Towards In-Situ Data Storage in Sensor Databases

D. Zeinalipour-Yazti, V. Kalogeraki, D. Gunopulos,
A. Mitra, A. Banerjee, W. Najjar

Department of Computer Science & Engineering
University of California - Riverside,
Riverside CA 92521, USA
{csyiazti, vana, dg, amitra, anirban, najjar}@cs.ucr.edu

Abstract. The advances in wireless communications along with the exponential growth of transistors per integrated circuit lead to a rapid evolution of *Wireless Sensor Devices (WSDs)*, that can be used for monitoring environmental conditions at a high fidelity. Following the current trends, WSDs are soon expected to automatically and continuously collect vast amounts of temporal data. Organizing such information in centralized repositories at all times will be both impractical and expensive. In this paper we discuss the challenges from storing sensor readings *In-situ* (at the generating sensor). This creates a network of tiny databases as opposed to the prevalent model of a centralized database that collects readings from many sensors. We also discuss some of the inherent problems of such a setting, including the lack of efficient distributed query processing algorithms for handling *temporal* data and the lack of efficient access methods to locally store and retrieve large amounts of sensor data. The presented solutions are in the context of the *RISE (Riverside Sensor)* hardware platform, which is a wireless sensor platform we developed for applications that require storing in-situ many MBs of sensor readings.

1 Introduction

The improvements in hardware design along with the wide availability of economically viable embedded sensor systems enable researchers nowadays to sense environmental conditions at extremely high resolutions. Traditional approaches to monitor the physical world include passive sensing devices which transmit their readings to more powerful processing units for storage and analysis. *Wireless Sensor Devices (WSDs)* on the other hand, are tiny computers on a chip that is often as small as a coin or a credit card. These devices feature a low frequency processor (≈4-58MHz) which significantly reduces power consumption, a small on-chip flash card (≈32KB-512KB) which can be used as a small local storage medium, a wireless radio for communication, on-chip sensors, and an energy source such as a set of AA batteries or solar panels [9]. This multitude of features constitute $WSDs$ powerful devices which can be used for in-network processing, filtering and aggregation [8, 7, 11]. Large-scale deployments of sensor

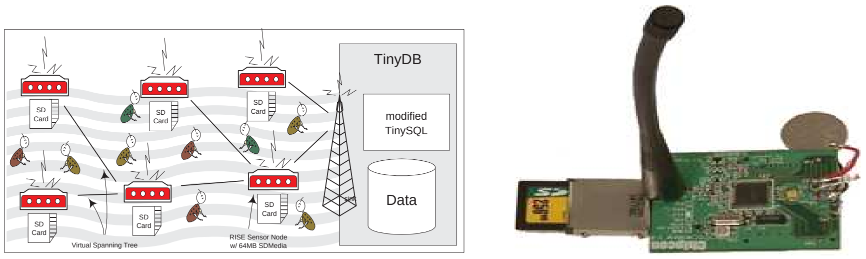


Fig. 1. a) Soil-Organism Monitoring Application: Each sensor stores locally on external flash memory the CO_2 levels in a sliding window fashion. The user might then ask: *"Find the time instance on which we had the highest average CO_2 levels in the last month?"*. b) Our platform: The RISE (Riverside Sensor), which is the first sensor device that features a large external storage medium (an SDMedia flash card) .

network devices have already emerged in environmental and habitant monitoring[17, 9], seismic and structural monitoring [12], factory and process automation and a large array of other applications [18, 7, 8, 11].

Conventional approaches to monitoring have focused on dense deployed networks that either transfer the data to a central sink node or perform in-network computation and generate alerts when certain events arise. An important attribute of these applications is that the time interval between consecutive query re-evaluations (*epoch*) is small because the applications require the ability to quickly react to various alerts. For example, a query might continuously manipulate the temperature at some region in order to identify fires or other extreme situations (e.g. *"Find which sensors record a temperature>95F?"*). Therefore the querying node (*sink*) must continuously maintain an updated view of the values recorded at the sensors [11, 7]. In such *short-epoch* applications, the frequency of updates and the timely delivery of information from the sensors play a vital role in the overall success of the system.

