$ in Figure 2(b) for example. As an initial step, we take the two extreme data items $\mathbf{o}_1$ and $\mathbf{o}_7$ from $\mathcal{D}$ and treat the outcome $\mathcal{O}_2 = \big(\mathbf{o}_1, \mathbf{o}_7\big)^\intercal$ as a rough approximate version. Approximating $\mathcal{D}$ with $\mathcal{O}_2$ introduces an error ε equal to the maximum of all distances from data items in $\mathcal{D} \setminus \mathcal{O}_2$ to the h-$\mathcal{V}$ space line segment $\overline{\mathbf{o}_1\,\mathbf{o}_7}$, i.e. $\varepsilon = \max_{\mathbf{o}_h \in \mathcal{D} \setminus \mathcal{O}_2} \big\{\epsilon_{\mathcal{V}}\big(\mathbf{o}_h, \overline{\mathbf{o}_1\,\mathbf{o}_7}\big)\big\}$. The farthest data item identified for this step is $\mathbf{o}_4$, which can be observed from Figure 3(a). The emergence of $\mathbf{o}_4$ extends previous outcome $\mathcal{O}_2$ to $\mathcal{O}_3 = \big(\mathcal{O}_2^\intercal, \mathbf{o}_4\big)^\intercal$; and it divides the remaining data items in $\mathcal{D} \setminus \mathcal{O}_3$ into two intervals $\mathcal{D}\,(1\!:\!4) = \big\{\mathbf{o}_2, \mathbf{o}_3\big\}$ and $\mathcal{D}\,(4\!:\!7) = \big\{\mathbf{o}_5, \mathbf{o}_6\big\}$. We then separately test data items in $\mathcal{D}\,(1\!:\!4)$ and $\mathcal{D}\,(4\!:\!7)$ for their closeness to respective line segments $\overline{\mathbf{o}_1\,\mathbf{o}_4}$ and $\overline{\mathbf{o}_4\,\mathbf{o}_7}$. Among all non-empty intervals, the one with the globally farthest data item is to be divided next following the exact process as we divide $\mathcal{D}\,(1\!:\!7)$. As Figure 3(b) shows, this step identifies $\mathbf{o}_2$ as the farthest data item; and the respective interval is $\mathcal{D}\,(1\!:\!4)$. $\mathbf{o}_2$ divides $\mathcal{D}\,(1\!:\!4)$ into two further intervals $\mathcal{D}\,(1\!:\!2) = \emptyset$ and $\mathcal{D}\,(2\!:\!4) = \big\{\mathbf{o}_3\big\}$. We repeat until either (i) all data items from $\mathcal{D}$ have been selected, or (ii) the farthest data item yields $\varepsilon = 0$ (i.e. like $\mathbf{o}_5$ from $\overline{\mathbf{o}_4\,\mathbf{o}_6}$ in Figure 3(d)). The final outcome $\mathcal{O}_7 = \big(\mathbf{o}_1, \mathbf{o}_7, \mathbf{o}_4, \mathbf{o}_2, \mathbf{o}_6, \mathbf{o}_3, \mathbf{o}_5\big)^\intercal$ is the *multi-version array* (*MVA*) of dataset $\mathcal{D}$, which we denote $\pi\big(\mathcal{D}\big) = \mathcal{O}_7$.

Algorithm 1 generalizes the above procedure for *data shuffling*. We employ a priority queue Q to manage the precedences of intervals. On line 12, the PUSH(Q, e, w) operation pushes an entity e into Q with rank w. Specifically, an entity e comes out from Q (via the POP operation on line 14) following all entities with higher rank values and possibly some entities with the same rank.

In Figure 4, we use a tree hierarchy to organize the intervals of data items being divided while shuffling dataset $\mathcal{D}$. Algorithm 1 can be viewed as a *best-first* [21] traversal of the binary tree in Figure 4. The itinerary of traversal is depicted as dashed arrows.

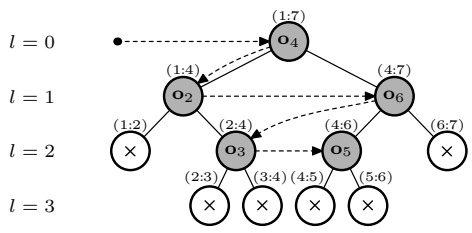


Figure 4: Intervals of data items being divided while shuffling dataset $\mathcal{D}$. Labels on nodes are the farthest data items identified from respective intervals (marked above nodes). Shadowed nodes are data items in $\pi\big(\mathcal{D}\big) \setminus \big\{\mathbf{o}_1, \mathbf{o}_7\big\}$. Dashed arrows indicate the order by which data items are selected.

Algorithm 1 SHUFFLE DATASET TO CONSTRUCT *MVA*

```
1:  procedure FindFarthest(D, p, q)                          ▷ D(p:q) ≠ ∅
2:      ⟨m, ε⟩ ← ⟨null, 0⟩
3:      for each o_i ∈ D (p < i < q) do
4:          if ε_V(o_i, o_p o_q) ≥ ε then
5:              ⟨m, ε⟩ ← ⟨i, ε_V(o_i, o_p o_q)⟩              ▷ Equation 5
6:      return ⟨m, ε⟩
7:  procedure Shuffle(D)                          ▷ D = (o_{h_1}, o_{h_2}, ..., o_{h_k})^T
8:      O ← (o_{h_1}, o_{h_k})^T                                  ▷ k ≥ 2
9:      Q ← ()                                          ▷ Q is a priority queue
10:     if k > 2 then
11:         ⟨m_1, ε_1⟩ ← FindFarthest(D, h_1, h_k)
12:         Push(Q, ⟨h_1, h_k, m_1, ε_1⟩, ε_1)
13:     while not IsEmpty(Q) do
14:         ⟨p, q, m, ε⟩ ← Pop(Q)
15:         O ← (O^T, o_m)^T
16:         if ε = 0 then
17:             break
18:         if exists o_u ∈ D (p < u < m) then          ▷ D(p:m) ≠ ∅
19:             ⟨m_u, ε_u⟩ ← FindFarthest(D, p, m)
20:             Push(Q, ⟨p, m, m_u, ε_u⟩, ε_u)
21:         if exists o_v ∈ D (m < v < q) then          ▷ D(m:q) ≠ ∅
22:             ⟨m_v, ε_v⟩ ← FindFarthest(D, m, q)
23:             Push(Q, ⟨m, q, m_v, ε_v⟩, ε_v)
24:     return O
```

Theorem 1 (Time Complexity) *The time complexity for shuffling a dataset $\mathcal{D}$ of k data items is $O(k^2)$ in worst case.*

Proof *For each level l of the tree in Figure 4, there are at most 2^l data items being added to $\pi\big(\mathcal{D}\big)$. And selecting these data incurs a total of $k - 2 - 2^{l-1}$ comparisons of distance. Thus for the entire process of traversing the tree, a total of $O(H\,k)$ basic operations are performed, where H is the height of the tree. For a binary tree with k nodes, the height H is at least $\lceil \log_2 (k-1) \rceil$ and at most k, therefore the time complexity is $O(k^2)$ in worst case.* □

Theorem 2 (Space Complexity) *The space complexity for shuffling a dataset $\mathcal{D}$ of k data items is $O(k)$.*

Proof *We use a priority queue to manage the non-empty intervals to be added to $\pi\big(\mathcal{D}\big)$. Not considering the two extreme data items, there can be at most $k - 2$ non-empty intervals. Thus the number of entities in the priority queue does not exceed $k - 2$. When the priority queue is implemented using a maximum heap, it requires $O(k)$ space. Thus the space complexity is $O(k)$.* □

