Developing and Deploying Sensor Network Applications with AnduIN*

Daniel Klan, Katja Hose, Kai-Uwe Sattler
Department of Computer Science & Automation
Ilmenau University of Technology, Germany
{*first.last*}@tu-ilmenau.de

ABSTRACT

Wireless sensor networks have become important architectures for many application scenarios, e.g., traffic monitoring or environmental monitoring in general. As these sensors are battery-powered, query processing strategies aim at minimizing energy consumption. Because sending all sensor readings to a central stream data management system consumes too much energy, parts of the query can already be processed within the network (in-network query processing). An important optimization criterion in this context is where to process which intermediate results and how to route them efficiently. To overcome these problems, we propose AnduIN, a system addressing these problems and offering an optimizer to decide which parts of the query should be processed in-network. It also considers optimization with respect to more complex data analytics task such as burst detection. Furthermore, AnduIN offers a Web-based frontend for declarative query formulation and deployment. In this paper, we mainly focus on deployment and discuss AnduIN's components alleviating deployment and usability.

1. INTRODUCTION

Wireless sensor networks (WSN) have evolved to a powerful infrastructure for monitoring in many applications, ranging from supply chain monitoring in enterprises to large-scale environmental monitoring. In most cases, these applications are not restricted to sensor nodes but also contain a global data processing component performing more complex processing and analytics tasks. A valuable tool for implementing this component are data stream management systems (DSMS), which simply take the sensor network as input source.

However, developing and deploying a WSN application is still a complex and time-consuming task. Some important challenges are handling continuously generated data, accessing sensor (data) in case of heterogenous sensors, and dealing with limitations of the sensor hardware in terms of limited computing power, memory, and battery lifetime as well as the characteristics of WSN such as economical sleep time, expensive radio communication, and ad-hoc routing.

To address some of these challenges, several middleware approaches have been developed in the past couple of years, e.g., TinyDB [19], Cougar [20], and GSN [1]. Although these approaches simplify the development of sensor-based applications, they either do not support more complex data processing tasks or lack a declarative paradigm of query formulation and deployment. Thus, we argue that an approach is needed that resembles the idea of distributed databases: supporting a declarative, query-oriented way of formulating data processing tasks (including operations appropriate for processing sensor data) and performing the necessary query decomposition and query shipping automatically and transparently. In particular the latter is a difficult task in WSN: we have to take into account the limited capabilities of the individual nodes requiring, for example, code generation and over-the-air programming, the aspect of ad-hoc routing to exchange data between nodes as well as the goal of optimizing the nodes' power consumption in order to increase network lifetime.

In this paper, we present AnduIN – a system addressing these issues. AnduIN is a system combining DSMS (data stream management system) and INQP (in-network query processing) functionalities, which consists of

- a query processor for data streams supporting complex data mining operations in addition to basic processing operators such as filters, joins, and aggregates,
- a sensor node component, i.e., a library of operators for placing on the sensor nodes,
- a Web-based box-and-arrows frontend for query formulation and deployment,
- an optimizer component as part of the query processor for decomposing processing tasks between query processor and sensor nodes based on the optimization of energy consumption.

The contribution of the work presented in this paper is twofold:

1. a flexible declarative approach for specifying the data processing part of a WSN application which
2. combines the in-network query processing paradigm with data stream processing.

We argue that such a combination allows to exploit the advantages of both paradigms (Figure 1): supporting complex data processing tasks (DSMS) as well as higher autonomy and energy awareness (INQP).

*This work was in part supported by the BMBF under grant 03WKBD2B.

DMSN'09, August 24, 2009, Lyon, France. Copyright @ 2009 ACM 978-1-60558-284-9/08/08... $5.00.

