The MicroPulse Framework for Adaptive Waking Windows in Sensor Networks

Demetrios Zeinalipour-Yazti, Panayiotis Andreou, Panos K. Chrysanthis[‡],
George Samaras, Andreas Pitsillides

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,cspitsil}@cs.ucy.ac.cy

***Abstract*—In this paper we present *MicroPulse*, a novel framework for adapting the waking window of a sensing device S based on the data workload incurred by a query Q. Assuming a typical tree-based aggregation scenario, the *waking window* is defined as the time interval τ during which S enables its transceiver in order to collect the results from its children. Minimizing the length of τ enables S to conserve energy that can be used to prolong the longevity of the network and hence the quality of results. Our method is established on profiling recent data acquisition activity and on identifying the bottlenecks using an in-network execution of the Critical Path Method. We show through trace-driven experimentation with a real dataset that MicroPulse can reduce the energy cost of the waking window by three orders of magnitude.**

***Index Terms*—Critical Path Method, Waking Window, Scheduling, Sensor Networks.**

I. Introduction

Recent advances in embedded computing have made it feasible to produce small scale sensors, actuators and processors that can be used for ad-hoc deployments of environmental monitoring infrastructures [13], [5], [9]. The longevity of a Wireless Sensor Network (WSN) heavily relies on the existence of power-efficient algorithms for the acquisition, aggregation and storage of the sensor readings.

Communicating over the radio in a WSN is the most energy demanding factor among all other functions, such as storage [17] and processing [9]. The energy consumption for transmitting 1 bit of data using the MICA mote is approximately equivalent to processing 1000 CPU instructions [9]. One way to cope with the energy challenge is to power down the radio transceiver during periods of inactivity. In particular, it has been shown that sensors operating at a 2% duty cycle can achieve lifetimes of 6-months using two AA batteries [8].

The continuous interval during which a sensor node S enables its transceiver, collects and aggregates the results from its children, and then forwards them all together to its own parent is defined as the *waking window* τ. Note that τ is continuous because it would be very energy-demanding to suspend the transceiver more than once during the interval of an *epoch*, which specifies the amount of time that sensors have to wait before re-computing a continuous query.

It is important to mention that the exact value of τ is query-specific and can not be determined accurately using current techniques. For instance a sensor does not know in advance how many tuples it will receive from its children. Choosing the correct value for τ is a challenging task as any wrong estimate might disrupt the synchrony of the query routing tree.

The objective of this work is to automatically tune τ, locally at each sensor without any a priori knowledge or user intervention. Note that in defining τ we are challenged with the following trade-off:

- **Early-Off Transceiver:** Shall S power-off the transceiver too early reduces energy consumption but also increases the number of tuples that are not delivered to the *sink*, the root of the routing tree. As a result the sink will generate an erroneous answer to the query Q; and
- **Late-Off Transceiver:** Shall S keep the transceiver active for too long decreases the number of tuples that are lost due to powering down the transceiver too early but also increases energy consumption. Thus, the network will consume more energy than necessary which is not desirable given the scarce energy budget of each sensor.

In this paper we present *MicroPulse*, a novel framework for adapting the waking window of a sensing device S based on the data workload incurred by a query Q. Our ideas are established on profiling recent data acquisition activity and on identifying the bottlenecks using an in-network execution of the Critical Path Method.

The *Critical Path Method (CPM)* [10] is a graph-theoretic algorithm for scheduling project activities. It is widely used in project planning (construction, product development, plant maintenance, software development and research projects). The core idea of CPM is to associate each project milestone with a vertex v and then define the dependencies between the given vertices using *activities*. For instance, the activity $v_i \Leftarrow v_j$ denotes that the completion of v_i depends on the completion of v_j. Each activity is associated with a weight (denoted as $\overset{weight}{\Leftarrow}$) which quantifies the amount of time that is required to complete v_i assuming that v_j is completed. The critical path allows us to define the minimum time or otherwise the maximum path that is required to complete a project (i.e.,

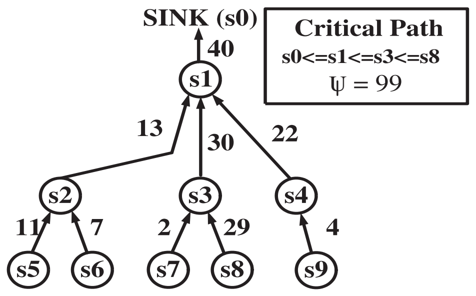

Fig. 1. The figure illustrates nine sensing devices (shown as vertices) and the respective time cost (shown as edges) to transfer the results from an arbitrary and continuous query Q to the sink (s_0). MicroPulse utilizes this information in order to locally adapt the waking window of each sensor using the *Critical Path Method*.

milestone v_0). Any delay in the activities of the critical path will cause a delay for the whole project.