3.3 Approximate Data Recovery

According to Algorithm 1, any prefix $\mathcal{O}$ of *MVA* $\pi(\mathcal{D})$ is a subset of the original dataset $\mathcal{D}$. And by Observation 2, if prefix $\mathcal{O}$ contains two or more data items, then it always yields an approximate version $\widehat{\mathcal{D}}$ of the original dataset $\mathcal{D}$. To be precise, for every missing data item $\mathbf{o}_h \in \mathcal{D} \setminus \mathcal{O}$, an approximate counterpart $\hat{\mathbf{o}}_h$ can be recovered from $\mathcal{O}$ using Equation 4. An interesting property of *multi-version array* (*MVA*) is that the L_∞-norm error (Cf. Equation 1) of the approximate version $\widehat{\mathcal{D}}$ recovered from prefix $\mathcal{O}$ is guaranteed to be bounded by the h-$\mathcal{V}$ space distance (Cf. Equation 5) between the first data item $\mathbf{x}$ *not* in $\mathcal{O}$ (i.e. $\mathbf{x}$ is the first data item in the postfix $\pi(\mathcal{D}) \setminus \mathcal{O}$ of *MVA* $\pi(\mathcal{D})$) and the approximate counterpart $\hat{\mathbf{x}}$ built from $\mathcal{O}$ using Equation 4. Formally, we have,

Theorem 3 (Recoverability and Error Bound) *Suppose $\pi(D)$ is 'shuffled' from dataset $\mathcal{D}$ using Algorithm 1, and $|\pi(\mathcal{D})| = k$. Given any prefix $\mathcal{O}$ of $\pi(D)$, where $|\mathcal{O}| = b$ ($2 \leq b < k$), it is guaranteed that prefix $\mathcal{O}$ always yields approximate version $\widehat{\mathcal{D}}$ such that $L_\infty(\widehat{\mathcal{D}}, \mathcal{D}) \leq \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$, where $\mathbf{x}$ is the $(b+1)^{th}$ data item in $\pi(\mathcal{D})$, i.e. $\mathbf{x} = \pi(\mathcal{D})[b+1]$, and $\hat{\mathbf{x}}$ is the approximate counterpart of $\mathbf{x}$ recovered from $\mathcal{O}$ using Equation 4.*

Proof *Dataset recovery is by means of Observation 2. Specifically, for all integer $h \in [h_1, h_k]$, where h_1 and h_k are respectively the location stamps of the first and second data items in prefix $\mathcal{O}$, we recover each approximate data item $\hat{\mathbf{o}}_h$ ($h = h_1, h_1+1, \ldots, h_k$) using Equation 4. If $\mathbf{o}_h \in \mathcal{O}$, i.e. $\mathbf{o}_h = \mathcal{O}[i]$ ($i = 1, 2, \ldots, b$), then Equation 4 ensures $\hat{\mathbf{o}}_h = \mathbf{o}_h$. Thus the approximation error $\|\hat{\mathbf{o}}_h - \mathbf{o}_h\|_\infty = 0$. If $\mathbf{o}_h \in \mathcal{D} \setminus \mathcal{O}$, then Algorithm 1 ensures that the $(b+1)^{th}$ data item $\mathbf{x}$ in $\pi(\mathcal{D})$ is at least one of the farthest from respective h-$\mathcal{V}$ space line segments (otherwise, the priority queue in Algorithm 1 should not have* POP*'ed data item $\mathbf{x}$ before all data items in $\mathcal{D} \setminus \mathcal{O} \setminus \{\mathbf{x}\}$). Thus for all data items $\mathbf{o}_h \in \mathcal{D} \setminus \mathcal{O}$, we have $\|\hat{\mathbf{o}}_h - \mathbf{o}_h\|_\infty \leq \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$. In summary, for all $\mathbf{o}_h \in \mathcal{D}$ and respective $\hat{\mathbf{o}}_h \in \widehat{\mathcal{D}}$, it is guaranteed that $\|\hat{\mathbf{o}}_h - \mathbf{o}_h\|_\infty \leq \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$. In other words, $L_\infty(\widehat{\mathcal{D}}, \mathcal{D}) \leq \|\hat{\mathbf{x}} - \mathbf{x}\|_\infty$.* □

The intuitive idea of Theorem 3 is to always extract *MVA* prefixes of $b \geq 3$ data items. In doing so, we can obtain from the prefix (i) approximate version $\widehat{\mathcal{D}}$ recovered from the initial $b-1$ data items of $\mathcal{O}$, and (ii) L_∞-norm error bound of $\widehat{\mathcal{D}}$ computed using the b^{th} data item of $\mathcal{O}$. Now we are able to realize the two basic operators α and ϵ specified in Section 2.2. Pseudo codes are given in Algorithm 2. The RECOVER($\mathcal{O}$) procedure on line 1 implements operator α, and the ERROR($\mathcal{O}$) procedure on line 8 implements operator ϵ. On lines 5 and 11, APPROX($\mathcal{X}$, h) utilizes Equation 4 to recover an approximate data item $\hat{\mathbf{o}}_h$ from *MVA* prefix $\mathcal{X}$. On line 11, the call to LOCSTAMP($\mathbf{x}$) returns the location stamp of $\mathbf{x}$.

Algorithm 2 RECOVER APPROXIMATE DATASET FROM *MVA*

```
1:  procedure RECOVER(O)                          ▷ O = (o_{h_1}, o_{h_k}, o_{m_1}, ...)^T
2:      D̂ ← ()^T                                  ▷ initialize empty dataset
3:      b ← |O|                                   ▷ b ≥ 3
4:      for h ← h_1 upto h_k do
5:          ô_h ← APPROX(O[1:b−1], h)             ▷ Equation 4
6:          D̂ ← (D̂^T, ô_h)^T
7:      return D̂
8:  procedure ERROR(O)                            ▷ O = (o_{h_1}, o_{h_k}, o_{m_1}, ...)^T
9:      b ← |O|                                   ▷ b ≥ 3
10:     x ← O[b]                                  ▷ extract the b^th data item from O
11:     x̂ ← APPROX(O[1:b−1], LOCSTAMP(x))         ▷ Equation 4
12:     return ||x̂ − x||_∞
```

3.4 Discussions

In reality, raw sensor data may contain *intrinsic error* due to flawed sensors, environmental noises, etc. Our philosophy is that the *data shuffling* algorithm (Cf. Algorithm 1) should treat input datasets as 100% accurate. If in case the intrinsic error in raw sensor data is well understood, the proposed techniques can be even more efficient, for *data shuffling* no longer needs to put all data items into the *multi-version array* (*MVA*), but may terminate when the approximation error drops to a level comparable to the intrinsic error of the input dataset. The existence of outliers [29] in raw sensor data could affect the performance of *MVA*-based data approximation. We allow domain experts to deploy in-network outliers detector which drop abnormal data before *data shuffling*. The *dataset recovery* operator α (Cf. Algorithm 2) automatically estimates dropped data items using available ones. Besides spatial correlation, sensor data may also exhibit temporal and intra-attribute correlations [7, 9]. Our techniques can co-exist with most cutting-edge methods for exploiting such correlations, because *data shuffling* only rearranges the input datasets and the values of individual sensor readings are never changed. Last but not least, *pairwise linear approximation* is not the only means for recovering approximate versions of datasets. Our choice is due to complexity concerns, because the limited computing capability of sensor motes calls for light-weight algorithms for frequent operations.

4. EAQ SUB-QUERY DESIGN

We consider enabling *ϵ-approximate querying* (*EAQ*) based on *multi-version array*s (*MVA*). Recall from Section 2 that *EAQ* depends on iterations of sub-queries to obtain approximate versions of datasets at different accuracy levels. For example, if dataset $\mathcal{D}$ is available as *MVA* $\pi(\mathcal{D})$, sub-query ρ can be called q times to retrieve consecutive fragments $\Delta_1, \Delta_2, \ldots, \Delta_q$ from $\pi(\mathcal{D})$ ($q = 1, 2, \ldots$), and the outcomes $\mathcal{O}_{b_q} = \bigcup_{i=1}^{q} \Delta_i$ are *MVA* prefixes of strictly increasing length, i.e. $|\mathcal{O}_{b_1}| < |\mathcal{O}_{b_2}| < \ldots < |\mathcal{O}_{b_q}|$. We notice that the research objectives in Section 2.3, i.e. to ensure a sub-query ρ satisfy both Definition 2 and Definition 3, are always achieved when the outcomes of sub-query ρ are a series of *MVA* prefixes having monotonically decreasing error bounds.

