MINT Views: Materialized In-Network Top-K Views in Sensor Networks*

Demetrios Zeinalipour-Yazti, Panayiotis Andreou, Panos K. Chrysanthis‡, George Samaras

Department of Computer Science, University of Cyprus, P.O. Box 20537, 1678 Nicosia, Cyprus
‡ Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
{dzeina,cs98ap1}@cs.ucy.ac.cy, panos@cs.pitt.edu, cssamara@cs.ucy.ac.cy

ABSTRACT

In this paper we present MINT (*Materialized In-Network Top-k*) Views, a novel framework for optimizing the execution of continuous monitoring queries in sensor networks. A typical materialized view V maintains the complete results of a query Q in order to minimize the cost of future query executions. In a sensor network context, maintaining consistency between V and the underlying and distributed base relation R is very expensive in terms of communication. Thus, our approach focuses on a subset $V'(\subseteq V)$ that unveils only the k highest-ranked answers at the sink for some user defined parameter k. We additionally provide an elaborate description of energy-conscious algorithms for constructing, pruning and maintaining such recursively-defined in-network views. Our trace-driven experimentation with real datasets show that MINT offers significant energy reductions compared to other predominant data acquisition models.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]:

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

View Management, Top-k Query Processing, In-Network Aggregation, Sensor Networks

1. INTRODUCTION

The improvements in hardware design along with the wide availability of economically viable embedded sensor systems make it feasible today to interact and understand the physical world at an extremely high fidelity [29, 17, 21]. The applications of sensor networks range from environmental monitoring (such as atmosphere and habitant monitoring [29, 25]) to seismic and structural monitoring as well as industry manufacturing [8, 21]. One of the key challenges in this new era of sensor networks is the retrieval of sensor readings using energy-aware algorithms.

In traditional data acquisition techniques [20, 30, 17], the sensor data is transmitted to the *sink* (also denoted as *base station* or *querying node*) immediately after it is acquired from the physical world. Although in-network aggregation significantly reduces the consumption of energy, the oblivious transmission of all query results from all sensors at every acquisition round is still the most energy demanding factor in such environments [29, 36, 25, 38].

In this paper we model the retrieval of data on the presumption that the user is only interested in the k highest ranked answers rather than all of them. As an example consider the case of first responders to a big building fire, attempting to identify the k-most hot epicenters. We propose MINT Views, a novel framework to minimize messaging and thus energy consumption in the execution of continuous monitoring queries. Like other frameworks, we support single-relation queries with the standard aggregate functions but our focus is to optimize top-k queries over *multi-tuple* answers. Such answers are very typical for queries with a `GROUP-BY` clause or for non-aggregate queries.

A view V in relational databases is a virtual table that contains the results from an arbitrary query Q which is evaluated every time V is referred to. In order to avoid the unnecessary re-execution of Q it is beneficial to store V on secondary storage. This introduces the notion of a *materialized view* (referred to as *view* hereafter).

Views have a clear *space* versus *time* tradeoff: A *fully* materialized view V requires more space but also less time in evaluating Q, whereas a *partially* materialized view V' requires less space but also more time in evaluating Q. Materialized views can potentially conserve energy as the application can avoid the expensive re-evaluation of the in-network query Q by utilizing the precomputed results.

Materialized views have been studied in numerous seminal papers including [3, 7, 6, 18]. Although a fully materialized view V maintains the complete results of a query Q, the distributed nature of a sensor network environment, along with its distinct characteristics, imposes some fundamental limitations to this model, summarized as follows:

*This is an enhanced version of our work in [39].

HDMS'07, July 5–6, 2007, Athens, Greece.

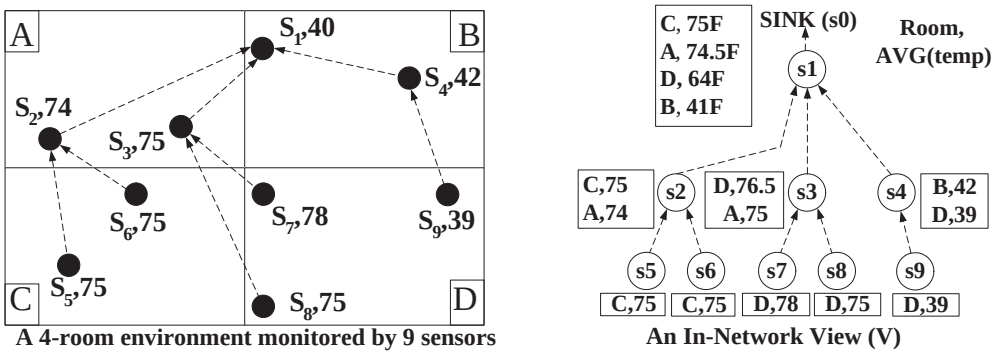

Figure 1: The left figure illustrates a sensor network scenario that consists of 9 sensors $\{s_1, ..., s_9\}$ deployed in four rooms $\{A, B, C, D\}$. The label next to each sensor denotes the identifier of the node and the local temperature reading. The right figure presents a recursively defined In-Network View (V) to query 2. The label next to each node indicate the local averages for each room.

i. Firstly, maintaining consistency between V and the underlying and distributed base relation R (defined by the sensor readings) is very expensive in terms of energy. Thus, we focus on maintaining a subset $V'(\subseteq V)$ that unveils only the k highest-ranked answers for some user defined k; and

ii. Secondly, V' is recursively defined using the results that are stored at the lower-levels of the multi-hop routing tree which interconnects the sink with the sensing devices. Thus, traditional view maintenance techniques are not directly applicable.