In order to adapt the discussion to a sensor network context assume that each sensor s_i is represented by a CPM vertex. More formally, we map each s_i to the elements of the vertex set $V = \{v_1, v_2, \ldots, v_n\}$ using a 1:1 mapping function $f : s_i \rightarrow v_i$, $i \leq n$. Also, let the descendent-ancestor relations of the sensor network be denoted as edges in this graph.

An example with 9 sensing devices $\{s_1, \ldots, s_9\}$ is illustrated in Figure 1. The weights on the edges define the required time to propagate the query results between the respective pairs. It is easy to see that the total time to answer the query at the sink in the given network is at least $\psi = 99$, since the critical path is $s_0 \overset{40}{\Leftarrow} s_1 \overset{30}{\Leftarrow} s_3 \overset{29}{\Leftarrow} s_8$.

Having this information at hand enables the scheduling of transmission between sensors. In particular, sensor s_i can be scheduled to wake-up and transmit at the following deadlines (w_i): $w_1 = \psi - 40 = 59$, $w_2 = w_1 - 13 = 46$, $w_3 = w_1 - 30 = 29$, $w_4 = w_1 - 22 = 37$ while s_0 and s_1 will be listening for these transmissions during the intervals τ_0=[59..99) and τ_1=[29..59) respectively. The same intuition also applies for the leaf nodes, e.g., s_5 starts transmission at $w_5 = w_2 - 11 = 35$ and s_2 listens for this transmission in the range τ_2=[35..46). Additionally, the critical path enables a sensor x to estimate the interval during which its parent y will have its transceiver enabled. This is very useful because in the subsequent epochs and under a different workload, x can find out if it can deliver the new workload without first asking y to adjust its waking window.

It should be noted that the edges in Figure 1 have different weights. This is very typical for a sensor network as the link quality can vary across the network [13]. Another reason is that some sensors might return more tuples than other sensors.

Note that our scheduling scheme is distributed which makes it fundamentally different from centralized scheduling approaches like DTA [16] and TD-DES [1] that generate collision-free query plans at a centralized node. Additionally, our approach is also different from techniques such as [12] which segment the sensor network into sectors in order to minimize collisions during data acquisition.

Our Contributions

In this paper we make the following contributions:

- We formulate the problem of adapting the waking window τ of a sensing device in order to conserve energy that can be used to prolong the longevity of the network and hence the quality of results.
- We solve the waking window problem by combining a custom profiling structure and the critical path method.
- We experimentally validate the efficiency of our solution using real sensor dataset from Intel Research [6].

The remainder of the paper is organized as follows: Section II studies the waking window mechanism of popular data acquisition systems and discusses the advantages and shortages for each of these systems. Section III presents the underlying algorithms of the MicroPulse Framework. Section IV presents the experimental study using a trace-driven simulator and Section V concludes the paper.

II. Background and Related Work

In this section we study the waking window mechanism of the two most popular declarative acquisition frameworks: TAG [9], [8] and Cougar [15]. We start out our description by assuming that the query Q has been disseminated to n sensors.

Tiny Aggregation (TAG): In this approach, the epoch e is divided into a number of fixed-length time *intervals* $\{e_1, e_2, \ldots, e_d\}$, where d is the depth of the routing tree, rooted at the sink, that conceptually interconnects the n sensors. The core idea of this framework is summarized as follows: *"when nodes at level i+1 transmit then nodes at level i listen"*. More formally, a sensor s_i enables its transceiver at chronon $w_i = \lfloor e/d \rfloor * (d - depth(s_i))$ and keeps the transceiver active for $\tau_i = \lfloor e/d \rfloor$ chronons. Note that $\sum_{i=d}^{0}(e_i)$ provides a lower-bound on e, thus the answer will always arrive at the sink before the end of the epoch. Setting e as a prime number ensures the following inequality $\sum_{i=d}^{0}(e_i) < e$, which is desirable given that the answer has to reach the sink at chronon e.