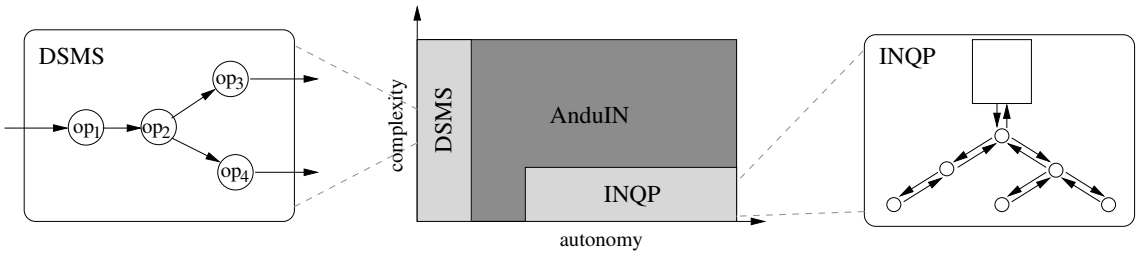

Figure 1: Classification

The remainder of this paper is organized as follows. After a brief review of related work in this area in Section 2, we present the overall architecture of AnduIN in Section 3. Next in Section 4, we present the data definition extensions we made to CQL in order to specify sensor node and network specific information as well as the steps of query decomposition. In Section 5 we discuss an example illustrating query decomposition and deployment using AnduIN. We conclude the paper in Section 6.

2. RELATED WORK

Usually, sensors are battery-powered and thus have a limited lifetime. In order to increase this lifetime, the literature proposes several approaches that try to minimize energy consumption. Among them, TinyDB [19], Cougar [20], and GSN [1] are the most prominent examples. For a sensor, communication with other sensors or a base station is the most expensive operation with respect to energy consumption [8]. Thus, the basic strategy of sending all sensor readings to a base station and processing the data there should be avoided. Therefore, it is worthwhile to perform some query operations (aggregations) already within the network (in-network processing). For this purpose, sensors need to know routing paths [9] on which to propagate readings and to perform aggregations. Such routing paths can be found by negotiation applying diffusion – examples are SPIN [16] and directed diffusion [13]. Routing paths can be organized as chains (PEGASIS [17]), trees (TinyDB [19], TAG [18]), or clusters (LEACH [11, 12], Cougar [20]).

By applying the concept of routing trees to perform aggregations, sensor readings are transferred according to the routing tree to their "parent" sensors, which perform aggregation and in turn propagate the obtained results to their parent nodes until the root node is reached. In addition to routing trees, TinyDB uses additional techniques to reduce memory consumption, e.g., routing indexes (semantic routing trees) defining value ranges for sensor nodes to determine if a node (and the subtree underneath) provides sensor readings that are relevant to the query or not. In case there are no relevant readings, the subtree can be pruned.

In any case, the problem of finding an optimal routing tree is NP-hard [15] so that in general heuristics are used for its automatic construction. However, in some cases it might be worthwhile to allow a user to actively influence deployment, e.g., by specifying which sensor readings should be aggregated, e.g., to compute the average temperature in a room with multiple sensors. Most approaches simply do not consider this aspect.

Apart from AnduIN, one of the few approaches considering this aspect is GSN, which introduces virtual sensors. The goal of GSN is to support flexible integration and discovery of sensor networks and sensor data. Thus, GSN does not aim at providing in-network query optimization techniques but at overcoming heterogeneity originating from the variety of available software and hardware platforms. So, GSN operates on a different level: it does not make any assumptions on the internals of a sensor network. GSN allows to specify virtual sensors. A virtual sensor can be defined on an arbitrary number of input streams (sensor readings or existing virtual sensors) and produces one output stream. It is registered in GSN with a description containing all information necessary for deploying and using it. Virtual sensors are managed by GSN containers that communicate in a peer-to-peer style and that are considered as cooperating peers in a decentralized system.

Based on GSN, COSMOS [21] aims at supporting large scale stream processing using a publish/subscribe system to achieve loosely-coupled communication. The system provides a stream query processing service for a large number of users and tries to exploit the fact that different queries can share communication that is necessary to compute other one of them. COSMOS may work with different stream processors such as TelegraphCQ [6], STREAM [2], and Aurora [5].

Many approaches for query processing in sensor networks [4, 13, 18, 20] focus only on in-network processing of aggregate operations. They do not consider complex data mining tasks such as clustering, frequent pattern mining, and burst detection. These queries have mostly been neglected. To overcome this problem, systems like HiFi [7] (the Berkeley High Fan-In system) combine an in-network stream processor with a data stream management system [2, 6]. The HiFi system is a combination of TinyDB (as an in-network stream processor) and TelegraphCQ [6] (data stream management system). TinyDB collects and aggregates sensor data, sends complete data to TelegraphCQ, which then processes complex operations on the resulting data streams. HiFi uses an architecture consisting of multiple layers of data stream management systems running on computers. Sensor networks and stream processors are considered mostly independent so that global optimization, as considered by AnduIN, is not supported.