To facilitate our description, consider the scenario in Figure 1, where we illustrate a deployment of 9 sensors in a 4-room building. Although our scenario applies to any type of query that yields multi-tuple answers, e.g. *"find the temperature for each sensor every one minute"* (Query 1) and *"find the average temperature of each room every one minute"* (Query 2), our description will focus on Query 2 for the remainder of this paper.

```
Query 1
SELECT sensorID, temp
FROM sensors
EPOCH DURATION 1 min

Query 2
SELECT roomno, AVERAGE(temp)
FROM sensors
GROUP BY roomno
EPOCH DURATION 1 min
```

To answer such queries with the predominant TAG-based [21, 20] in-network aggregation approach each node has to forward tuples of the form (room,sum,count) to its parent every single time instance[1]. One alternative approach is the notion of an *In-Network View (V)* (Figure 1 on the right). V materializes the result of Q and utilizes these results to speedup the next execution of Q. The performance of V largely relies on the premise of temporal coherence between consecutively acquired sensor readings as local changes will affect the intermediate views until the sink.

To improve the performance penalty of In-Network Views, we propose to prune the local views stored at each node and focus on the k highest-ranked answers rather than all of them. This turns out to be extremely useful because now sensors can discard view updates that do not refer to k highest-ranked answers. On the other hand, this also imposes an extremely challenging problem: *"a naive local greedy pruning strategy may easily discard tuples that will be finally among the k highest-ranked answers"*.

To understand this problem, consider again Query 2 but assume that we are only interested in the top-1 result. Such a query should return room $(C, 75F)$. Assuming that each node naively eliminates anything below its local top-1 result will lead us to the erroneous answer $(D, 76.5F)$. In particular, the leaves $\{s_5, s_6, s_7, s_8, s_9\}$ will send their only tuple to their respective parent. The parents $\{s_2, s_3, s_4\}$ will then aggregate the results of their children along with their own result and forward this result to their own parent (i.e., s_1). In particular, s_2 will send $(C, 75F)$, s_3 the tuple $(D, 76.5F)$ and s_4 the tuple $(B, 42F)$. It is now easy to see that if s_1 aggregates the results of its children $\{s_2, s_3, s_4\}$ along with its own result $(B, 40F)$, then this will yield V_0^{wrong}=$\{(D, 76.5F)$, $(C, 75F)$, $(B, 41F)\}$, where room D is the top-1 answer rather than room C.

Our MINT algorithm utilizes an intelligent upper bounding algorithm and a local parameter k to construct a subset of V, denoted as the *k-covered bound-set* V', to be materialized. We will show that any tuple outside V' can safely be eliminated during the execution of a query because this tuple cannot be among the k highest-ranked results.

The key idea of the MINT pruning algorithm is to exploit a set of $|\gamma|$ descriptors ($\gamma = \{\gamma_1, \gamma_2, ...\}$) in order to bound above the score of tuples that are not known at a given level of the sensor network. The elements in γ are application specific: these can either be known in advance, or these can be defined prior to setting up the execution of a query. In our discussion and experimentation, we will utilize the following instances: γ_1 = *"Maximum possible temperature value"* and γ_2 = *"Number of sensors in each room"*. For instance, the temperature sensor on the Mica Weather Board [29] might only record values between -40F to 250F and the baromet-

[1]For clarity in Figure 1, we only depict the average (i.e., sum/count).

ric pressure module can only measure pressure in the range 300mb to 1100mb. Thus, γ_1 can be known in advance. Additionally the acquisition and dissemination of γ_2 can be performed during initialization.

Our Contribution

In this paper we make the following contributions:

- We formulate the problem of constructing a hierarchy of recursively defined top-k views. We solve this problem by introducing MINT Views. We also present a stateless, non-materialized version of MINT, coined *INT* (In-Network Top-k) Views, that is appropriate for sensing device with limited memory.
- We introduce the notion of a *k-covered bound set* V' which only maintains the tuples of V that lead to the k highest ranked answers at the sink. We additionally provide energy-conscious techniques to incrementally and immediately update V'.
- We experimentally validate the efficiency of our propositions, with an extensive experimental study that utilizes real sensor readings from the UC-Berkeley study at the Great Duck Island in Maine [29] and atmospheric readings from the University of Washington [12].

The remainder of the paper is organized as follows: Section 2 formalizes our system model and Section 3 overviews the related research work. Section 4 introduces the MINT View Framework along with a description of its three phases: *construction, pruning and updating*. Section 5 presents an extensive experimental study and Section 6 concludes our paper.

2. SYSTEM MODEL AND TERMINOLOGY

In this section we will formalize our system model and the basic terminology upon which we will describe our algorithms. The main symbols and their respective definitions are summarized in Table 1.

Let S denote a set of n sensing devices $\{s_1, s_2, ..., s_n\}$. Assume that s_i $(i \leq n)$ is able to acquire m physical attributes $A = \{a_1, a_2, ..., a_m\}$ from its environment at every discrete time instance t. This generates temporal tuples of the form $\{t, a_1, a_2, ..., a_m\}$ at each sensor. At any given time instance, the aforementioned scenario yields an $n \times m$ matrix of readings $R{:=}(s_{ij})_{n \times m}$. This matrix is *horizontally fragmented* across the n sensing devices (i.e., row i contains the readings of sensor s_i and $R = \cup_{i \in n} R_i$.

In order to disseminate the query to the n sensors, we utilize a typical tree-based query dissemination mechanism [16], where the querying node sends the query Q to one sensor s_1 which recursively forwards the given query to all of its neighbors until all n sensors have received the given query. Without loss of generality, we adopt the *child anchor* mechanism proposed in [38], where a sensor s_i confirms to exactly one of its neighbors s_j that it wants to be its child. This provides s_j with a list of children so that s_j can know when all the answers from its children have arrived.

We finally assume a TAG [21] topology maintenance policy that adapts to a shifting network by having each node to monitor the link quality of its neighbors and to switch parents if the quality drops below a given threshold.