Claim 3 *Let ρ be an EAQ sub-query, and $\mathcal{P} = \langle \mathcal{O}_{b_1}, \mathcal{O}_{b_2}, \ldots \rangle$ be the full sequence of outcomes, i.e. MVA prefixes, obtained with ρ, where $\mathcal{O}_{b_q} = \bigcup_{i=1}^{q} \Delta_i$ ($q = 1, 2, \ldots$), $\Delta_i \neq \emptyset$ ($i = 1, 2, \ldots, q$). If $\epsilon(\mathcal{O}_{b_q}) > \epsilon(\mathcal{O}_{b_{q+1}})$ for all $q = 1, 2, \ldots$, then sub-query ρ always conforms to both Definition 2 and Definition 3.*

Proof *Suppose the targeted dataset is $\mathcal{D}$ and the respective MVA is $\pi(\mathcal{D})$. For Definition 2, we need to prove that for sufficiently large k, the error bound of the k^{th} outcome $\mathcal{O}_{b_k} = \bigcup_{i=1}^{k} \Delta_i$ in sequence $\mathcal{P}$ can be arbitrarily small. Since $\Delta_q \neq \emptyset$ ($q = 1, 2, \ldots$), the outcome $\mathcal{O}_{b_k}$ is extended upon every iteration of sub-query ρ. Thus, if k is sufficiently large, $\mathcal{O}_{b_k}$ ultimately grows to the length of the entire MVA $\pi(\mathcal{D})$. Obviously, it always holds that $\epsilon(\pi(\mathcal{D})) = 0$. Hence Definition 2 holds for ρ. For Definition 3, $\Delta_k \neq \emptyset$ ensures $|\mathcal{O}_{b_{k+1}}| > |\mathcal{O}_{b_k}|$ ($k = 1, 2, \ldots, q$, $q = 1, 2, \ldots$). By definition of $\mathcal{P}$, $\epsilon(\mathcal{O}_{b_j}) < \epsilon(\mathcal{O}_{b_i})$ if and only if $|\mathcal{O}_{b_j}| > |\mathcal{O}_{b_i}|$, and the series of intervals $(\epsilon(\mathcal{O}_{b_{k+1}}), \epsilon(\mathcal{O}_{b_k})]$ ($k = 1, 2, \ldots, q$) do not overlap. Thus given any $\varepsilon \in (\epsilon(\mathcal{O}_{b_q}), \epsilon(\mathcal{O}_{b_1})]$, it must fall in one and only one of the intervals, say, $(\epsilon(\mathcal{O}_{b_{i+1}}), \epsilon(\mathcal{O}_{b_i})]$ and $1 < |\mathcal{O}_{b_i}| \leq |\mathcal{O}_{b_q}|$. Hence Definition 3 holds for sub-query ρ.* □

Given a sufficiently long *multi-version array* (*MVA*), there may exist more than one sequences of prefixes having monotonically decreasing error bounds. In other words, we can design various *EAQ* sub-queries for use in different scenarios. For example, to support *online analysis* of sensor data across the network, we concern that the *MVA* prefixes obtained by sub-queries should 'grow' smoothly in length. This is to prevent communicating unnecessary details in case coarse datasets already suffice for, say, generating graphical visualizations for human users. Alternatively, a *time critical* query may always prefer to retrieve the shortest *MVA* prefix that yields an acceptable approximate version of targeted data. Following, we present two possible designs of *EAQ* sub-query.

Greedy Sub-Query (ρ_G) Upon the i^{th} iteration, sub-query ρ_G returns the shortest non-empty *MVA* fragment Δ_i such that $\epsilon\big(\mathcal{O}_{b_i}\big) < \epsilon\big(\mathcal{O}_{b_{i-1}}\big)$ where $\mathcal{O}_{b_i} = \mathcal{O}_{b_{i-1}} \cup \Delta_i$, and $\mathcal{O}_{b_{i-1}}$ is the outcome produced by ρ_G for the $(i-1)^{\text{th}}$ iteration.

Smooth Sub-Query (ρ_S) Sub-query ρ_S requires a precomputed array $\mathcal{E}$ that contains the longest monotonically decreasing sub-sequence of error bounds of all targeted *MVA* prefixes, i.e. $\mathcal{E} = \langle \varepsilon_{b_1}, \varepsilon_{b_2}, \ldots \rangle$ where $\varepsilon_{b_q} = \epsilon\big(\mathcal{O}_{b_q}\big)$, $\varepsilon_{b_q} > \varepsilon_{b_{q+1}}$ $(q = 1, 2, \ldots)$. Upon the i^{th} iteration, sub-query ρ_S returns $\Delta_i = \mathcal{O}_{b_i} \setminus \mathcal{O}_{b_{i-1}}$, where $\varepsilon_{b_{i-1}}$ and ε_{b_i} are respectively the $(i-1)^{\text{th}}$ and i^{th} error bounds in array $\mathcal{E}$.

Prototype implementations of sub-queries ρ_G and ρ_S are given in Algorithm 3. On line 13, LONGESTDECRSUBSEQ($\mathcal{A}$) returns the *longest monotonically decreasing sub-sequence* $\mathcal{E}$ for the array $\mathcal{A}$ of tuples $\langle b_i, e_i \rangle$ according to the values of e_i's. The well-known dynamic programming algorithm implements LONGESTDECRSUBSEQ in $O(k^2)$ time for input size $\big|\mathcal{T}\big| = k$, which is beyond the topic of this paper and omitted to reserve space. In Algorithm 3, we use the notation $\pi\big(\mathcal{D}\big)[1:b]$, such as on lines 2, 5, 6, etc., to represent the prefix of $\pi\big(\mathcal{D}\big)$ having b data items.

Algorithm 3 PROTOTYPES OF EAQ SUB-QUERIES

Require: $\pi\big(\mathcal{D}\big) = \big(\mathbf{o}_{h_1}, \mathbf{o}_{h_k}, \mathbf{o}_{m_1}, \mathbf{o}_{m_2}, \ldots, \mathbf{o}_{m_{k-2}}\big)^{\mathsf{T}}$
1: **procedure** GREEDYSUBQUERY(b) ▷ $b = 0$ or $b \geq 3$
2: $\Delta \leftarrow \pi\big(\mathcal{D}\big)[1:3]$
3: **if** $b \geq 3$ **then**
4: **for** $i \leftarrow b+1$ **upto** $\big|\pi\big(\mathcal{D}\big)\big|$ **do**
5: $\mathcal{O} \leftarrow \pi\big(\mathcal{D}\big)[1:i]$
6: **if** $\epsilon\big(\mathcal{O}\big) < \epsilon\big(\pi\big(\mathcal{D}\big)[1:b]\big)$ **then**
7: **break**
8: $\Delta \leftarrow \mathcal{O} \setminus (\pi\big(\mathcal{D}\big)[1:b])$
9: **return** Δ
Require: $\mathcal{E} \leftarrow \langle\rangle$
10: **procedure** SMOOTHSUBQUERY(i) ▷ $i \geq 1$
11: **if** ISEMPTY($\mathcal{E}$) **then**
12: $\mathcal{A} \leftarrow \langle \langle 3, \epsilon\big(\mathcal{O}_3\big)\rangle, \langle 4, \epsilon\big(\mathcal{O}_4\big)\rangle, \ldots, \langle \big|\pi\big(\mathcal{D}\big)\big|, 0\rangle \rangle$
13: $\mathcal{E} \leftarrow$ LONGESTDECRSUBSEQ($\mathcal{A}$)
14: $\langle b_i, \varepsilon_i \rangle \leftarrow \mathcal{E}[i]$
15: $\Delta \leftarrow \mathcal{O}_{b_i} \leftarrow \pi\big(D\big)[1:b_i]$
16: **if** $i > 1$ **then**
17: $\langle b_{i-1}, \varepsilon_{i-1} \rangle \leftarrow \mathcal{E}[i-1]$
18: $\Delta \leftarrow \mathcal{O}_{b_i} \setminus (\pi\big(D\big)[1:b_{i-1}])$
19: **return** Δ