3. ARCHITECTURE

As stated in Section 1, AnduIN consists of a DSMS (AnduIN-core), a sensor node component of operators (in-network processor), and a Web-GUI-based frontend. In the following, we give a brief overview of these components.

The DSMS provides a query processor for the CQL language [3] supporting the standard query features such as sliding windows, filtering, joins, grouping, and aggregation. A special feature is the incorporation of synopsis operators, which are exploited for complex data stream analytics. A synopsis operator can be used at any position in a query where a window specification is expected:

```
SELECT * FROM stream [ synopsis(params) ]
```

This operator computes an "aggregate" on the incoming stream. Examples are frequent patterns, cluster, or bursts as in the following query [14]:

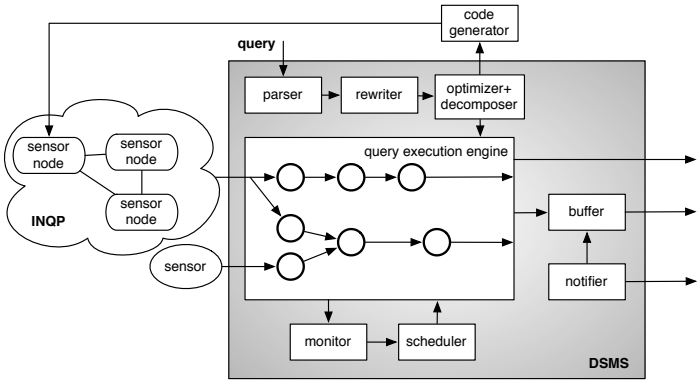

Figure 2: Architecture of AnduIN

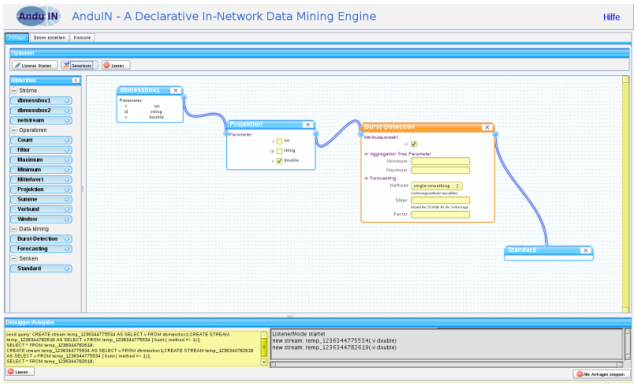

Figure 3: AnduIN GUI

```
SELECT value, timestamp
FROM s1 [ burst-detection(wlen => 100,
          threshold =>'Holt Winters') ]
```

CQL queries are processed in the usual way: the incoming query is parsed and translated into an internal representation. After the logical rewriting step, a query execution plan is derived – at the moment by a rule-based optimizer. Finally, the query plan is decomposed by taking the sensor network description into account (see Section 4).

The DSMS component provides two interfaces: one for receiving data from data providers (e.g., a sensor network) in the form of network ports (UDP and TCP) and a second one for the query issuer. Because the latter is important for application development, different ways of interaction are supported:

- a simple *pull* interface where the client reader is blocked until the next tuple can be retrieved,
- a *push* or listener interface where clients register their interest in certain queries and are notified by AnduIN when the tuples are available,
- a *buffer* interface where the result tuples are temporarily stored in a circular buffer of AnduIN and can be retrieved by the clients at any time.

Note that in all three cases the client sends a CQL query to AnduIN but can retrieve the results using one of these interfaces.

The overall architecture of the DSMS component is shown in Figure 2. In addition to the modules already described, the AnduIN query processor also contains a monitor and a scheduler to adjust the resources for the queries at runtime.