Table 1: Definition of Symbols

Symbol	Definition
Q	A Query
k	Number of requested results
s_i	Sensor number i (s_0 denotes the sink).
n	Number of Sensors $\{s_1, s_2, ..., s_n\}$
m	Number of Attributes at each sensors $\{a_1, a_2, ..., a_m\}$
V_i	Local View (the results to Q) at sensor s_i $(i \leq n)$
V_i'	Pruned View at s_i (unveils the top-k answers at s_1)

3. BACKGROUND AND RELATED WORK

The main contribution of this paper is the integration of the seemingly unrelated areas of *Top-k Query Processing* and *View Management* in order to provide a novel framework for the continuous acquisition of query answers from a sensor network.

3.1 Top-k Query Processing

An example of a Top-k query is to *"Find the k=5 rooms with the highest average temperature,"* which returns a subset of the complete answer set in order to minimize the cost metric that is associated with the retrieval of the complete answer set. This cost is usually measured in terms of disk accesses or network transmissions, depending on where the data physically resides.

Top-k query processing has been studied in a variety of contexts including middleware systems [13, 14], web accessible databases [4, 23], stream processors [2], peer-to-peer systems [1] and other distributed systems [5, 38, 37]. It has been shown in numerous studies [13, 5, 4, 38], that top-k query processing is meaningful only if the predicate k refers to a small subset of the complete answer set (usually up-to 5%). For larger values of k, the query optimizer can choose to retrieve the complete answer set.

The wave of centralized top-k query processing algorithms was succeeded in recent literature by their distributed counterparts, namely the *TPUT* [5] algorithm, the *TJA* [38] algorithm and the *TPAT* [32] algorithm. The distributed top-k query processing problem with probabilistic guarantees rather than exact answers was studied in *KLEE* [24]. In all these scenarios, the queries are sporadic while we focus on continuous scenarios where a query is repeatedly evaluated over a specific period of time.

A method for continually providing approximate answers in a hierarchical sensor network scenario by exploiting temporal coherency was addressed in *TINA* [26, 27]. The basic idea behind TINA is to send a reading from a sensor only if the reading differs from the last recorded reading by more than a stated tolerance ϵ. In the experimental evaluation of Section 5, we will evaluate the performance of our proposed algorithms against the version of TINA that always returns the correct answer (i.e., $\epsilon = 0$). The problem of continually providing approximate top-k answers in a client-server setting was studied in [2]. The problem is tackled by installing arithmetic constraints at each node which define the current Top-k scores at any point. This work was later extended to a hierarchical sensor network environment in [10]. In all cases the results are approximate and continuous over a single attribute, thus operate over individual attributes (columns), while our approach is exact and operates horizontally covering all tuple attributes.

3.2 View Management

A view V in relational databases is a virtual table that contains the results from an arbitrary query Q which is evaluated every time V is referred to.

View Management has been another area of great contributions over the last decades [3, 7, 6, 18]. Materialized Views, in particular, have been extremely important in OLAP and Data Warehousing, where users are required to get quick answers to their aggregate queries over extremely large datasets. Most of the proposed solutions assume powerful and complex centralized or distributed DBMSes. Materialized views have also been extremely important in mobile databases because they provided the means to support disconnected operations [34, 33]. Similarly to mobile databases, we focus on wireless (sensor) devices with limited energy, CPU and memory resources. Additionally, our work is fundamentally different from Temporal View Management [31, 22], as our queries are not historic.

The notion of views in the context of sensor networks, has appeared in two very recent works. The first one proposes a new abstraction, coined *Model-based Views* which provides users with a unified view of data that hides away the irregularities of sensor data [11]. These views are implemented outside the sensor network. Thus, their scope and objective is supplementary to our approach, in which we utilize in-network views to optimize the acquisition of data from sensing devices. The second work [35] is similar to our approach but it uses in-network views to support ad-hoc queries in a data-centric environment as opposed to continuous and top-k queries in our approach.

The problem of materialized views which are generated by top-k queries in a centralized DBMS scenario was recently addressed in [9]. In particular, the authors study the problem of answering a top-k query from a set of N materialized top-k answers. These answers refer to different top-k queries which are neither distributed nor organized in a hierarchy, as this is the case in our setting. Finally in [19], the authors propose to exploit fully materialized views in sensor networks in order to speedup the execution of multiple queries. However these views are complete, rather than top-k, therefore their setting is closer to the TINA framework rather than the solutions proposed in this paper.

4. THE MINT VIEW FRAMEWORK

In this section we describe the underlying algorithms of the MINT View Framework. These also support, INT Views, MINT's stateless version that is appropriate for sensing device with limited memory. For ease of exposition, we present our framework in the following three conceptual phases:

A. *The Creation Phase*, executed during the first acquisition of readings from the distributed sensors. This phase results in n distributed views V_i ($i \leq n$);

B. *The Pruning Phase*, during which each sensor s_i locally prunes V_i and generates V_i' ($\subseteq V_i$). V_i' contains only the tuples that might be located among the final top-k results; and

C. *The Update Phase*, executed once per epoch, during which s_i updates its parent node with V_i'.

The above conceptual phases are executed distributively using the tree-based query routing protocol established by the operating system layer [16] after the query has been disseminated to the n sensors.

Algorithm 1 : Construct MINT/INT View

Input: A distributed sensor s_i ($\forall s_i \in S$) that generates m attributes $\{a_1, a_2, ..., a_m\}$, a query Q, an empty buffer $V_i = \{\}$
Output: A set of n distributed views $V = \{V_1, V_2, ..., V_n\}$.

```
1: procedure CONSTRUCT_MINT_VIEW(s_i, Q)
2:     // Execute Q and store the answer in V_i (takes O(1) time).
3:     insert(π_Q(σ_Q(current_reading())), V_i);
4:     for j = 1 to |children(s_i)| do
5:         c = child(s_i, j); // c is the j^th child of node s_i
6:         // w is a list of tuples returned to query Q.
7:         w = Construct_Mint_View(c, Q);
8:         for l = 1 to |w| do
9:             // w_l is the l^th entry of table w.
10:            // Inserts tuple w_l into local table V_i in O(1) time.
11:            insert(w_l, V_i);
12:        end for
13:    end for
14:    send(V_i, parent(s_i));
15: end procedure
```

4.1 MINT Creation Phase

The first phase of the algorithm is a recursive execution of Algorithm 1 at all sensors in a given network. Recall that a sensor generates an $(m+1)$-tuple of the form $v = \{t, a_1, a_2, ..., a_m\}$ at each timestamp t. A sensor starts out by performing the selection σ_Q that retains the tuples that satisfy the selection criterion (e.g., temperature>60). Note that a sensor can acquire concurrently several readings, all of which might not be of interest to a particular query. For example, the MICA Weather board which was utilized in the Great Duck Island study [29] supplements the MICA motes with 14 physical parameters. Thus, we only project the attributes related to Q prior to storing the result in the in-memory buffer V_i (line 3). The next step of the algorithm merges the tuples that arrive from the children of s_i into V_i (line 4-13). This yields an in-network view similar to Figure 1 (right).