Note in Algorithm 3 that *EAQ* sub-queries ρ_G and ρ_S are designed for iterative execution. We take ρ_G for example. When called for the first time, it yields $\mathcal{O}_b = \rho_G(0) = \pi\big(\mathcal{D}\big)[1:3]$, where $b = \big|\mathcal{O}_b\big|$. We then make a successive call to ρ_G with b as input, and it yields $\Delta_{b'} = \rho_G(b) = \pi\big(\mathcal{D}\big)[b+1:b']$, where $b' > b$. Hence, we obtain outcome $\mathcal{O}_{b'} = \mathcal{O}_b \cup \Delta_{b'}$ and $b' = \big|\mathcal{O}_{b'}\big|$. Making further calls to ρ_G using the size b_q of previous outcome $\mathcal{O}_{b_q}$ as input, we can obtain extended outcome $\mathcal{O}_{b_{q+1}} = \mathcal{O}_{b_q} \cup \Delta_{b_{q+1}}$. The iteration stops when the outcome suffices for users' purpose, or when ultimately some outcome $\mathcal{O}$ yields $\epsilon\big(\mathcal{O}\big) = 0$.

5. EAQ-BASED QUERY PROCESSING

The *ϵ-approximate querying* (*EAQ*) scheme promises efficient and flexible processing of various queries. Following, we present three examples to outline *EAQ*-based query processing.

5.1 Spatial Window Query

A *spatial window* query [32] specifies a two-dimensional rectangular window in the network area. Query results are the data items from all sensor motes in the window.[1] An example is shown in Figure 5, where the spatial window $\mathcal{W}$ is a rectangular region $\mathcal{W} = (0.125, 0.5) - (0.375, 0.6875)$ relative to the network area. Recall from Section 3.1 that geographic locations can be mapped to sensor motes' identifiers using *Hilbert* curve. Hence, the list of sensor motes $\mathcal{S}_W$ in window $\mathcal{W}$ can be resolved at the base station s_0 or at any sensor mote s_h. As Claim 1 suggests (Cf. Section 3.1), sensor motes in $\mathcal{S}_W$ can be divided into special groups $\mathcal{S}_u^v$ of sensor motes that always yield λ-localized datasets. For the example in Figure 5, there exist two such groups, i.e. $\mathcal{S}_W = \mathcal{S}_{69}^{76} \cup \mathcal{S}_{115}^{126}$.

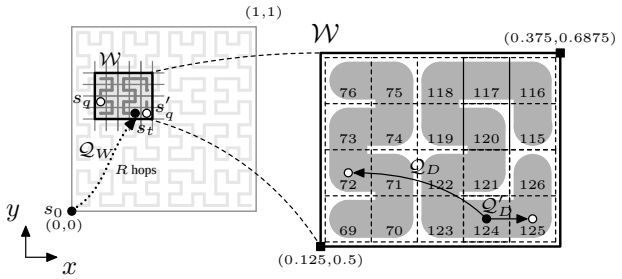


Figure 5: *EAQ*-based window query processing. The query is propagated R-hops from base station s_0 to a randomly picked mote s_t in window $\mathcal{W}$. s_t then invokes separate *EAQ* data access processes for obtaining each dataset stored in $\mathcal{W}$.

We formulate a *spatial window* query $\mathcal{Q}_W$ as $\mathcal{Q}_W = \langle \mathcal{W}, f_P \rangle$, where $\mathcal{W}$ is the spatial window, and f_P is a preference flag. When f_P is toggled *ON*, users can explicitly call *EAQ* sub-queries to refine approximate datasets available at the base station, otherwise, the behavior of $\mathcal{Q}_W$ attempts to emulate that of conventional *ad hoc* window queries [32], i.e. to return desired data items all at once. The processing of *spatial window* queries is as follows.

Step 1: (s_0 initiates $\mathcal{Q}_W$) s_0 randomly selects a sensor mote s_t in window $\mathcal{S}_W$ and propagates $\mathcal{Q}_W$ to s_t. For the example in Figure 5, s_{124} is selected as s_t at this step.

Step 2: (s_t receives $\mathcal{Q}_W = \langle \mathcal{W}, f_P \rangle$) s_t locally resolves the list of motes $\mathcal{S}_W$ from $\mathcal{W}$. For each group $\mathcal{S}_u^v$ of motes, i.e. with consecutive identifiers between u and v, in $\mathcal{S}_W$, s_t randomly picks a mote s_q from $\mathcal{S}_u^v$ and sends $\mathcal{Q}_D = \langle u, v, f_P \rangle$ to mote s_q. As in Figure 5, s_{124} picks $s_{72} \in \mathcal{S}_{69}^{76}$ and $s_{125} \in \mathcal{S}_{115}^{126}$

[1] Spatial window queries can as well return aggregational statistics of data in the window. We focus on the non-aggregate case.

as s_q, and sends two packets $\mathcal{Q}_D = \langle 69, 76, f_P \rangle$ and $\mathcal{Q}'_D = \langle 115, 126, f_P \rangle$ to motes s_{72} and s_{125} respectively.

Step 3: (s_q **receives** $\mathcal{Q}_D = \langle u, v, f_P \rangle$) s_q collects the latest data items from all motes in $\mathcal{S}_u^v$, and stores them in a local dataset $\mathcal{D}$. When data collection completes, s_q converts $\mathcal{D}$ to *MVA* $\pi(\mathcal{D})$ using Algorithm 1. If f_P is toggled *ON*, s_q sends $\Delta_1 = \pi(\mathcal{D})[1:3]$ to s_0, otherwise, s_q invokes sub-query ρ_S to extract consecutive fragments Δ_i ($i = 1, 2, \ldots$) from $\pi(\mathcal{D})$, and sends all fragments to s_0 one after another.

Step 4: (s_0 **receives** Δ_i **from** s_q) s_0 updates previous outcome $\mathcal{O}_{i-1}$ (i.e. may be $\emptyset$) to $\mathcal{O}_i = \mathcal{O}_{i-1} \cup \Delta_i$. If f_P is toggled *ON* and the user is not satisfactory with $\widehat{\mathcal{D}}_i = \alpha(\mathcal{O}_i)$, then s_0 sends $\mathcal{Q}_U = \langle \rho, b \rangle$ to s_q, where $\rho = \rho_G$ and $b = |\mathcal{O}_i|$.

Step 5: (s_q **receives** $\mathcal{Q}_U = \langle \rho, b \rangle$) s_q executes $\rho(b)$ over the *MVA* $\pi(\mathcal{D})$ obtained in *Step 2*, and returns $\Delta = \rho(b)$ to s_0.

When f_P is toggled *OFF*, the above process optimizes for *response time*. Without *EAQ*, query results are available only after the base station has received every data item in all targeted datasets. Hence, in the conventional case, response time $\mathcal{T}(\mathcal{D})$ for obtaining a dataset $\mathcal{D}$ is $\max_{\mathbf{o}_h \in \mathcal{D}} \{\mathcal{T}(\mathbf{o}_h)\}$, where $\mathcal{T}(\mathbf{o}_h)$ is the time sending data item $\mathbf{o}_h$ from s_h to s_0. For *EAQ*-based query processing, an approximate dataset $\widehat{\mathcal{D}}$ is available at base station s_0 once the first *MVA* fragment Δ_1 of $\mathcal{D}$ has arrived. Since Δ_1 only contains 3 data items and can be put into one radio packet, the response time $\mathcal{T}^*(\widehat{\mathcal{D}}) = \mathcal{T}(\Delta_1)$ can be much smaller than $\mathcal{T}(\mathcal{D})$.