Besides the textual query interface, AnduIN provides a Web-GUI inspired by Yahoo! Pipes[1]. This GUI follows the popular box-and-arrow paradigm, which enables the user to create data flow processes by connecting operators through click, drag, and drop functionalities (Figure 3). When the user finishes editing a query, the graphical query representation is transformed into one or several CQL queries, which are then transmitted to the AnduIN engine.

The second part of AnduIN is the sensor node component, which consists basically of a library of C functions for the Contiki OS[2] of our sensor nodes. These functions implement several query operators running locally on the sensor nodes.

[1] http://pipes.yahoo.com/
[2] http://www.sics.se/contiki/

4. QUERY SPECIFICATION AND DECOMPOSITION

As described above, AnduIN contains an CQL processor. In contrast to systems like TinyDB [19], where the network is described in the query declaration, AnduIN separates the data modeling process from the data manipulation process (similar to traditional database systems). In the data modeling process, users are able to design data sources (e.g., sensor networks, static relations, or other stream sources) and its structures (e.g., fields, sampling intervals, topologies). In order to support different data sources, we introduced appropriate CQL extensions.

In the data manipulation phase, the user can design queries using the continuous query language CQL independent from the origin of the data source.

4.1 Register Sensor Nodes

New queries are propagated as sensor node images all over the network, that is all reachable nodes within the sensor network receive updates.

In general, all nodes within a sensor network receive the same image. Supposed we deploy complex data mining tasks like the anomaly region detection described in [10], images must differ in their functionality and nodes must receive additional data. For instance, when the sensor does not have a GPS device but its localization is known. Nevertheless, the user should be able to describe the nodes' localization in order to instance spatial queries, which can be preprocessed within the network.

For this reason, AnduIN supports the registration and description of sensor nodes. Let us show an example:

```
ADD SENSOR 15 (temp double, hum double)
        LOCALIZATION [47° 25', 010° 59']
```

This line will add a sensor node identified by id 15, providing temperature and humidity. Additionally, the sensor is assigned some localization information. In order to support a bulk registration, we plan to import sensor network description using the XML based Sensor Model Language (SensorML[3]).

4.2 Logical Sensor Networks

Usually, sensor networks are self-organizing ad-hoc networks, that is data is forwarded by sensor nodes using multi-hop routing trees, which are built dynamically, depending on the sensors' behavior. In addition to this physical layer, the literature proposes different approaches to build logical network layers (e.g., aggregation trees [19]). In general, the logical layer is designed automatically.

[3] see http://vast.uah.edu/SensorML/

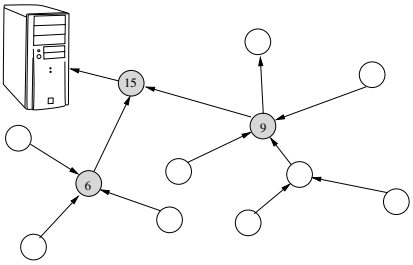

Figure 4: Example Topology

Since the problem of finding the optimal tree is NP-Hard [15], only suboptimal solutions are possible. Building a logical layer, approaches like the aggregation tree use semantic knowledge about the query. Physical design patterns are not considered within the logical network.

In AnduIN, we additionally support manual development of a logical network layer. So, the user is able to design network topologies, which AnduIN could use to process complex operations within the sensor network. Let us consider an example: a modern building is equipped with a sensor network to measure values like temperature, humidity, and occupancy. In general, the sensors will build an ad-hoc network, independent from the underlying building architecture, that is all measurements are sent using multi-hop messages to the central stream engine, which processes the data. In order to preprocess data in-network, for instance to monitor a room or a floor, the network nodes have to know this semantics.

In AnduIN, we can describe hierarchies using simple CQL commands, for example:

```
CREATE STREAM net_stream (id int, temp double)
       NETWORK [ 15 (6, 9 )]
       SAMPLING 30 SECONDS
```

Using this statement, a data stream called *net_stream* is registered. The data stream provides data from a sensor network source, consisting of two attributes (id and temperature). All concerned sensor nodes, i.e., all nodes within the sensor network (including not registered nodes) delivering an id and a temperature value, are measured each 30 seconds. Developing a logical network topology is only possible with already registered nodes. The simple example above describes the logical overlay shown in Figure 4. It is sufficient to register the three gray nodes. In case a complex operation is based on leader node processing, the light gray nodes will automatically be chosen by AnduIN.