If the various values at each node of the depicted tree do not change across consecutive timestamps, then V can efficiently provide the answer to the subsequent re-execution of Q. On the contrary, whenever we have a deviation, or a change, in a parameter at s_i, this change has to cascade all the way up to the sink. A change at all sensors has a worst-case message complexity of $O(n)$ for every single timestamp of the *epoch* duration, thus we seek to optimize this process through the proposition of the pruning phase.

4.2 MINT Pruning Phase

Algorithm 1 constructs a hierarchy of views, where ancestor nodes in the routing hierarchy maintain a superset view of their descendants. Before we explain the details of the pruning phase which minimizes messaging between sensors consider the following query:

```
Query 3
SELECT TOP k roomno, AVERAGE(temp)
FROM sensors
GROUP BY roomno
EPOCH DURATION 1 min
```

which returns the k rooms with the highest average temper-

Algorithm 2 : Prune MINT/INT View

Input: A distributed sensor s_i ($\forall s_i \in S$), a buffer V_i that contains the local view, a set of descriptors $\gamma = \{\gamma_1, \gamma_2, ...\}$, a query result parameter k.
Output: A locally pruned view V_i', such that V_0' can be utilized to answer a top-k query Q.

```
1: procedure PRUNE_MINT_VIEW(V_i)
2:     for j = 1 to |V_i| do // Identify the pruning threshold v_k^lb.
3:         v_j = V_i[j] // v_j=(v_j^lb, v_j^ub) pair.
4:         kHighest(v_j^lb,kBuff)
5:         bucketinsert(v_j^ub,sortedUBs)
6:     end for
7:     v_k^lb = min(kBuff);
8:     for j = 1 to |sortedUBs| do
9:         v_j^ub =sortedUBs[j]
10:        If (v_j^ub < v_k^lb) then break; end if
11:        add_to_candidates(v_j, V_i');
12:    end for
13: end procedure
```

ature. If s_i could locally define the k-highest answers to Q2 (at s_0), then s_i could use this information to prune its local view V_i. However, this is a recursively defined problem that can only be solved once all tuples percolate up to the sink s_0. In order to avoid this, we utilize a set of descriptors γ which are utilized to bound above the attributes in V_0 and subsequently enable a powerful pruning framework.

Consider the example of Figure 2 (left), where we illustrate the V_i for a given sensor. Prior to the execution of $Q2$ we established that γ_1= *"Maximum possible temperature value"*=120 and γ_2= *"Number of sensors in each room"*=5. The figure indicates the *sum* and *count* for several room numbers. By observing column 3 (i.e., count), it becomes evident that the *sum* for the rooms $\{2, 5, 11, 12, 15\}$ is a partial value of the *sum* returned at the sink (since $\gamma_2 = 5$).

On the contrary, the tuple of room 6 is already in its final form (i.e., 500). In this example the *sum* of each row is bounded above using the following formula $sum' = sum + (\gamma_2 - count) * \gamma_1$ and bounded below using the actual attribute *sum*. This creates six lower-bound (lb) and upper-bound (ub) pairs which precisely show the range of possible values for the *sum* attribute at the sink.

Having such knowledge locally, it can now help us to prune (lb, ub) pairs which will not be in the final top-k result. The intuition behind our algorithm is to identify the k^{th} highest lower bound (i.e., v_k^{lb}) and then eliminate all the tuples that have an upper bound (i.e., v^{ub}) below v_k^{lb}. Figure 2 (right), visually depicts this idea. We will prove that by applying locally such an operation yields at the end the correct top-k tuples at the sink. In order to achieve this we define the notion of a *k-Covered Bound-Set* as following:

Definition 1: **k-Covered Bound-Set (V_i')** is the subset of V_i that satisfies the following condition: If there is some $v \notin V_i'$, then $v^{ub} < v_k^{lb}$, where v_k^{lb} is the k^{th} highest lower bound[2].

Algorithm 2 illustrates the pruning of V_i at some arbitrary node s_i and the construction of the candidate set V_i'. This algorithm applies to both the MINT View and the INT View frameworks. The first step of the algorithm (lines 2-6) identifies the pruning threshold v_k^{lb}. This threshold allows the algorithm to prune-away tuples that will not be in the result.

Although V_i physically resides in main memory, we want to minimize the running time of our algorithms in order to accommodate the scarce energy budget. In particular, we utilize similarly to the well known *selection algorithm*, a k-element buffer $kBuff$ in order to locate v_k^{lb} in linear time (i.e., $O(k)$ per tuple). This procedure takes place inside the $kHighest$ function which inserts v_j^{lb} into $kBuff$, if the former is larger than the minimum item in $kBuff$.

The next step of the algorithm is to locate the tuples that have an upper bound v^{ub} below the threshold v_k^{lb}. By visually examining Figure 2, it is easy to see that an efficient way to do so is to create an ordered list of upper bounds and then perform a linear scan in descending order until a tuple v_j^{ub} ($<v_k^{lb}$) is located. Any upper bound below or equal to v_j^{ub} can be safely eliminated.