On the other hand, a class of applications that was not addressed to this date are *long-epoch* applications. In these applications the user needs an answer to his query more sparsely (e.g. weekly or monthly), although the sensor still acquires data from its surrounding environment frequently (e.g. every second). The user might then ask: *"Find the time instance on which we had the highest average temperature in the last month?"*. In order to evaluate this query using current techniques would require each sensor to report all its values for the last month. This happens because the data is fragmented across the different nodes and an answer to the query can only be obtained after accessing all distributed relations in their entirety. We call this type of *in-situ* data fragmentation *vertical partition*, because each sensor's timeseries is one dimension in the n-dimensional space of sensor readings. This makes it a challenging task to answer user queries efficiently.

Our Contribution: In this paper we study the deployment of sensor devices characterized by large external memories. This will allow each sensor node to accumulate measurements over a large window of time, avoiding the multi-hop burden of transferring everything to the sink. This creates a network of tiny databases as opposed to the prevalent model of a centralized database that collects readings from many sensors. We also address some of the inherent problems of such a distributed database setting. Specifically we propose efficient distributed query processing algorithms for efficiently answering top-k queries in a distributed environment. These queries have been extensively studied by the database community and their task is to retrieve the k highest ranked answers to a given query. An example of a top-k query might be *"Find the three moments on which we had the highest average temperature in the last month?"*.

Temporal and top-k queries are useful in a number of contexts. Our work is motivated by the requirements of the Bio-Complexity and the James Reserve Projects at the Center of Conservation Biology (CCB) at UC Riverside.[1] CCB is working towards the conservation and restoration of species and ecosystems by collecting, evaluating scientific information (Figure 1a). The bio-complexity project is designed to develop the kinds of instruments that can monitor the soil environment directly, rather than in laboratory recreations.

We have developed the *RISE platform*, in which sensors feature a large external memory (SD flash memory). RISE sensors are able to store measurements of Carbon-dioxide levels in the soil as well as ambient sound from the surrounding environment over a large period of time. This will allow scientists to monitor the long-term behavior of certain soil micro-organisms and bird species.

We also address the efficient evaluation of top-k queries in our platform by sketching an algorithm that estimates some threshold below which tuples do not need to be fetched from the sensor nodes. Key ideas of our algorithm are to transmit only the necessary information towards the querying node and to perform the query execution in the network rather than in a centralized way.

2 The RISE Platform

The *RISE (RIverside SEnsor)* platform (see figure 1b) employs a System-on-Chip interfaced with a large external storage memory, an off-the-shelf SD (Secure Digital) media card, to develop a new paradigm of "sense and store" as opposed to the prevalent "sense and send". The RISE platform was conceived by observing the twin trends of falling flash memory prices and the need of larger memories on sensor devices for more efficient querying, processing and communication. Also, higher levels of device integration at low cost and size now provide us with single chip solutions for most of the sensor and communication needs, reducing complexity and improving performance. The RISE wireless platform is built around the Chipcon CC1010 System on Chip (SoC), which together with just a few external passive components and the required sensors constitutes a

[1] http://www.ccb.ucr.edu/

powerful, robust and versatile wireless embedded sensor system. The following is a description of the important components of the RISE platform :

1. **The MicroController Unit (MCU):** The Chipcon CC1010 SoC is a true single-chip RF transceiver with an integrated high performance 8051 micro-controller and high end features which include a 32KB flash memory, an SPI (Serial Peripheral Interface) bus, DES encryption, 26 general I/O pins and many other components constituting it appropriate for a multitude of sensor and computation needs.