When f_P is toggled *ON*, the above process optimizes for *energy efficiency*. In general, the cost of in-network query processing is three-fold (i) the cost for query propagation, (ii) the cost for data processing, and (iii) the cost for result reporting. Without *EAQ*, $\mathcal{Q}_W$ is first sent to any mote in $\mathcal{W}$ and then spread across $\mathcal{W}$. Upon receiving $\mathcal{Q}_W$, each mote directly sends the latest data item to the base station s_0. Hence, in the conventional case, the overall cost is $\mathcal{C}(\mathcal{D}) = O(R) + O(|\mathcal{D}|) + O(R|\mathcal{D}|)$, where $R \gg 1$ is the average number of hops between s_0 and motes in $\mathcal{W}$. Consider *EAQ*-based query processing. *Step 1* and *2* incur $O(R)$ communication. *Step 3* costs $O(|\mathcal{D}|)$ for data collection and $O(R|\Delta_1|) = O(R)$ for sending Δ_1 to s_0. Invoking *Step 4* and *5* for $q-1$ iterations consumes $\sum_{i=2}^{q} O(R|\Delta_i|)$. Thus the overall cost is $\mathcal{C}^*(\widehat{\mathcal{D}}) = O(R) + O(|\mathcal{D}|) + \sum_{i=1}^{q} O(R|\Delta_i|)$. Since $\sum_{i=1}^{q} |\Delta_i| \leq |\pi(D)| \leq |\mathcal{D}|$, it always holds that $\mathcal{C}^*(\widehat{\mathcal{D}}) \leq \mathcal{C}(\mathcal{D})$.

5.2 MVA-Index and Value Range Query

Value range query [6] is a generalized version of *spatial window* query. It extends the two-dimensional *spatial window* $\mathcal{W}$ to a multi-dimensional *range predicate* Γ. Γ can have the same dimensions as data items. For the i^{th} dimension, Γ specifies an interval $[L_i, U_i] \subseteq [0, 1]$ to indicate the range of interest. For example, the spatial window $\mathcal{W}$ in Section 5.1 can be rewritten as a *range predicate* $\Gamma_W = \langle [0.125, 0.375], [0.5, 0.6875], [1 - \frac{\delta}{T}, 1], [0, 1], \ldots, [0, 1] \rangle$. Query results are data items from across the network that fall in the *range predicate* Γ, i.e. data item $\mathbf{o}_h$ is in query results if and only if every component θ_i of $\mathbf{o}_h$ fall in the respective interval $[L_i, U_i]$ of predicate Γ. The major challenge in processing *value range* queries is on how to efficiently locate all candidate data items avoiding exhaustive search or flooding over the entire sensor network.

We notice that *MVA* prefixes of network-stored datasets can be used to form a distributed index, which helps *value range* queries to prune search space while searching data across the network. We next describe the procedure of building the *MVA*-index for datasets stored in-network. An example is shown in Figure 6.

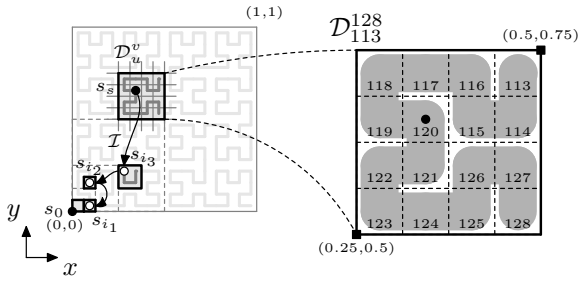


Figure 6: Building *MVA*-index for datasets in-network. The group of motes $\mathcal{S}_u^v$ elect s_s for shuffling the latest dataset $\mathcal{D}_u^v$ and generating an *MVA* prefix $\mathcal{I}$ as index entry. $\mathcal{I}$ is stored, tailored and propagated by consecutive index motes s_{i_3}, s_{i_2}, and s_{i_1} along the route to the base station s_0.

Step 1: (data shuffling) all motes in $\mathcal{S}_u^v$ elect a mote s_s. s_s then collects all data items in $\mathcal{D}_u^v$ to its local storage, and converts them to *MVA* $\pi(\mathcal{D}_u^v)$ using Algorithm 1. For the example in Figure 6, this step elects s_{120} as s_s for shuffling $\mathcal{D}_{113}^{128}$.

Step 2: (s_s **generates index entry** $\mathcal{I}$) s_s invokes *EAQ* sub-query ρ_G to extract a series of fragments $\Delta_1, \Delta_2, \ldots, \Delta_{l+1}$ from $\pi(\mathcal{D}_u^v)$, where l is the order of the *Hilbert* curve filling the network area. *MVA* prefix $\mathcal{I} = \bigcup_{i=1}^{l+1} \Delta_i$ is the index entry to be sent to index mote s_i, where $i = \lfloor \frac{s-1}{4} \rfloor + 1$.

Step 3: (s_i **receives** $\mathcal{I} = \bigcup_{j=1}^{b} \Delta_j$) index mote s_i stores $\mathcal{I}$ in local storage. If $i > 0$ then s_i tailors $\mathcal{I}$ to $\mathcal{I}' = \bigcup_{j=1}^{b-1} \Delta_j$ and sends $\mathcal{I}'$ to next index mote $s_{i'}$, where $i' = \lfloor \frac{i-1}{4} \rfloor + 1$.

The above process propagates an *MVA* prefix $\mathcal{I}$ toward the base station. At each index mote s_{i_j} ($j = l, l-1, \ldots, 1$), the *MVA* prefix is stored locally and tailored to a shorter prefix $\mathcal{I}'$. The shortest *MVA* prefix finally arrives at the base station s_0. When index entries are built for all datasets in the sensor network, it is guaranteed that (i) s_0 always stores the shortest *MVA* prefixes of all datasets, and (ii) the $(i+1)^{\text{th}}$ shortest *MVA* prefixes of any dataset can be found among the groups of sensor motes $\mathcal{S}_1^{4^{i-1}}$ ($i = 1, 2, \ldots, l$), where l is the order of the *Hilbert* curve filling the network grid.

Let $\mathcal{Q}_\Gamma = \langle \Gamma \rangle$ be a *value range* query, where Γ is the range predicate. The processing of $\mathcal{Q}_\Gamma$ takes the reversed route of building *MVA*-index. At base station s_0, $\mathcal{Q}_\Gamma$ first examines the shortest *MVA* prefixes of all datasets in the entire network, and then lists those possibly containing desired data. For each candidate dataset $\mathcal{D}_u^v$, a packet $\mathcal{Q}_\Gamma = \langle \Gamma, u, v \rangle$ is forwarded along the series of index motes s_{i_j} of dataset $\mathcal{D}_u^v$, where $i_j = \lfloor \frac{u-1}{4^j} \rfloor + 1$ ($j = l, l-1, \ldots, 1$). At index motes, $\mathcal{Q}_\Gamma$ can examine the approximate datasets recovered from respective index entries. Chances are, at a certain index mote, the approximate dataset $\widehat{\mathcal{D}}_u^v = \alpha(\mathcal{I})$ can be sufficiently accurate for $\mathcal{Q}_\Gamma$ to determine whether $\mathcal{D}_u^v$ contains desired data items or not. If it does, accurate data items desired can be fetched directly from where $\mathcal{D}_u^v$ is stored, otherwise, $\mathcal{Q}_\Gamma$ can return $\emptyset$ instantly.