The ordered list can be constructed in parallel with the location of the pruning threshold v_k^{lb}. In particular, while scanning for v_k^{lb}, we insert each upper bound v_j^{ub} into a new table *sortedUBs* (line 5). This takes only O(1) per tuple as we utilize an idea similar to *bucketsort*. However, if memory is limited then this optimization can be avoided without any consequence on the correctness of our approach.

In lines 8-12, we finally perform a linear scan of the *sortedUBs* table in descending order and stop when we find a tuple v_j^{ub} that is below v_k^{lb}. The correctness of our algorithm is established by Theorem 1.

Theorem 1. *The k-Covered Bound-Set V_i' correctly identifies the k-highest ranked answers to Q.*

Proof (by contradiction): Let v denote an arbitrary tuple which is not included in the k-Covered Bound-Set V_i'. We have to show that v will have a smaller value than any of the k highest-ranked tuples w (i.e., $v < w$). Assume that $v \geq w$. It always holds that $v^{ub} \geq v$ which consequently yields $v^{ub} \geq w$ (by using the assumption). However if $v^{ub} \geq w$, then v would have been included in V_i', by definition 1, a contradiction □

4.3 MINT Update Phase

In the previous step, we transformed V_i into a pruned subset V_i'. We shall now describe how to incrementally and recursively update V_i'. Let T' denote the V_i' taken at the last execution of Q. The below description only applies to the MINT View framework, for which T' is available. The update phase of the INT View framework is simply a re-execution of Algorithm 1 which re-constructs V_i' from the beginning.

Since our objective is to identify the correct results at the sink, we utilize an *immediate* view maintenance mechanism: "*As soon as a new tuple is generated at s_i, this update is reflected in V_i'*". In order to minimize communication, s_i only re-transmits V_i' to its parent, if V_i' has changed (*temporal coherence filter as in TINA*). Additionally, in order to minimize energy consumption even further, we seek to minimize processing consumption as well. Therefore, we aim to construct V_i' by avoiding the re-executing of Algorithm 2.

[2]Due to contraposition, the condition could also be expressed using the implication *if $v^{ub} \geq v_k^{lb}$, then $v \in V_i'$.*

room	sum	count	sum'
2	200	4	320
5	270	4	390
6	500	5	500
11	460	4	580
12	290	3	530
15	130	2	490

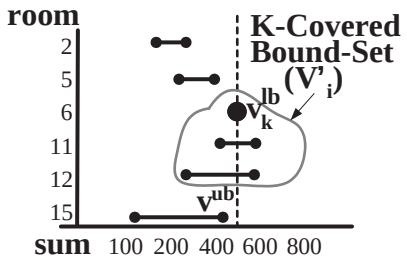

Figure 2: The left table illustrates the V_i of a given node during the execution of query Q3. The right figure illustrates the intuition of the pruning algorithm. In particular, we plot the (lb,ub) ranges for the various returned tuples at some arbitrary node. We then generate a k-covered bound set V_i' using Algorithm 2. We only propagate a tuple u to the parent of s_i, if $u \in V_i'$.

Algorithm 3 presents the MINT Update Algorithm. In particular, line 3 shows that any tuple update x with an upper bound (denoted as x^{ub}) less than the v_k^{lb} can be *ignored*. In the opposite case, we add the tuple x to the set of candidates V_i' (line 4). Now the remaining question is whether v_k^{lb} has changed by this addition of x. If $x^{lb} \leq v_k^{lb}$ is true then v_k^{lb} has not changed. Consequently, s_i only propagates the update x towards its parent rather than a complete view update. In the implementation we buffer these updates until all children send their updates to their parents. If on the contrary $v_k^{lb} < x^{lb}$, then v_k^{lb} might have changed. As a result s_i has to reconstruct V_i' using Algorithm 2 and transmit the complete V_i' to its parent. This re-construction procedure is necessary to guarantee the correctness of our framework. Note that the reconstruction only happens for $|V_i'|$ elements rather than all the elements (i.e., $|V_i|$), had we executed Algorithm 2 for the first time.

Algorithm 3 : **Update MINT View**

Input: A buffer T' that contains the V_i' of the previous time instance, the v_k^{lb} of T', a tuple update x from some child.
Output: A locally pruned view V_i', such that V_0' can be utilized to answer a top-k query Q.

```
1: procedure UPDATE_MINT_VIEW(T', v_k^lb, x)
2:     V_i' = T';
3:     if (v_k^lb ≤ x^ub) then
4:         add_to_candidates(x, V_i');
5:         if (x^lb ≤ v_k^lb) then
6:             send(x, parent(s_i)); // Single tuple x update
7:         else  // x^lb > v_k^lb
8:             Prune_MINT_View(V_i'); // Using Algorithm 2
9:             send(V_i', parent(s_i)); // Complete V_i' update
10:        end if
11:    end if
12:    T' = V_i';
13: end procedure
```

4.4 Discussion

MINT vs. INT: The differences of the two algorithms are summarized as following: i) MINT exploits a temporal coherence in order to suppress view updates that do not change between consecutive time instances, while INT has to re-send these updates, because it is stateless. ii) In MINT, we only have to update V_i' using Algorithm 3 (in $O(|V_i'|)$ time), while in INT we have to construct it every time from the beginning, in $O(|V_i|)$ time, using Algorithm 2. iii) INT has the advantage of not requiring any extra storage thus is more appropriate for sensors for which the storage is at premium.

Deferred View Updates: In order to minimize communication even more in the MINT/INT Views, we could have opted for a *deferred* view maintenance mechanism, rather than a *immediate* one. A *deferred* mechanism could propagate changes periodically, after a certain number updates or even randomly. In all cases this would produce probabilistic answers at the sink, as the sink would not have at its disposal the most up-to-date view. Although deferred view maintenance mechanisms are extremely interesting in the context of sensor networks, as these allow us to trade accuracy versus energy consumption, in this paper we only focus on exact answers.