2. **The SD-Card interface:** An SD-Card (Secure Digital Card) has been interfaced to the main chip using the SPI bus equipping the RISE platform with a large external storage memory (up to a 4 GB!). Data can be buffered on the 32KB flash memory for efficiently reading and writing on the SD-Card. Data is transferred to the SDCard in blocks of 512 bytes at a maximum rate of 82KBps (although the SPI interface supports up to 3MBps). We have developed tiny access method structures which are deployed directly on the sensor. These provide efficient sorted and random access to local data in the event of some query.

3. **The OS & Compilation:** To facilitate ease and modularity of programming, we have ported the most prevalent design environment, the TinyOS (version 1.1) and nesC (version 1.2alpha1), facilitating easier and modular programming, interfacing of an SD-CARD and developing the reactive methodology of query based response on large datasets stored locally on the nodes.

4. **Deployed Sensors:** The platform has a temperature sensor and is also being interfaced with a CO_2 sensor and a microphone.

Note that the energy cost of writing to flash memory is much cheaper than the RF transmission cost even in the case of a single hop. We have measured the performance of transmitting one byte over the RF radio and found that it requires 164 μJ while storing the same byte on the flash card requires 1.5 μJ. Although writing to the external flash card can only be performed on a page-to-page basis (i.e. 512 bytes), the 32KB on-chip flash memory allows us to buffer a page before it is written out. This in combination of the fact that the energy required for the transmission of one byte is roughly equivalent to executing 1120 CPU instructions, makes local storage and processing highly desirable.

3 The Query Processing Framework

In this section we expand on the class of queries we consider in the RISE platform. This class represents queries that are interesting and important in our framework that is characterized by long-epochs and large storage capacities at individual sensors. We also describe and contrast alternative frameworks that have been proposed for data acquisition in sensor networks.

3.1 Temporal and top-k queries in RISE

We use a query dissemination mechanism similar to the one described in [8, 7], which creates a "virtual" *Query Spanning Tree (QST)* interconnecting all nodes in the network. This provides each node with the next hop towards the sink (See Figure 1). Alternatively each node could maintain multiple parents in order to achieve fault tolerance [1].

Let $G(V, E)$ denote the undirected and connected network graph that interconnects n sensors in V using the edge set E. The edges in E, represent the virtual connections (i.e. nodes are within communication radius) between the sensors in V. Also assume that each sensor has enough storage to record a window of m measurements. Each measurement has the form (ts, val), where ts denotes the timestamp on which the measurement was taken and val the recording at that particular time moment.[2] Essentially each sensor v_i has locally the following timeseries $list(v_i) = \{o_{i1}, o_{i2}, \ldots, o_{im}\}$, where o_{ij} denotes the recording of the i^{th} sensor node at the j^{th} time moment. Each time moment could logically be viewed as a collection of n values. A node can maintain several lists (e.g. temperature, humidity, others); for simplicity we assume that only one such time-series is being maintained. We look at two main classes of queries.

Temporal Queries: The queries we consider allow the user to find the state of the sensor network at different time intervals, but also to identify intervals that certain conditions hold. Examples of such queries are: *"Find the time intervals such that the sensor values satisfy a given condition,"* and *"Given a sequence of values, identify time intervals that show similar sequences in the values recorded by the sensor."*

Top-k Queries: An example of a top-k query is *"Find the k time instances with the highest average reading across all sensors."* More formally, consider $Q = (q_1, q_2, \ldots, q_n)$, a top-k query with n attributes. Each attribute of Q refers to the corresponding attribute of an object and the query attempts to find the k objects which have the maximum value in the following scoring function:
$Score(o_i) = \sum_{j=1}^{n} w_j * sim_j(q, o_i)$, where $sim_j(q, o_i)$ is some similarity function which evaluates the j^{th} attribute (sensor) of the query q against the j^{th} attribute of an object o_i and returns a value in the domain [0,1] (1 denotes the highest similarity). Since each sensor might have a different factor of importance, we also use a weight factor w_j ($w_j > 0$), which adjusts the significance of each attribute according to the user preferences. Note that, similarly to [4], we require the score function to be *monotone*. A function is monotone if the following property holds: if $sim_j(q, o_1) > sim_j(q, o_2)$ $(\forall j \in m)$ then $Score(o_1) > Score(o_2)$. This is true when $w_j > 0$.