5.3 Queries with QoS Constraints

Many queries in sensor networks need to specify *requirements* or *preferences* on how they should be processed. For example, *time critical* applications may specify a deadline by which query results must be available at the base station, and *long running* applications may, due to sustainability concerns, require a query not to consume more energy than a preallocated *budget*, i.e. maximum number of wireless radio packets. Conventionally, sensor networks

have to deploy special infrastructures, such as real time communication protocols, to support queries with *QoS* constraints. For *EAQ*-based query processing, the iterative data access process makes it possible for various queries to check application-specific *QoS* constraints between sub-queries. For example, a *time critical* window query can always choose to obtain coarse yet instant results, and gradually refines them as the deadline approaches. Likewise, the *energy budget* of queries can also be allocated to sub-queries, so that *just-affordable* results are available when the budget runs out. In this regard, *EAQ* offers a uniform framework for processing various queries with virtually any potential *QoS* constraints.

6. EXPERIMENTAL EVALUATION

We deploy a real sensor network testbed to evaluate the effectiveness and efficiency of proposed techniques. The evaluation consists of two aspects of studies (i) the approximation errors of *MVA*, and (ii) the energy consumption of *EAQ*.

6.1 Experiment Settings

The testbed for experimental evaluation is an indoor sensor network consisting of 40 TelosB[2] sensor motes. The query processing system is built on TinyOS 2.1.0 [22]. The data items being collected contain PAR (Photo-synthetically Active Radiation) light, temperature, and voltage sensor readings. Placement of sensor motes is shown in Figure 7. Of all 40 sensor motes, 36 are deployed in the area of interest, *one* is reserved for base station, and the remaining three compose a routing path connecting the network area and the base station. Identifiers of sensor motes are assigned according to a 4-order *Hilbert* curve, i.e. filling a 16×16 grid whose south-west corner overlaps the network area. As can be observed in Figure 7, the network contains three groups of motes having consecutive identifiers $\mathcal{S}_1^{20}$, $\mathcal{S}_{29}^{36}$, and $\mathcal{S}_{53}^{60}$.

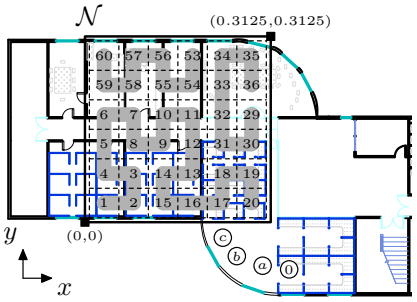


Figure 7: Placement of sensor motes in the testbed. The network area $\mathcal{N}$ is the south-west corner of a 16×16 grid filled with a 4-order Hilbert curve. There are 36 sensor motes in the network area, divided into three groups $\mathcal{S}_1^{20}$, $\mathcal{S}_{29}^{36}$, and $\mathcal{S}_{53}^{60}$. The circles are the base station 0 and three routers a, b, and c.

6.2 Implementation Details

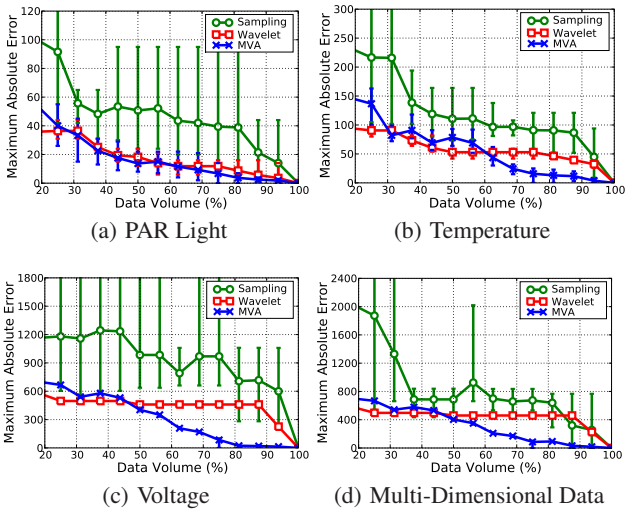
Due to energy-efficiency concerns, the actual structure of data items is not exactly the same as formulated in Section 2.1. First, time stamps are in radio packets, not in individual data items. Second, sensor readings are in raw ADC output format, which are 16-bit unsigned integers, rather than normalized real numbers. The workload contains *spatial window* queries having four types of window sizes 2×4, 4×2, 4×4 and 6×6. At the base station, the query processing system decomposes all window queries into standalone data access processes over individual datasets. In particular, datasets yielded by S_1^8, S_9^{16}, S_1^{16}, S_{13}^{20}, S_{29}^{36}, and S_{53}^{60} are queried by the workload, which can have two sizes 8 and 16. For logging communication statistics, each sensor mote locally maintains a set of counters on radio packets sending and receiving. The statistics are collected before, during and after processing any query. The cost for obtaining statistics is not considered. Because TelosB motes do not have positioning devices, we manually place them according to the *Hilbert* curve in Figure 7. Communication is done via a reliable routing protocol that *retransmits* data upon failure, which is developed from *directed diffusion* [17].

[2]Crossbow Technology: TelosB. http://www.xbow.com/

6.3 Study of Approximation Error

Concerning the well-known property that sensor data have strong spatial correlation [18], we want to evaluate how well the correlation can be exploited from *MVA* prefixes of various lengths. Note that *MVA* is primarily used for building multiple approximate versions of data incrementally. Hence, the evaluation focuses more on the decreasing trend of errors, rather than on error values at particular data volumes. *Haar wavelet* [4] and *sampling* are chosen as counterparts, because they both support incremental access of network-stored data. Sampling in itself does not support recovery of the entire dataset, we thus employ *pairwise linear approximation* to build approximate datasets from sampled data items. The error metric is L_∞-norm, as formulated in Equation 1.

Figure 8 studies the relation between approximation error and the percentage of data used for building the approximate version. We consider different attribute types. Figure 8(a), 8(b), and 8(c) capture the cases where data items are tuples of the form $\mathbf{o} = (h, v)^\intercal$, where v is either PAR light, temperature or voltage. Figure 8(d) is for multi-dimensional data of the form $\mathbf{o} = (h, v_1, v_2, v_3)^\intercal$, where v_1 is PAR light, v_2 is temperature, and v_3 is voltage. As



(a) PAR Light (b) Temperature

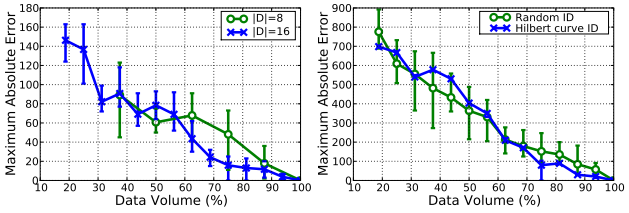
(c) Voltage (d) Multi-Dimensional Data

Figure 8: Approximation errors of *MVA* for (a) PAR light, (b) temperature, (c) voltage, and (d) multi-dimensional data, compared with Haar wavelet and sampling.

Figure 8 shows, the approximation errors of *MVA* almost always decrease smoothly as more data items are being used. The data reduction rates of *MVA*-based approximation are either comparable to, or better than, *Haar wavelet* for all four types of data, and both outperform *sampling* by large factors.

There are several factors impacting the performance of *MVA*-based data approximation. We study the impacts of *input dataset size* and *identifiers assignment* for sensor motes. For the former,

we compare the performance of *MVA* for approximating datasets of sizes 8 and 16. For the latter, we study the performance of *MVA* when the identifiers of motes are assigned randomly, and compare with the case when *Hilbert* curve is used. The impacts of the two factors are depicted in Figure 9(a) and 9(b) respectively. Figure 9(a)



(a) Dataset size (b) Identifiers Assignment

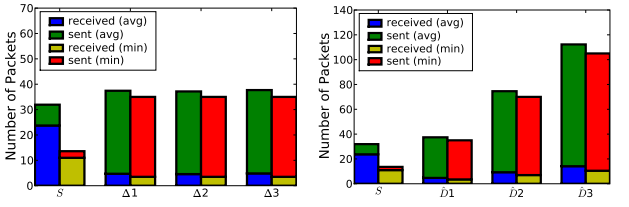
Figure 9: Factors impacting the approximation errors of *MVA*: (a) input dataset size, (b) assignment of motes' identifiers.

shows that the approximation errors of smaller datasets are suffering larger deviations from the mean. Figure 9(b) shows that randomly assigning motes' identifiers also causes the approximation errors to deviate heavily. In both cases, the decreasing trend of the approximation errors of *MVA* becomes unstable.