In-Memory Buffering: The materialized views and temporary results of all algorithms, can either reside in an SRAM-based buffer or a Flash-based buffer. For instance, a typical MICA mote with a 2KB SRAM might need to exploit the 512KB on-chip flash memory, while Intel's i-mote might easily store these results in the 64KB SRAM. There is a growing trend for more available local storage in sensor devices [36] and therefore local buffering of results is not a threat to our model.

Supported Query Types: We support single-relation queries with the standard aggregate functions (i.e., `SUM, MIN, MAX` and `AVERAGE`). In contrast with other frameworks, we optimize queries with *multi-tuple* answers. Such answers could be generated by a `GROUP-BY` clause, or by a non-aggregate query. Note that for *single-tuple* answers, such as those generated by an aggregate query without a group-by clause, there is no notion of a top-k result.

5. EXPERIMENTAL EVALUATION

In this section we present an extensive experimental comparison of INT and MINT Views against two other popular query processing frameworks namely, TAG and TINA.

5.1 Experimental Methodology

We adopt a trace-driven experimental methodology in which a real dataset from n sensors is fed into our custom-built C++ simulator. Our methodology is as following:

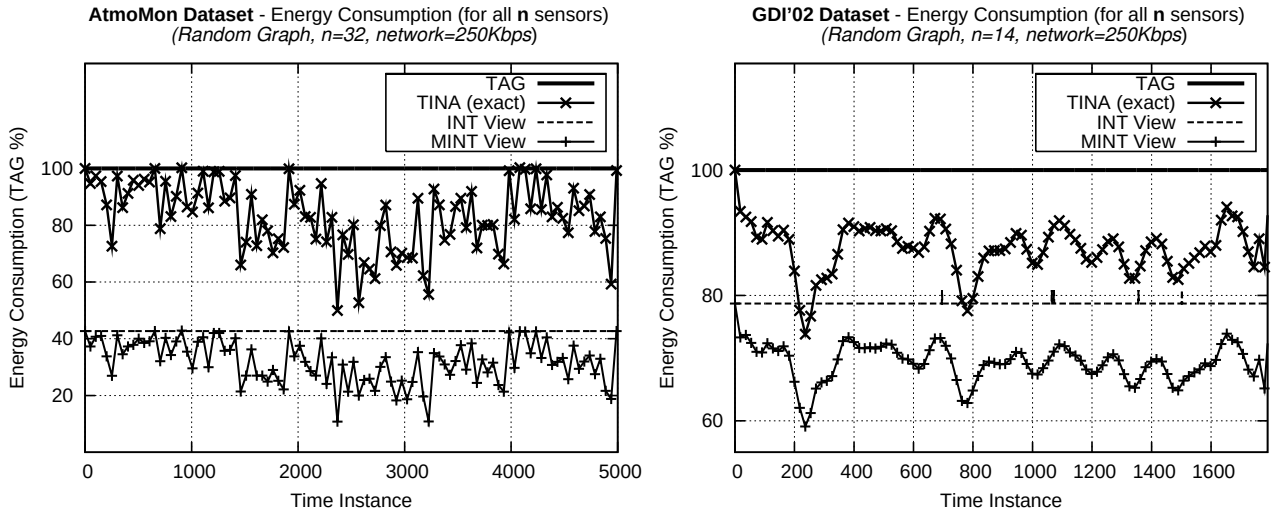

Figure 3: Energy Consumption for the TAG, TINA, INT View and MINT View frameworks using the TelosB sensor energy model.

Algorithms: We implemented i) *TAG*, which relies on in-network aggregation to minimize communication; ii) TINA (exact), which deploys in addition to in-network aggregation suppression of consecutive values if these do not change; iii) INT and MINT Views, as these were described in section 4. As a baseline for comparison we utilize the results from the TAG approach.

Communication Protocol: Our communication protocol is structured in the following way: each message is associated with a 5 *Byte* TinyOS header. This is augmented with an additional 6B application layer header that includes: (i) the sensor identifier (1B), (ii) the message size (4B) and the depth of a cell from the querying node (1B). In each message we allocate 2B for environmental readings (e.g., temperature, humidity, etc.), 4B for aggregate values (max, min and sum) and 8B for timestamps.

Sensing Device: We use the energy model of Crossbow's new generation TelosB [8] sensor device to validate our ideas. TelosB is a ultra-low power wireless sensor equipped with a 8 MHz MSP430 core, 1MB of external flash storage, and a 250Kbps RF Transceiver that consumes 23mA when the radio is on. Our performance measure is *Energy*, in *Joules*, that is required at each discrete time instance to resolve the query. The energy formula is as following: $Energy(Joules) = Volts \times Amperes \times Seconds$. For instance the energy to transmit 30 bytes at 1.8V is : $1.8V \times 23 * 10^{-3}A \times 30 * 8bits/250kbps = 39\mu J$.

Datasets: We utilize two datasets:

1. *Washington State Climate (AtmoMon):* This is a real dataset of atmospheric data collected at 32 sensors in the Washington and Oregon states, by the Department of Atmospheric Sciences at the University of Washington [12]. More specifically, each of the 32 sensors maintains the average temperature and wind-speed on an hourly basis for 208 days between June 2003 and June 2004 (i.e., 4990 time moments).

2. *Great Duck Island (GDI 2002):* This is a real dataset from the habitat monitoring project deployed in 2002 on the Great Duck Island which is 15km off the coast of Maine [29], USA. We use readings from the 14 sensors that had the largest amount of local readings (out of the initial 43 sensors). The GDI dataset includes readings such as: light, temperature, thermopile, thermistor, humidity and voltage. In both datasets we randomly and uniformly divide the sensors in 16 and 4 areas respectively. Our query is *"SELECT TOP-k area, AVG(temp) FROM sensors GROUP BY area"*, where k is configured as the 5% of the complete answer set.