The create stream statement only registers a new logical network topology as a source so that a new *virtual network source* is created. The registration has no effect to running images within the sensor network. A new image is created not before the user defines a query requesting the network source. Using virtual sources, we are able to design complex queries based on different logical sensor networks (e.g., differing in their topology and sampling rate). In case a new query (using at least one virtual network as source) is added to the running query list of AnduIN, a new sensor node image containing the in-network part of all current running queries is generated and propagated within the sensor network.

4.3 Query Decomposition

As described in Section 3, the logical query plan is decomposed into a local plan (executed by the stream engine) and an in-network plan, which results in new sensor node images that are automatically distributed applying over-the-air-programming. In a first step, the initial logical plan is reorganized by a set of rules – rule-based optimization is a popular heuristic to reduce the search space for further optimization algorithms. In the next step, the physical optimizer of AnduIN's core engine transforms the resulting logical plan into possible physical plans, the transformation starts from the sources nodes.

parameter	description
m	number of sensor nodes within the network
m_l	number of leader nodes
r_s	the rate of taking measurements in $\frac{1}{s}$
c_{wake}	costs to wake up the node
c_{sample}	measurement costs
c_{msg}, $\hat{c}_{msg}$	message costs
h	average number of hops from a source node to the central instance (we assume that the sensor nodes are organized in a balanced tree)
c_{cpu}^{loc}	costs for local (on the source node itself) computations
c_{cpu}^{neigh}	costs for processing data from a neighborhood
σ_j	the selectivity of operator j, $\sigma_0 = 1$

Table 1: Parameters influencing the cost model

Similar to traditional databases, in AnduIN a logical operator can be mapped to a number of different physical operators, both in the local DSMS and in the INQP. Currently, we enumerate all possible physical plans derived from one logical plan. Subsequently, we choose the plan minimizing the overall sensor network power consumption C (in $\frac{\mu J}{s} = W$ (Watt)). In order to evaluate the obtained physical plans, we developed two cost models. The first one estimates the costs for primitive in-network processing, i.e., the costs for source nodes measuring their values and performing some operations. Afterwards, results are sent to the central instance, which completes data processing:

$$C_{simple} = m \cdot r_s \cdot \Bigg[c_{wake} + c_{sample} + \sum_{j=1}^{p} c_{cpu}(j) \cdot \sigma_{(j-1)} + \prod_{j=1}^{p} \sigma_j \cdot \Big(h + (h-1) \cdot c_{wake} \Big) \cdot c_{msg} \Bigg]$$

Our second cost model also considers complex mining tasks, where at least one node per cluster, respectively per neighborhood, computes some operations exclusively for the cluster it belongs to. Nevertheless, source nodes also preprocess data:

$$C_{innet} = r_s \cdot \Bigg[m \cdot (c_{wake} + c_{sample} + \sum_{j=1}^{p} c_{cpu}^{loc}(j) \cdot \sigma_{(j-1)}^{loc}) + \prod_{j=1}^{p} \sigma_j^{loc} \cdot (m - m_l) \cdot (2 + c_{wake}) \cdot c_{msg} + m_l \cdot \Bigg(\sum_{k=1}^{q} c_{cpu}^{neigh}(k) \cdot \sigma_{(k-1)}^{neigh} + \prod_{k=1}^{q} \sigma_k^{neigh} \cdot \Big((h-1) + (h-2) \cdot c_{wake} \Big) \cdot \hat{c}_{msg} \Bigg) \Bigg]$$

All parameters influencing the cost model are shown in Table 1. A detailed derivation of our cost model is described in [10].

Having the optimal plan, we insert a connection manager between the local part and the in-network part. The connection man-

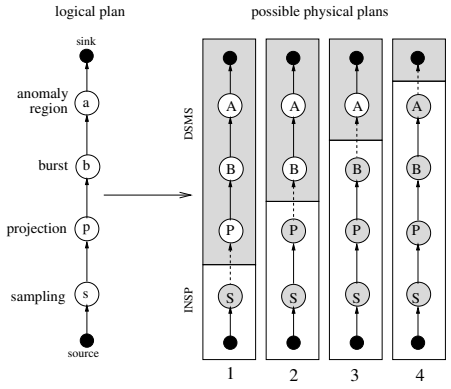

Figure 5: Example Query Plan Optimization

ager receives results from the sensor network and forwards them to the DSMS part. The local plan is then executed at the DSMS using the connection manager as stream source.