[2] Sensors are time synchronized through a lower layer mechanism (e.g. The Operating System).

4 Query Evaluation Techniques

From the sink's point of view, denoted as v', the data in this scenario is vertically fragmented across the network. Therefore answering such a query would require v' to gather the whole space of $n * m$ values. In this section we sketch the TJA algorithm which alleviates the burden of transferring everything to the sink.

4.1 A Taxonomy of Data Gathering Techniques

Below we provide a taxonomy of data gathering techniques as a function of the available storage available at each node:

1. **Sense and Send (SS):** In this naive case each sensor node propagates its generated value towards its parent every time such value becomes available. This is, according to the terminology of [1], the LIST approach.

2. **Sense, Merge and Send (SMS):** In this scheme, each node aggregates the values coming from its children before forwarding its values to its parent. This is essentially the TAG approach [8]. In this scheme, all aggregates can not be treated in the same way. For example *Distributive Aggregates (e.g. Sum, Max, Min, Count)* can locally be aggregated into one value. *Holistic Aggregates (e.g. Median)* on the other hand, can not be treated in the same way as aggregation into one value could produce a wrong result.

3. **Sense, Store, Merge and Send (SSMS):** This is the scheme deployed in our platform, RISE. Each sensor node maintains locally in the flash memory a window of m measurements. This sliding window evolves with time, and therefore, once the limit of the available storage is reached, at each new time moment the oldest measurement is deleted. We note however that given the capacities of flash cards m can be very large. Registered queries can perform some local aggregation, if the correctness of the query outcome is not violated, before values are propagated towards the parent. Note that this is not possible in current systems such as TinyDB [7].

The three gathering techniques outlined above basically represent the scale of available memory at the sensor nodes (i.e. $SSMS \supset SMS \supset SS$). We believe that although the SMS approach offers in practice the most efficient way to cope with short-epoch applications, the SSMS approach is more practical for long-epoch applications.

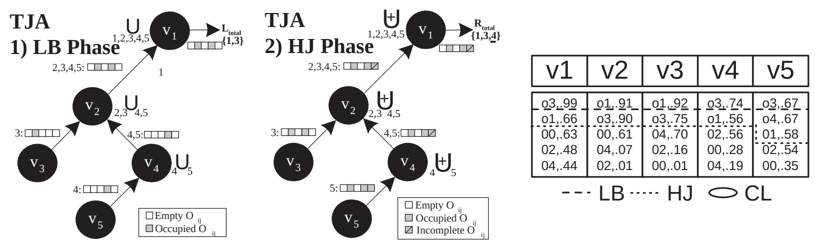
We note that under the SS model evaluating the kinds of queries we propose requires sending all information to the sink. Under the SMS model we can design algorithms that perform aggregation or more sophisticated computation in the network, there are however significant limitations. Due to the short-epoch emphasis of this model when information gets older than the window of interest, we have to discard this information or we have to transmit it for permanent storage to the sink (or other specially designated nodes in the network).