5.2 Energy Consumption

In the first experimental series we evaluate the energy consumption of the four algorithms. Due to the exploitation of temporal coherence in the TINA and MINT Views, the energy value between consecutive time instances can greatly vary. To facilitate our presentation, we apply a spline interpolation smoothing between consecutive data points which accurately approximates the TINA and MINT curves.

In Figure 3 (left), we plot the results using the AtmoMon dataset. Since we utilize TAG as the baseline of comparison, it always has a value of 100%. The TAG line accounts for approximately 5.3mJ of energy for all 32 nodes of the network. Recall that in TAG a sensor always transmits all aggregated tuples to the sink. Although TINA (exact) still returns all answers to the sink, it takes the energy consumption down to 4.4mJ with a standard deviation of 0.66mJ. This validates that by exploiting temporal coherence can be beneficial in most cases. The INT Views approach on the other hand, performs in-network pruning of the results which reduces the energy consumption to 2.26mJ (i.e., $\approx$ 58% less than TAG).

Finally MINT Views exploit temporal coherence in ad-

dition to top-k pruning and only consume an average of 1.69mJ±0.23mJ which is equivalent to a 68% energy reduction from TAG, 38% energy reduction from TINA and 10% from INT. The reason why the TINA and MINT Views follow a similar pattern is because in both curves the energy reduction is dominated by the savings that are due to the temporal coherence between consecutive time points.

By repeating the same experiment on the GDI'02 dataset, we observe in Figure 3 (right), that MINT continuous to maintain a competitive advantage over TAG and TINA. In particular, we observe that MINT consumes 30% less energy than TAG (i.e. 1.96±0.19mJ versus 2.83mJ). We noticed that smaller-sized networks are not beneficial for INT and MINT Views, because shallow query routing trees can not facilitate top-k pruning. This is also the case for the GDI'02 dataset which consists of only of 14 sensors. We also observe in this illustration the surges (deviations) in the INT View Mechanism (e.g., at time instance 780). This is an indication that the top-k answer has changed at the particular timestamp and that this has brought some increase in energy consumption, until the updates propagate to the sink. Similar surges also exist in the MINT curve but these can not be observed due to the temporal coherence fluctuation.

5.3 Pruning Magnitude

We next study the pruning magnitude of the k-Covered Bound-Set V_i'. In Figure 4 we plot with a white box the average number of tuples at each level of the topology (for all 4990 time instances). We also plot with a dashed box the aggregate number of tuples eliminated by Algorithm 2.

We observe that the closer we move towards the sink, the pruning power of our framework increases exponentially. This is attributed to the fact that the cardinality of V_i can increase in the worst case exponentially as well (i.e., each sensor reports a different room number). In particular, we observe that the pruning at level five to one ranges from 0% (where only leaf nodes exist), to 39% in level two and 77% in level one. It is important to highlight the fact that such a pruning presents a reduction of more than 20,000 tuples at level one alone.

A final remark is that these results apply to both MINT and INT, as these two algorithms only differ in how V_i' is maintained and not on the final content of the in-network view.

6. CONCLUSIONS AND FUTURE WORK

This paper introduces and formalizes the problem of exploiting materialized in-network views in order to optimize the execution of continuous queries in sensor networks. We formulate the problem of constructing a hierarchy of recursively defined top-k views. We solve this problem by introducing MINT Views. We also present a stateless, non-materialized version of MINT, coined *INT* (In-Network Top-k) Views that is appropriate for sensing device with limited memory. Our trace-driven experimentation with real datasets from UC-Berkeley and the University of Washington show that MINT offers tremendous energy reductions.

In the future we plan to implement and validate our ideas using a nesC prototype that is currently under development. We also aim to provide an efficient solution to approximate top-k views which could offer even more energy savings when exact answers are not needed.

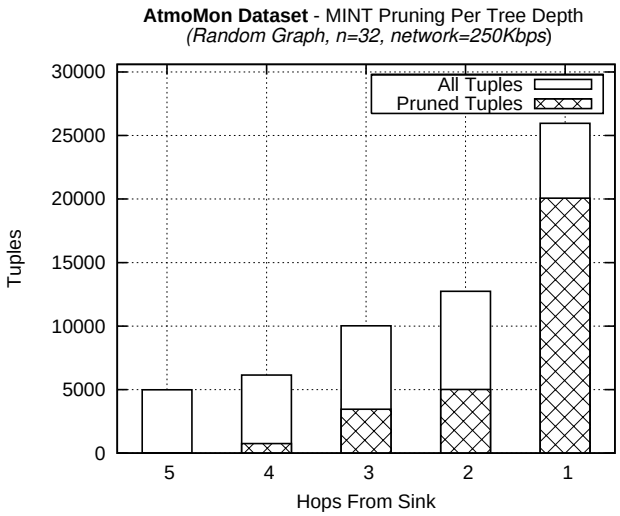

Figure 4: Pruning Magnitude of MINT Views.

Acknowledgements: We would like to thank Joe Polastre (UC Berkeley) for the Great Duck Island data trace. This work was supported in part by the US National Science Foundation under projects S-CITI (#ANI-0325353) and AQSIOS (#IIS-0534531), the European Union under the project mPower (#034707) and the Cyprus Research Foundation under the project GEITONIA (#PLYP-0506).

7. REFERENCES

[1] Balke W.-T., Nejdl W., Siberski W., Thaden U., "Progressive Distributed Top-K Retrieval in Peer-to-Peer Networks", In *ICDE*, pp.174-185, 2005.

[2] Babcock B. and Olston C., "Distributed Top-K Monitoring", In *ACM SIGMOD*, pp.28-39, 2003.

[3] Blakeley J. and Larson P.A. and Tompa F.W., "Efficiently Updating Materialized Views", In *ACM SIGMOD*, pp.61-71, 1986.

[4] Bruno N., Gravano L. and Marian A., "Evaluating Top-K Queries Over Web Accessible Databases", In *ICDE*, pp.319-362, 2004.

[5] Cao P. and Wang Z., "Efficient Top-K Query Calculation in Distributed Networks", In *ACM PODC*, pp.206-215, 2004.