For instance, if the epoch is 31 seconds and we have a three-tiered network (i.e., d=3) like Figure 2 (top, left), then the epoch is sliced into three segments {10,10,10}. During interval [0..10), nodes at level 3 will transmit while nodes at level 2 will listen; during interval [10..20) level 2 nodes transmit while level 1 nodes listen; and finally during [20..30), level 1 nodes transmit and the sink (level 0) listens. Thus, the answer will be ready prior the completion of chronon 31 which is the end of the epoch.

The parent wake-up window τ is an over-estimation (in the above example 10 seconds!) of the actual time that is required to transmit between the children and a parent . The rationale behind this over-estimation is to offset the limitations in the quality of the clock synchronization algorithms [9] but in reality it is too coarse. In the experimental Section IV, we found that this over-estimation is three orders of magnitude larger than necessary.

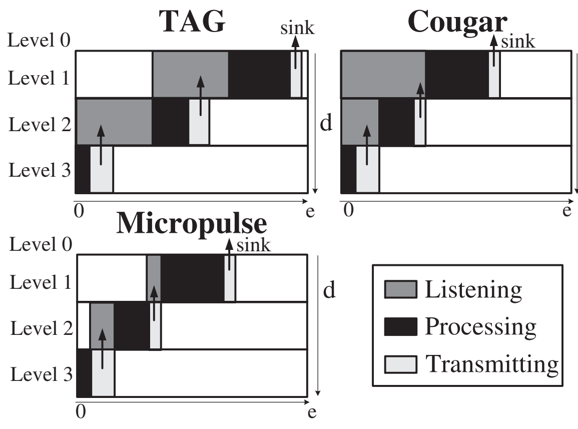

Fig. 2. The Waking (Listening) Window (τ) in TAG, Cougar and MicroPulse.

Additionally, it is not clear how τ is set under a *variable workload* which occurs under the following circumstances: i) from a *non-balanced topology*, where some nodes have many children and thus require more time to collect the results from their dependents; and ii) from *multi-tuple answers*, which are generated because some nodes return more tuples than other nodes (e.g. because of the query predicate).

The MicroPulse framework presented in this paper gracefully handles both cases of variable workload by utilizing the Critical Path Method. Our framework, like TAG, utilizes the TinyOS MAC layer [14] to handle the collisions that will occur if nodes in the same vicinity transmit during the same interval.

Cougar: In this approach, each sensor maintains a *waiting list* that specifies the children for each node. Such a list can be constructed by having each child explicitly acknowledging its parent during the query dissemination phase. Having the list of children enables a sensor to shut down its transceiver as soon as all children have answered. This yields a set of non-uniform waking windows $\{\tau_1, \tau_2, \ldots\}$ as opposed to TAG where we have a single τ which is uniform for all sensors (i.e., $\lfloor e/d \rfloor$).

The main drawback of Cougar is that a parent node has to keep its transceiver active from the beginning of the epoch until all children have answered. In particular, it holds that $\tau_i > \tau_j$ if $depth(v_i) < depth(v_j)$. In order to cope with children sensor that may not respond, Cougar deploys a timeout h. To understand the drawback of Cougar consider Figure 2 (top, right), where level 2 and level 1 nodes have activated their transceivers at chronon zero and wait for the leaf nodes to respond. If we have a failure at any given node x, then each node on the path $x \rightarrow \ldots \rightarrow s_0$ (sink), has to keep its radio active for h additional seconds.

III. The MicroPulse Framework

In this section we describe the underlying algorithms of the MicroPulse Framework. We divide our description in the following three conceptual phases:

1) *Construction Phase*, executed once prior the execution of Q, during which the sink constructs the routing tree and becomes aware of the critical path cost ψ.
2) *Pulse Phase*, executed once prior the execution of Q, during which each sensor s_i tunes its wake-up chronon w_i and waking window τ_i according to the value ψ.
3) *Adaptation Phase*, executed when a topology or workload change occurs which results in a new critical path.

A. The Critical Path Construction Phase

This phase starts out by having each node s_j select one node s_i as its parent. This results in a *waiting list* similarly to Cougar [15]. To accomplish this task, the parent s_i is notified through an explicit acknowledgement or becomes aware of s_j's decision by snooping the radio. Note that in both TAG [9] and Cougar [15] nodes select as their parent whichever sensor forwarded the query first. Alternatively, nodes could have chosen as their parent the neighbor with the smallest hop count from the sink or the one with the highest signal strength. In more recent frameworks, like GANC [11] and Multi-Criteria Routing [7], sensors select their parents based on query semantics, power consumption, remaining energy and others. In more unstable topologies a node can maintain several parents [4] in order to achieve fault tolerance but this might impose some limitations on the type of supported queries. Nevertheless, all these alternatives are supplementary to this step.