4.2 Providing Local Access Methods

Efficiently evaluating the queries described above requires efficient access to the data that is stored on the "external" flash memory. However, flash memory features some distinct characteristics that differentiate it from other storage media. Specifically, deleting data stored on flash memory can only be performed at a block granularity (typically 8KB-64KB) and writing can only be performed at a page granularity (typically 256B-512B), after the respective block has been deleted. Finally, each flash page has a limited life-time (10K-100K writes), after which the page wears out and can no longer be used. The problem of indexing over magnetic disks and RAM memory is well studied in the database community, however indexing on flash memory in conjunction with the low energy budget of sensor nodes introduce many new challenges. We have designed and implemented efficient access methods that provide random and sorted access to records stored on the flash medium. Our access methods serve as primitive functions for the efficient execution of a wide spectrum of queries. Pages on the flash card are organized as a *heap file*, which is naturally ordered by time. Note that a flash card can only hold up to m pages ($o_{i0}..o_{im}$) and hence the available memory is organized as a circular array, in which the newest o_{ij} pair replaces the oldest o_{ij} pair if the memory becomes full.
i) Random Access By Value: An example of such operation is to locally load the records that have a temperature of 95F. In order to fetch records by their value we have implemented a *static* hash index with a swap directory that gracefully keeps in memory the directory buckets with the highest hit ratio. We use a static index as opposed to a dynamic hashing index, such as *extendible* or *linear*, because the latter structures are considerably more power demanding (i.e. due to page splits during insertions).
ii) Sorted Access By Value: An example of such operation is to locally load the records that have a temperature between 94F-96F. An important observation is that sensor readings are numeric readings in a discrete range (for example the temperature is between -40F and 250F). In order to enable such range queries, we currently use an extension of our random-access index in which we query every discrete value in the requested range. However, we are also developing a simple B+ tree index, which is a minimalistic version of its counterpart found in a real database system. It consists of a small number of non-leaf index pages which provide pointers to the leaf pages. In our current design, we keep a small number of highly used non-leaf index pages (such as the root) in main memory.

4.3 Efficient Top-k Query Evaluation in RISE

We now sketch a *Threshold Join Algorithm* which is an efficient top-k query processing algorithm for sensor networks. In the naive case, such queries could be answered by transferring all sensor values to the sink and then find the correct result. In our algorithm, we use an additional probing and filtering phase in order to eliminate this expensive step. More specifically, we use the following phases:



v1	v2	v3	v4	v5
o3,.99	o1,.91	o1,.92	o3,.74	o3,.67
o1,.66	o3,.90	o3,.75	o1,.56	o4,.67
00,.63	00,.61	04,.70	02,.56	01,.58
02,.48	04,.07	02,.16	00,.28	02,.54
04,.44	02,.01	00,.01	04,.19	00,.35

Fig. 2. The QST for two phases of the TJA Algorithm (the third phase is omitted as it does not contribute to the final result). The table on the right shows the objects qualifying in each phase.

1) the *Lower Bound* phase, in which the sink collects the union of the top-k results from all nodes in the network (denoted as L_{sink}=$\{l_1, l_2, \ldots, l_o\}$, $o \geq k$),
2) the *Hierarchical Joining* phase, in which each node uses L_{sink} for eliminating anything that has a value below the least ranked item in L_{sink},
3) the *Clean-Up* phase, in which the actual top-k results are identified.

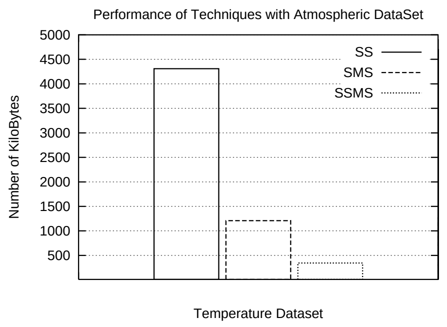
In figure 2, we can see the execution for the two initial phases of the algorithm. In the illustration, each node $[v_1..v_5]$ is assumed to have a local rank of five objects $[o_1..o_5]$ and the nodes are interconnected in a tree topology. The illustration shows that the sink requires only to fetch the objects above the lower line, which represents the execution of the second phase of our algorithm.

5 Experimental Evaluation

We have tested our top-k algorithm in a Peer-to-Peer network using a real dataset of temperature measurements collected at 32 sites in Washington and Oregon.[3] Each site (node) maintained the average temperature on an hourly basis for 208 days between June 2003 and June 2004 (i.e. 4990 time moments), and our query was to find the 10 moments at which the average temperature was the highest. Our algorithm uses our local access methods to execute efficiently. In Figure 3a we compare our approach with the *SS* approach (sending all data over the network), and our results indicate that *SS* consumes an order of magnitude more network bytes than the *SSMS* approach. We also compare our approach with a simpler approach that computes the scores of all tuples in the network, combining partial results as data is transmitted to the sink. This approach does not use any index methods, and can be implemented in the *SMS* framework. Our preliminary results show that our approach significantly outperforms this technique.