6.4 Study of Energy Consumption

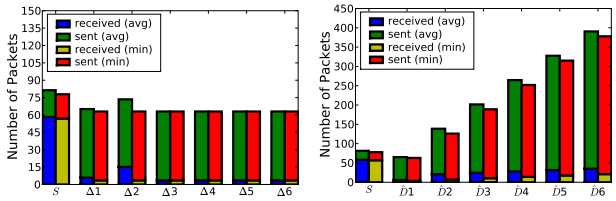
We study the energy consumption for obtaining approximate versions of datasets using *EAQ*. The study considers multi-dimensional data of the form $\mathbf{o} = (h, v_1, v_2, v_3)^\intercal$, where h is the location stamp, v_1 is for PAR light, v_2 is for temperature, and v_3 is for voltage. Targeted datasets are stored respectively at six groups of sensor motes, i.e. $\mathcal{S}_1^8$, $\mathcal{S}_9^{16}$, $\mathcal{S}_{13}^{20}$, $\mathcal{S}_{29}^{36}$, $\mathcal{S}_{53}^{60}$ and $\mathcal{S}_1^{16}$. Datasets are *shuffled* to *MVA* when being accessed for the first time. As Section 6.2 mentions, datasets are of two sizes 8 and 16. A radio packet can hold up to 3 data items, thus datasets of size 8 can have 3 *MVA* fragments, and those of size 16 can have 6 fragments.

The same dataset is being retrieved in an average of 10 data access processes. Within each data access process, we employ the sub-query ρ_S (Cf. Section 4) to extract fragments from the *MVA* of respective dataset. *MVA* fragments are forwarded about 4 hops before arriving at the base station, which can be much larger in some applications. Statistics about packets sending and receiving are collected for *data shuffling* and for each iteration of ρ_S. Figure 10(a) shows the cost for obtaining *MVA* fragments Δ_1, Δ_2, Δ_3 from datasets of size 8. Figure 10(b) studies the accumulated costs for obtaining approximate datasets $\widehat{\mathcal{D}}_q = \alpha\left(\bigcup_{i=1}^{q}\Delta_i\right)$ $(q = 1, 2, 3)$ from scratch. As Figure 10 shows, the refinement costs are almost constant, and the accumulated costs for obtaining approximate versions are proportional to the total numbers of refinements invoked. Moreover, the amortized cost of *data shuffling* is comparable to that of one refinement. The numbers of packets sent during refinements are considerably large, because the communication protocol retransmits packets whenever failure happens. A similar study is conducted for datasets of size 16, as Figure 11 shows.



(a) Refinement Cost (b) Accumulated Cost

Figure 10: Costs for obtaining datasets of size 8. (a) is the cost of refinements, (b) is the accumulated cost for obtaining approximate versions. $\bar{S}$ is the amortized cost of data shuffling.



(a) Refinement Cost (b) Accumulated Cost

Figure 11: Costs for obtaining datasets of size 16. (a) is the cost of refinements, (b) is the accumulated cost for obtaining approximate versions. $\bar{S}$ is the amortized cost of data shuffling.

We further study the average energy consumption of individual motes for processing one query. Figure 12 collects the long-running *per-query* energy consumption of each mote in the group $\mathcal{S}_1^{16}$. The study in Figure 12 shows that the energy consumption at the majority of sensor motes are balanced, with the exceptions of motes s_7, s_8, s_{15}, and s_{16}. The four motes s_7, s_8, s_{15}, and s_{16} receive more

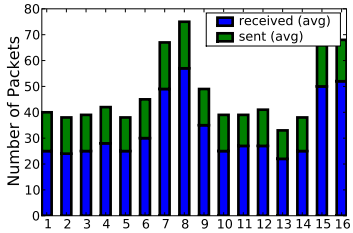


Figure 12: Average energy consumption of individual sensor motes in group $\mathcal{S}_1^{16}$ for processing one query.

data items than others. This is because, during the experiment, the query processing system had explicitly required these motes to receive data for an unrelated task.

6.5 Summary

Through experiments, we study the approximation error of *MVA* and the energy consumption of *EAQ*. For the former, the experiment evaluates the effectiveness of *MVA* for exploiting correlations from different types of sensor data, as well as the impacting factors for *MVA*-based data approximation. For the latter, experiment results show that the energy consumption of data access using *EAQ* is proportional to the number of refinements invoked. Moreover, the study on the energy balance of sensor motes shows that the long-running energy consumption of *EAQ* is balanced. Due to page limitations, we are unable to show results about *EAQ*-based *value range* query processing. The focus of evaluation is on the cost of building *MVA*-index, and on how well the search space of *value range* queries can be pruned using the *MVA*-index.

7. RELATED WORK

To extend the lifetime of battery-powered sensor networks, a large number of previous work, as [28] surveys, suggest *not* to exhaustively retrieve raw sensor data at full-resolution. Instead,

information are being retrieved from sensor networks in one of three forms, (i) *events*, (ii) *aggregational statistics*, and (iii) *(non-aggregate) approximations*. Generating *events* in sensor networks typically requires sophisticated domain knowledge to be embedded in sensor motes, e.g. *to infer 'intrusion' events from 'vibration' and 'infrared motion' sensors* [1]. On the contrary, *aggregations* and *approximations* are meant for general-purpose applications.

Aggregation combines raw sensor readings into numerical statistics as data are forwarded via a tree-like topology toward the base station. Earlier work [25] enables aggregational queries in sensor networks as a service. In [8], the idea of approximation is incorporated in the processing of continuous aggregational queries, i.e. to filter unnecessary updates while ensuring a fixed error bound. [27] takes a further step to support (ε, δ)-approximate aggregation, where users can specify any error bound ε and confidence δ. The drawback shared among all aggregational queries is that numerical statistics only tell the overall condition about a region, and are unable to provide detailed views of data.

Approximation methods for sensor data are diversified. The common principle is to exploit the inherited correlations in sensor data while ensuring a prescribed error bound, or a high probability to satisfy the error bound. [20] exploits temporal correlation from the signals produced by a single sensor. SBR [7] can approximate streams produced by multiple sensors on a single mote using *base signals*, which exploits both temporal and intra-attribute correlations. GAMPS [12] groups multiple sensor motes so that signals within each group can be compressed jointly. BBQ [10] uses statistical models to reduce the sensing and communication cost for processing queries with user-specified error intervals and confidence bounds. Ken [5] uses a pair of probabilistic models, one at the base station and the other at sensor motes in-network, to reduce communication cost for continuously obtaining approximate data from the sensor network. These existing techniques cannot be used to support *EAQ*, because they are designed for *fixed-error* data approximation, and are unable to reduce repetitive information among multiple versions of approximate data. Transform domain methods [2], such as *Discrete Fourier Transform* (DFT), *Discrete Cosine Transform* (DCT) and Wavelet, can produce approximate data at multiple accuracy levels. Unfortunately, DFT and DCT only ensure L_2-norm error bounds [12], where individual sensor readings may deviate arbitrarily far from their accurate values. Wavelet can be used in a way to minimize L_∞-norm error [13]. But, unlike when used for minimizing L_2-norm [2], the accumulated cost for obtaining q approximate versions, i.e. q sets of wavelet coefficients, could be greater than that for obtaining any of the q versions alone. Sampling methods can be used to incrementally obtain data from sensor networks [24], as the proposed *EAQ* scheme does. However, the full set of data usually cannot be recovered from samples with bounded L_∞-norm error. Contour map query [3] is also an important approximation method for sensor data, but it is a standalone query type, and previously obtained contour maps usually cannot be used by other types of queries to produce sound results.