In the next step, we profile the activity of the incoming and outgoing links, and then propagate this information towards the sink. In particular, we execute one round of data acquisition where each sensor s_i maintains one counter for its parent connection (denoted as s_i^{out}) and one counter per child connection (denoted as $s_{i,j}^{in}$), where j denotes the identifier of the child. These counters measure the *time* that is required to transmit the tuples between the respective sensors and will be utilized as indicators of the link workload in the subsequent epochs. Note that these counters account for more time than what is required had we assumed a collision-free MAC channel. Additionally, it is important to mention that we could have deployed a more complex structure rather than the counters s_i^{out} and $s_{i,j}^{in}$. That would allow a sensor to obtain a better statistical indicator of the link activity. By projecting the time costs obtained for each edge to a virtual spanning tree creates a distributed structure, similar to the one depicted in Figure 1.

The final step is to percolate these local edge costs to the sink by recursively executing the following in-network function at each sensor s_i:

$$\psi_i = \begin{cases} 0 & \text{if } s_i \text{ is a leaf node,} \\ max_{\forall j \in children(s_i)}(\psi_j + s_{i,j}^{in}) & \text{otherwise.} \end{cases}$$

The critical path cost is then ψ_0 (denoted for brevity as ψ). Using our working example of Figure 1, we will end up with the following values : $\psi_{5 \leq i \leq 9} = 0$, $\psi_4 = 4$, $\psi_3 = 29$, $\psi_2 = 11$, $\psi_1 = 59$ and $\psi_0 = 99$.

B. The Pulse Phase

In this phase we shape the waking window τ_i and the wake-up chronon w_i of each sensor by disseminating ψ, constructed during phase 1, in the network. Algorithm 1 presents the main steps of this procedure which has a message complexity of

Algorithm 1 : **MicroPulse: The Pulse Phase**

Input: n sensing devices $\{s_1, s_2, \ldots, s_n\}$ and the sink s_0, the Critical Path cost ψ, the epoch e.

Output: A set of n waking windows $\{\tau_1, \tau_2, \ldots, \tau_n\}$ and n wake-up times $\{w_1, w_2, \ldots, w_n\}$.

Execute these steps beginning from s_0 (top-down):

1) If $\psi > e$ then abort *"The Critical Path is larger than the Epoch"*.
2) Find the maximum $s^{in}_{i,j}$ in s_i's children and denote the identifier of this sensor as $maxchild$. Now calculate the waking time w_i as follows:

$$w_i = \psi - s^{in}_{i,maxchild} - a - b - c, \quad (1)$$

where a, b and c are three variables which offset the costs of *processing*, *the inaccurate clock* and *collisions at the MAC layer*. The waking window is the interval:

$$\tau_i = [w_i..(w_i + s^{in}_{i,maxchild}))) \quad (2)$$

3) Now disseminate ψ to s_i's children. Upon receiving ψ, each child node s_j decreases ψ locally, as follows:

$$\psi = \psi - s^{out}_j \quad (3)$$

4) At the same time with step 3, disseminate $s^{in}_{i,maxchild}$ to s_i's children. This information, will be useful to define the *latency tolerance* (λ_i) of s_i in the next adaptation phase.
5) Repeat steps 2-4, recursively until all sensors in the network have set w_i and τ_i respectively ($i \leq n$).

$O(n)$. At the end of the algorithm execution each sensor knows exactly when it should wake up (i.e., w_i) and for how long (i.e., τ_i).

To facilitate our presentation assume that the *processing*, *the inaccurate clock* and *collisions at the MAC layer costs*, denoted as a, b and c respectively, are all equal to zero. Executing Algorithm 1 on the example of Figure 1 which has a ψ equal to 99 along with an epoch e of 100 yields the following triples (s_i, w_i, τ_i): { $(s_0, 59, [59..99))$, $(s_1, 29, [29..59))$, $(s_2, 46, [46..59))$, $(s_3, 29, [29..59))$, $(s_4, 37, [37..59))$, $(s_5, 35, [35..46))$, $(s_6, 39, [39..46))$, $(s_7, 27, [27..29))$, $(s_8, 0, [0..29))$, $(s_9, 33, [33..37))$ }

C. Adaptation Phase

In this section we summarize the main ideas behind the adaptation phase which adjusts the critical path in cases of workload or topology changes.