We have also calculated the energy gains that can be achieved by using the sense and store methodology by measuring the energy consumption of storing

[3] http://www-k12.atmos.washington.edu/k12/grayskies/ .



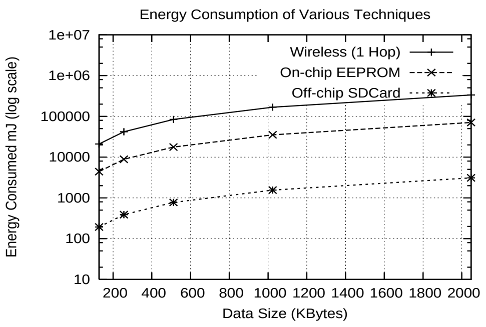


Fig. 3. a) Number of bytes transmitted in the *SS*, *SMS*, and *SSMS* models using atmospheric data. b) A comparison of the amount of energy expended to transfer data via the wireless interface vs Storing it on the on-chip EEPROM and the off-chip SD-Card.

data locally on flash card as opposed to blindly sending it over the wireless network. Specifically, we used the RISE mote to measure the cost of transmitting packages of various sizes over a 9.6Kbps radio (at 60mA) and storing the respective data locally on flash (see figure 3b). In the case of 512B (one page), we found that it takes on average $416ms$ or 82,368μJ . Comparing this with the 763.12μJ required for writing the same amount of data to local flash, along with the fact that transmission of one byte is roughly equivalent to executing 1120 CPU instructions, makes local storage and processing highly desirable.

6 Related Work

There has been a lot of work in the area of query processing, in-network aggregation and data-centric storage in sensor networks. To our knowledge, our work is the first that addresses issues related to in-situ data storage in sensor devices with large memories.

Systems such as TinyDB[7] and Cougar[11] achieve energy reduction by pushing aggregation and selections in the network rather than processing everything at the sink. Both approaches propose a declarative approach for querying sensor networks. These systems are optimized for sensor nodes with limited storage and relatively short-epochs, while our techniques are designated for sensors with larger external flash memories and longer epochs. Note that in TinyDB users are allowed to define fixed size materialization points through the `STORAGE POINT` clause. This allows each sensor to gather locally in a buffer some readings, which cannot be utilized until the materialization point is created in its entirety. Therefore even if there was enough storage to store MBs of data, the absence of efficient access methods makes the retrieval of the desired values quite expensive.

A large number of flash-based file systems have been proposed in the last few years, including the Linux compatible Journaling Flash File System (JFFS and JFFS2)[15], the Yet Another Flash File System (YAFFS)[16] specifically designed for NAND flash with it being portable under Linux, uClinux, and Windows CE. The first file system for sensor nodes was Matchbox and this is provided as an integral part of the TinyOS [5] distribution. Recently the Efficient Log Structured Flash File System (ELF)[2] shows that it offers several advantages over Matchbox including higher read throughput and random access by timestamp. However the main job of a file system is to organize the sectors of the storage media into files and directories and to determine whether these are being used or not. Filenames are then accessible by their unique identifier (such as an inode). Therefore a filesystem does not support retrieving records by their value as we do in our approach.

An R-tree and B-Tree index structure for flash memory on portable devices, such as PDA's and cell phones, has been proposed in [13] and [14] respectively. These structures use an in-memory address translation table, which hides the details of the flash wear-leveling mechanism. However, such a structure has a very large footprint (3-4MB) which constitutes it inapplicable in our context.