[6] Chaudhuri S., Krishnamurthy R., Potamianos S., Shim K., "Optimizing Queries with Materialized Views". In *ICDE*, pp.190-200, 1995.

[7] Colby L.S., Griffin T., Libkin L., Mumick I.S., Trickey H., "Algorithms for Deferred View Maintenance" In *ACM SIGMOD*, pp.469-480, 1996.

[8] Crossbow Technology, Inc. http://www.xbow.com/

[9] Das G., Gunopulos D., Koudas N., Tsirogiannis D., "Answering Top-k Queries Using Views", In *VLDB*, pp.451-462, 2006.

[10] Deligiannakis A., Kotidis Y., Roussopoulos N., "Compressing historical information in sensor networks", In *ACM SIGMOD*, pp.527-538, 2004.

[11] Deshpande A. and Madden S., "MauveDB: Supporting Model-Based User Views in Database Systems", In *ACM SIGMOD*, pp.73-84, 2006.

[12] Earth Climate and Weather, University of Washington: http://www-k12.atmos.washington.edu/k12/grayskies/

[13] Fagin R., "Combining Fuzzy Information from Multiple Systems", In *ACM PODS*, Montreal, Canada, Vol.58, No.1, pp.83-99, 1999.

[14] Fagin R., Lotem A. and Naor M., "Optimal Aggregation Algorithms For Middleware", In *ACM PODS*, pp.102-113, 2001.

[15] Gay D., Levis P., Von Behren R. Welsh M., Brewer E. and Culler D., "The nesC Language: A Holistic Approach to Networked Embedded Systems", In *ACM PLDI*, pp.1-11, 2003.

[16] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., "System Architecture Directions for Networked Sensors", In *ACM SIGOPS Operating Systems Review*, Vol.34, No.5, pp.93-104, 2000.

[17] Intanagonwiwat C., Govindan R. Estrin D., "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In *ACM MOBICOM*, pp.56-67, 2000.

[18] Larson P-A., H. Z. Yang: "Computing Queries from Derived Relations", In *VLDB*, pp.259-269, 1985.

[19] Lee K.C.K., Lee W-C., Zheng B., Winter J., "Processing Multiple Aggregation Queries in Geo-Sensor Networks", In *DASFAA*, pp.20-34, 2006.

[20] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "The Design of an Acquisitional Query Processor for Sensor Networks", In *ACM SIGMOD*, pp.491-502, 2003.

[21] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In *USENIX OSDI*, Vol.36, pp.131-146, 2002.

[22] Maiocchi R., Pernici B., "Temporal Data Management Systems: A Comparative View", In *IEEE TKDE*, Vol.3, No.4, pp.504-524, 1991.

[23] Marian A., Gravano L., Bruno N., "Evaluating Top-k Queries over Web-Accessible Databases", In *TODS*, Vol.29, No.2, pp.319-362, 2004.

[24] Michel S., Triantafillou P., Weikum G., "KLEE: A Framework for Distributed Top-K Query Algorithms", In *VLDB*, pp.637-648, 2005.

[25] Sadler C., Zhang P., Martonosi M., Lyon S., "Hardware Design Experiences in ZebraNet", In *ACM SenSys*, pp.227-238, 2004.

[26] Sharaf M.A. , Beaver J., Labrinidis A. and Chrysanthis P.K., "TiNA: a scheme for temporal coherency-aware in-network aggregation", In *MobiDe*, pp.69-76, 2003.

[27] Sharaf M.A. , Beaver J., Labrinidis A. and Chrysanthis P.K., "Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks ", In *VLDB Journal*, Vol. 13, Iss.4, pp.384-403, 2004.

[28] Shenker S., Ratnasamy S., Karp B., Govindan R., Estrin D., "Data-centric storage in sensornets", In *ACM SIGCOMM Computer Communication Review*, Vol. 33 , Iss. 1, pp.137-142, 2003.

[29] Szewczyk R., Mainwaring A., Polastre J., Anderson J., Culler D., "An Analysis of a Large Scale Habitat Monitoring Application", In *ACM SenSys*, pp.214-226, 2004.

[30] Yao Y., Gehrke J.E., "The cougar approach to in-network query processing in sensor networks", In *SIGMOD Record*, Vol.32, No.3, pp.9-18, 2002.

[31] Yang J. and Widom J., "Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing", In *EDBT*, pp.389-403, 1998.

[32] Yu H., Li H., Wu P., Agrawal D., Abbadi A.E., "Efficient Processing of Distributed Top-k Queries", In *DEXA*, pp.65-74, 2005.

[33] Weissman-Lauzac S., Chrysanthis P.K., "Personalizing information gathering for mobile database clients," In *ACM SAC*, pp.49-56, 2002.

[34] Weissman-Lauzac S., Chrysanthis P.K., "Utilizing Versions of Views within a Mobile Environment", In *ICCI*, pp. 201-208, 1998.

[35] Xia P., Chrysanthis P.K., Labrinidis A., "Similarity-Aware Query Processing in Sensor Networks", In WPDRTS, 2006.

[36] Zeinalipour-Yazti D., Lin S., Kalogeraki V., Gunopulos D., Najjar W., "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices", In *Usenix FAST*, pp. 31-44, 2005.

[37] Zeinalipour-Yazti D., Lin S., Gunopulos D., "Distributed Spatio-Temporal Similarity Search" In *ACM CIKM*, pp. 14-23, 2006.

[38] Zeinalipour-Yazti D., Vagena Z., Gunopulos D., Kalogeraki V., Tsotras V., Vlachos M., Koudas N., Srivastava D., "The Threshold Join Algorithm for Top-K Queries in Distributed Sensor Networks", In *VLDB's DMSN*, pp.61-66, 2005.

[39] D. Zeinalipour-Yazti, P. Andreou, P. Chrysanthis and G. Samaras, "MINT Views: Materialized In-Network Top-k Views in Sensor Networks", In MDM, 2007.