To motivate our discussion, consider the scenario where a child in round (r+1) requires more time to deliver the results to its parent than in round r (i.e., an increased workload). In the worst case this might require a complete reconstruction of the critical path cost. Fortunately, the structures deployed by MicroPulse enable a child to know the interval during which its parent will have an active transceiver. Therefore, the given child may be able to start delivering the workload earlier, if so, succeeding in completing the transmission on-time.

In particular, each sensor s_i knows the maximum incoming edge of its parent X from step 4 of Algorithm 1, denoted as $s^{in}_{X,maxchild}$. The child s_i then calculates the *latency tolerance*[1] λ_i of its parent X as follows:

$$\lambda_i = s^{in}_{X,maxchild} - s^{out}_i \quad (4)$$

Note that λ_i provides a sensor s_i with the maximum leeway from the workload indicator s^{out}_i. For instance in Figure 1, s_7 calculates $\lambda_7 = 29 - 2 = 27$. Thus, s_7's workload can increase by 27 chronons without affecting the synchrony of the query routing tree. The same also applies to the opposite case, where we have a decrease in the workload s^{out}_i. If s_i is not on the critical path then there is absolutely no consequence on efficiency. If on the other hand, s_i is on the critical path then a decrease in s^{out}_i might result in some waste of energy as s_i's parent will have a larger τ than necessary.

In order to cope with such cases and the fact that $s^{in}_{X,maxchild}$ might change or λ_i might become negative, we reconstruct and re-pulse the critical path in the network periodically after a certain number of changes, failures or collisions. However, these updates represent the minority of the cases as our framework is tailored for stationary wireless sensor networks where the critical path will not change frequently. In the future we plan to devise more elaborate algorithms to cope with the adaptation under more dynamic environments.

IV. Experimental Evaluation

In this section we describe our experimental methodology and the results of our evaluation.

A. Experimental Methodology

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

Algorithms: We have implemented the *TAG*, *Cougar* and *MicroPulse* waking window algorithms. We utilize a failure rate of 20% where a sensor has a probability of 0.2 to not participate in a given epoch. We set the child waiting timer h to 200ms.

Sensing Device: We use the energy model of Crossbow's new generation TelosB [2] 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, $1.8mA$ in active mode with the radio off and $5.1\mu A$ in sleep mode. Our performance measure is *Energy*, in *Joules*, that is required at each discrete chronon 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 : $1.8V \times 23 * 10^{-3}A \times 30 * 8bits/250kbps = 39\mu J$.

[1]The *latency tolerance* is often also referred to as *slack*.

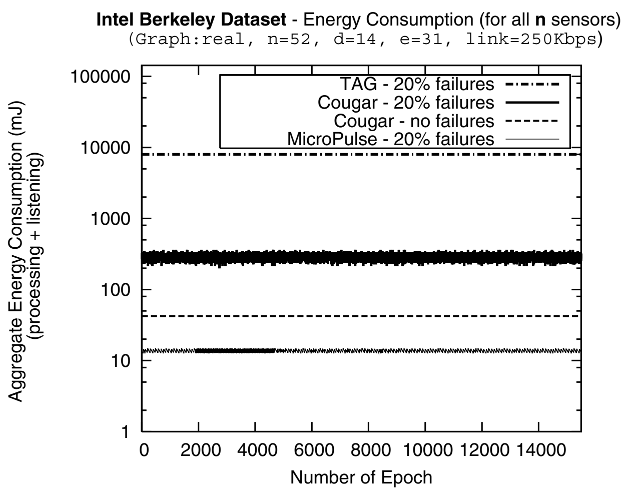

Fig. 3. Energy Consumption: Comparing the aggregate energy cost of the three algorithms using the TelosB energy model.

Dataset: We utilize a real dataset from Intel Berkeley Research [6]. This dataset contains data that is collected from 58 sensors deployed at the premises of the Intel Research in Berkeley between February 28th and April 5th, 2004. The motes utilized in the deployment were equipped with weather boards and collected time-stamped topology information along with humidity, temperature, light and voltage values once every 31 seconds (i.e., the epoch). The dataset includes 2.3 million readings collected from these sensors. We use readings from the 52 sensors that had the largest amount of local readings since some of them had many missing values. Our query is *"SELECT moteid, temp FROM sensors"*. The depth of the query spanning tree is 14.