In *Data Centric Routing (DCR)*, such as directed diffusion [6], low-latency paths are established between the sink and the sensors. Such an approach is supplementary to our framework. In Data Centric Storage (DCS) [10] data with the same name (e.g. humidity readings) is stored at the same node in the network, offering therefore efficient location and retrieval. However the overhead of relocating the data in the network will become huge if the network generates many MBs of GBs of data. Finally, local compression techniques, such as the one proposed in [3], would improve the efficiency of our framework and their investigation will be a topic of future research.

7 Conclusions

In this paper we discussed many of the data management issues that arise in the context of the *RISE* sensor network platform. In RISE, sensors feature a large memory which creates a new paradigm for power conservation in long epoch applications. We believe that many applications can benefit from a large local storage, as such storage can be used for local aggregation or compression before transmitting the results towards the sink. We expect that this in addition with the provisioning of efficient access methods will also provide a powerful new framework to cope with new types of queries, such as temporal or top-k, that have not been addressed adequately to this date. In the future we plan to investigate the effectiveness of our framework in field experiments which will be conducted in conjunction with the Center of Conservation Biology at UC-Riverside.

References

1. Considine J., Li F., Kollios G., Byers J., "Approximate Aggregation Techniques for Sensor Databases". In *ICDE'04*, Boston, MA, 2004.
2. Dai H., Neufeld M., Han R., "ELF: an efficient log-structured flash file system for micro sensor nodes", In *SenSys'04*, Baltimore, MD, 2004.

3. Deligiannakis A., Kotidis Y., Roussopoulos N. ”Compressing historical information in sensor networks”, In *SIGMOD'04*, Paris, France, 2004.
4. Fagin R., ”Fuzzy Queries In Multimedia Database Systems”, In *PODS'98*.
5. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K.. “System architecture directions for network sensors”, In *ASPLOS'00*, Cambridge, MA, 2000.
6. Intanagonwiwat C., Govindan R. Estrin D. ”Directed diffusion: A scalable and robust communication paradigm for sensor networks”, In *MobiCOM'00*.
7. Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., ”The Design of an Acquisitional Query Processor for Sensor Networks”, In *SIGMOD'03*.
8. Madden S.R., Franklin M.J., Hellerstein J.M, Hong W.. ”TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks”, In *OSDI'02*, Boston, MA, 2002.
9. Sadler C., Zhang P., Martonosi M., Lyon S., “Hardware Design Experiences in ZebraNet”, In *Sensys'04*, Baltimore, MD, 2004.
10. Shenker S., Ratnasamy S., Karp B., Govindan R., Estrin D., ”Data-centric storage in sensornets”, ACM SIGCOMM Computer Communication Review, 2003.
11. Yao Y., Gehrke J.E., ”Query Processing in Sensor Networks”, In *CIDR'03*, Asilomar, CA, 2003.
12. Xu N., Rangwala S., Chintalapudi K., Ganesan D., Broad A., Govindan R. and Estrin D., “A Wireless Sensor Network for Structural Monitoring”, In *Sensys'04*, Baltimore, MD, 2004.
13. Wu C-H., Chang L-P., Kuo T-W., “An Efficient R-Tree Implementation Over Flash-Memory Storage Systems”, In *ACM GIS'03*, New Orleans, Louisiana, 2003.
14. Wu C-H., Chang L-P., Kuo T-W., “An Efficient B-Tree Layer for Flash Memory Storage Systems”, In *RTCSA'03*, Tainan, Taiwan, 2003.
15. Woodhouse D. “JFFS : The Journalling Flash File System” Red Hat Inc., Available at: `http://sources.redhat.com/jffs2/jffs2.pdf`
16. Wookey “YAFFS - A filesystem designed for NAND flash”, In *Linux 2004*.
17. Szewczyk R., Mainwaring A., Polastre J., Anderson J., Culler D., “An Analysis of a Large Scale Habitat Monitoring Application”, In *SenSys'04*, Baltimore, Maryland, Nov 2004.
18. Crossbow Technology Inc. http://www.xbow.com/