Choosing L_∞-norm as error metric can increase the utility of approximate data [20]. If an approximate dataset has L_∞-norm error bound, then the maximum deviation from all approximate data items to respective accurate values is bounded. This guarantees that many important queries (such as *MAX* and *TOP-K*) and useful mining tasks [12] can yield sound results with bounded errors. [20] proves that L_∞-norm is an upper bound of L_2 and L_1, namely, both L_2 and L_1 norms are bounded if L_∞-norm is bounded.

EAQ allows users or query executors to incrementally 'refine' obtained approximate data to reach arbitrary accuracy. In a way, this can be viewed as a non-aggregate variation of *online aggregation* [16] for distributed environments. *EAQ* enables queries to produce iterative feedbacks that guide their execution, as the ones with *QoS* constraints do. This resembles the spirit of *adaptive query processing* [15] for traditional DBMS. To the best of our knowledge, there is no prior work of this catalog aiming at energy-efficiency in sensor networks, or other distributed environments.

The idea of shuffling sensor datasets is motivated by the *DP line simplification* algorithm named after *Douglas* and *Peucker* [11]. Nevertheless, the proposed *data shuffling* algorithm exhibits three major differences from the *DP* algorithm. First, the *DP* algorithm lossily compresses a polygonal line to meet a predetermined error bound, while our *data shuffling* algorithm derives an equivalent representation *MVA* of the input dataset. Second, *DP* follows a divide-and-conquer process to select most isolated vertex separately for different segments of the polygonal line, whereas *data shuffling* establishes a full order among all data items. Third, *DP* utilizes *Euclidean* distance to identify the most isolated vertex, whereas *data shuffling* resorts to L_∞-norm.

Hilbert curve [19] assigns identifiers to sensor motes, which implicitly clusters sensor data with high correlations. Besides *Hilbert* curve, there exist other alternatives, such as *Z*-curve, *Peano* curve, etc., that may work reasonably well for the purpose of serializing sensor motes [26]. These tools are considered part of the network infrastructure, like communication protocols, and are transparent to the main techniques proposed in this paper.

We employ *spatial window* query to introduce *EAQ*-based query processing. [32] presents techniques to traverse the query window without assuming underlying infrastructures. Our *MVA*-based index in Section 5.2 can answer both accurate and approximate *value range* queries with a uniform infrastructure, whereas previous work [23] and [6] can only support accurate *value range* queries using respective index structures called DIM and Pool.

8. CONCLUSIONS

We propose an arbitrary-accuracy data access scheme *EAQ* underlying various queries in sensor networks. Unlike previous methods, *EAQ* does not require users or query executors to specify error bounds *a prior*, yet it can provide, through runtime refinements, multiple versions of approximate data with guaranteed L_∞-norm error bounds. *EAQ* promises efficient and flexible processing of various queries in sensor networks, including, but not limited to, *spatial window* query, *value range* query and queries with *QoS* constraints, like *response deadline* and *energy budget*. We enable *EAQ* with a novel *data shuffling* algorithm. The algorithm converts sensor datasets into special representations *MVA*. From prefixes of an *MVA*, we can always recover approximate versions of the entire dataset, where individual data items have guaranteed error bounds. We deploy a real sensor network testbed for experimental evaluation. Experiment results show that (i) *MVA* can effectively exploit correlations from sensor datasets at various accuracy levels, and (ii) *EAQ* can efficiently refine obtained approximate data to reach arbitrary accuracy, and the accumulated cost is proportional to the number of refinements invoked. For future work, there are many more data services, in sensor networks and in other data management systems, that may be enabled or enhanced by the idea of *EAQ*. We are particularly interested in enabling *online analytical processing (OLAP)* for communication-constrained environments.

9. REFERENCES

[1] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridharan, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang,

M. Nesterenko, R. Shah, S. S. Kulkarni, M. Aramugam, L. Wang, M. G. Gouda, Y. ri Choi, D. E. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. ExScal: Elements of an extreme scale wireless sensor network. In *IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 102–108, 2005.

[2] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *IEEE Data Engineering Bulletin*, 20(4):3–45, 1997.

[3] C. Buragohain, S. Gandhi, J. Hershberger, and S. Suri. Contour approximation in sensor networks. In *Intl Conf. on Distributed Computing in Sensor Systems (DCOSS)*, pages 356–371, 2006.

[4] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 111–122, 2000.

[5] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, page 48, 2006.

[6] Y.-C. Chung, I.-F. Su, and C. Lee. Supporting multi-dimensional range query for sensor networks. In *Intl. Conf. on Distributed Computing Systems (ICDCS)*, page 35, 2007.

[7] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing historical information in sensor networks. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 527–538, 2004.

[8] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical in-network data aggregation with quality guarantees. In *Intl. Conf. on Extending Database Technology (EDBT)*, pages 658–675, 2004.

[9] A. Deshpande, C. Guestrin, W. Hong, and S. Madden. Exploiting correlated attributes in acquisitional query processing. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, pages 143–154, 2005.

[10] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 588–599, 2004.

[11] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[12] S. Gandhi, S. Nath, S. Suri, and J. Liu. GAMPS: Compressing multi sensor data by grouping and amplitude scaling. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 171–182, 2009.

[13] M. N. Garofalakis and A. Kumar. Wavelet synopses for general error metrics. *ACM Transactions on Database Systems*, 30(4):888–928, 2005.

[14] C. Guestrin, P. Bodi, R. Thibau, M. Paski, and S. Madde. Distributed regression: an efficient framework for modeling sensor network data. In *Information Processing in Sensor Networks (IPSN)*, pages 1–10, 2004.

[15] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering Bulletin*, 23(2):7–18, 2000.

[16] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 171–182, 1997.

[17] C. Intanagonwiwat, R. Govindan, D. Estrin, J. S. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, 2003.

[18] A. Jindal and K. Psounis. Modeling spatially correlated data in sensor networks. *ACM Transactions on Sensor Networks*, 2(4):466–499, 2006.

[19] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24, 2001.

[20] I. Lazaridis and S. Mehrotra. Capturing sensor-generated time series with quality guarantees. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, page 429, 2003.

[21] R. C. T. Lee, S. S. Tseng, R. C. Chang, and Y. T. Tsai. *Introduction to the Design and Analysis of Algorithms: A Strategic Approach*. McGraw-Hill, 2005.

[22] P. Levis, D. Gay, V. Handziski, J. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, et al. T2: A second generation OS for embedded sensor networks. Technical report, University of California, Berkeley, 2005.

[23] X. Li, Y.-J. Kim, R. Govindan, and W. Hong. Multi-dimensional range queries in sensor networks. In *ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, pages 63–75, 2003.

[24] S. Lin, B. Arai, D. Gunopulos, and G. Das. Region sampling: Continuous adaptive sampling on sensor networks. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, page 794, 2008.

[25] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[26] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[27] C. Siyao and J. Li. Sampling based (ε, δ)-approximate aggregation algorithm for sensor networks. In *Intl. Conf. on Distributed Computing Systems (ICDCS)*, 2009.

[28] A. Skordylis, N. Trigoni, and A. Guitton. A study of approximate data management techniques for sensor networks. In *Intl. Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 1–12, 2006.

[29] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Intl. Conf. on Very Large Data Bases (VLDB)*, pages 187–198, 2006.

[30] G. Tolle, J. Polastre, R. Szewczyk, D. E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, pages 51–63, 2005.

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

[32] Y. Xu, W.-C. Lee, J. Xu, and G. Mitchell. Processing window queries in wireless sensor networks. In *IEEE Intl. Conf. on Data Engineering (ICDE)*, page 70, 2006.