B. Energy Consumption

In order to assess the efficiency of the MicroPulse algorithm we measure the energy that is spent on the waking window for the n sensors in isolation from the rest components (flash, weather board, etc.). In particular, we measure the time that is spent in each MicroPulse phase and then multiply this by the respective current and voltage. Figure 3 shows that the aggregate energy consumption for the 52 sensors under TAG, requires 7,984mJ. This is three orders of magnitude more than what the MicroPulse algorithm requires (13.75±0.58mJ). The big difference in performance is clearly attributed to the uniform size of τ in TAG which is $\approx$ 2.21 seconds (i.e., 31 (seconds) / 14 (depth)), while in MicroPulse τ is only $146ms$ on average.

The same figure also shows that the Cougar algorithm requires on average 288.97±24.42mJ which is one order of magnitude more than the energy required by MicroPulse. The disadvantage of the Cougar algorithm originates from the fact that the parents keep their transceivers enabled until all the children have answered or until the local timer h has expired (in cases of failures). Thus, any failure is automatically translated into a chain of delayed waking windows all of which consume more energy than necessary. This is shown by the third line in which Cougar under no failures requires only $42mJ$ which is only the 14% of what Cougar requires under failures. It is interesting to highlight that MicroPulse maintains a competitive advantage even over Cougar (no failures).

V. Conclusions and Future Work

This paper studies the problem of optimizing the length of the waking window in sensor networks in order to conserve energy. Our ideas are established on profiling recent data acquisition activity and on identifying the bottlenecks using the Critical Path Method. We have described the core ideas of our framework and some preliminary experimental results using our trace-driven simulator with real datasets from Intel Research. We found that MicroPulse offers tremendous energy reductions under realistic conditions. In the future we plan to devise efficient adaptation algorithms which will be triggered in cases of variable workload between consecutive chronons.

Acknowledgements: 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).

References

[1] Cetintemel U., Flinders A., Sun Y., "Power-efficient data dissemination in wireless sensor networks", In *ACM MOBIDE*, 2003.
[2] Crossbow Technology, Inc. http://www.xbow.com/
[3] Hill J., Szewczyk R., Woo A., Hollar S., Culler D., Pister K., "System Architecture Directions for Networked Sensors", In ASPLOS 2000.
[4] Considine J., Li F., Kollios G., and Byers J. "Approximate Aggregation Techniques for Sensor Databases", In IEEE ICDE 2004.
[5] Intanagonwiwat C., Govindan R. Estrin D. "Directed diffusion: A scalable and robust communication paradigm for sensor networks", In ACM MOBICOM 2000.
[6] Intel Lab Data http://db.csail.mit.edu/labdata/labdata.html
[7] Li Q., Beaver J., Amer A., Chrysanthis P., Labrinidis A. "Multi-Criteria Routing in Wireless Sensor-Based Pervasive Environments", In Pervasive Computing 2005.
[8] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "The Design of an Acquisitional Query Processor for Sensor Networks", In ACM SIGMOD 2003.
[9] Madden S.R., Franklin M.J., Hellerstein J.M., Hong W., "TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks", In OSDI 2002.
[10] Gross J.L, and Yellen J., "Graph Theory & Its Application", by Chapman & Hall/CRC Press, ISBN: 158488505X, 2005.
[11] Sharaf A.M., Beaver J., Labrinidis A., Chrysanthis P.K., "Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks", In VLDB 2004.
[12] Sharma D. and Zadorozhny V.I. and Chrysanthis P.K, "Timely data delivery in sensor networks using whirlpool", In DMSN 2005.
[13] Szewczyk R., Mainwaring A., Polastre J., Anderson J., Culler D., "An Analysis of a Large Scale Habitat Monitoring Application", In ACM SenSys 2004.
[14] Woo A., Culler D.E. "A transmission control scheme for media access in sensor networks", In MOBICOM 2001.
[15] Yao Y., Gehrke J.E., "Query Processing in Sensor Networks", In CIDR 2003.
[16] Zadorozhny V. and Chrysanthis P.K and Labrinidis A., "Algebraic Optimization of Data Delivery Patterns in Mobile Sensor Networks", In DEXA 2004.
[17] Zeinalipour-Yazti D., Lin S., Kalogeraki V., Gunopulos D., Najjar W., "MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices", In USENIX FAST'2005.