As described above, a new in-network query results in a new image for all sensor nodes within the network. Based on the result of the decomposition, the new code for the sensor nodes is generated (on code level), compiled, and finally distributed. The operator library is developed in ANSI C (based on the Contiki operation system), which simplifies extensibility. A new image contains at least of a sampling and a sending operator. Between this default operators, all outsourced operators will be embedded, that is new tuples measured by the sampling operator flow through all embedded operators and are sent to the sink by the sending operator.

5. CASE STUDY

In this section, we show an example for query processing and the resulting network deployment. Supposed we have a sensor network consisting of 100 nodes, the aim is to detect anomaly regions within the sensor network with sensor nodes detecting bursts as anomalies. For more details on anomaly region detection, please see [10].

In addition to the sensor attributes, the anomaly region detection algorithm needs localization information of all sensor nodes. In case of the absence of an automatic localization technique like GPS, the user must describe the spatial attributes for each sensor node. Furthermore, he has to describe the source stream. For simplification, we assume a data stream specified by the **CREATE STREAM**-statement from Section 4.

In the next step, the query is described. Note that in CQL nested queries are not possible. For this reason, we have to design complex, nested queries using views. Our aim is to detect regions showing abnormal behavior, abnormal behavior in this context means bursty data. Consequently, in a first step we have to describe a view, which defines the synopses *burst-detection*.

```
CREATE STREAM  s_burst AS
SELECT timestamp, temp
FROM net_stream [ burst-detection (w => 1000,
          threshold =>'Holt') ]
```

At this, we use an adaptive burst detection algorithm, which observes a window of length 1000. The adaptation of the burst thre"-shold is done by double exponential smoothing (also known as Holt-forecasting). The creation of this view does not have any effect to currently running queries within the based network.

Afterwards, we define the *anomaly-region* synopses as

```
SELECT timestamp, temp
FROM S_BURST [ anomaly-region (t => 0.5)]
```

where t denotes the threshold of interest, i.e., regions exceeding t will be shown.

measurement	time	energy
MCU wake up	$115\mu s$	$0.416\mu J$
sample humidity	$71ms$	$75.92\mu J$
sample temperature	$221ms$	$270.92\mu J$
sample light	$18ms$	$23,4\mu J$
sample CO2	$0.5s$	$237,5\mu J$

Table 2: Power consumption on real sensors

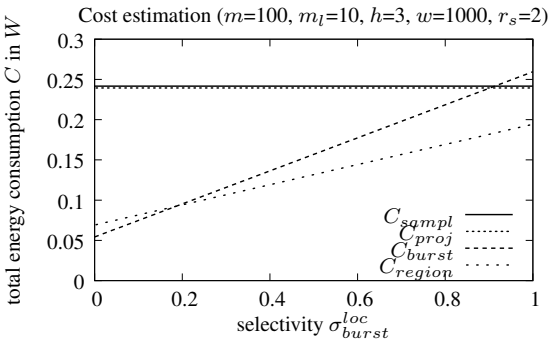

Figure 6: Power consumption on different selectivities

When AnduIN has translated the query into a logical query plan and successfully rewritten, it enumerates all possible execution plans. Figure 5 shows the logical plan, based on the query described above and a list of all (at current state) possible physical plans.

In the next step, all plans are evaluated using the cost model discussed in Section 4. At this, wake up and data sampling costs are constant sensor model dependent costs. It is sufficient to measure these values only once. Table 2 shows the power consumption for real sensors (Tmote Sky sensor nodes with 16 bit MCU MSP430F1611, 4 MHz clock rate, IEEE 802.15.4 compatible CC2420 transceiver with 250kBit/s)).

Parameters like the message costs c_{msg} and the processing costs c_{cpu} also depend on the sensor type. They also depend on the used operator and its configuration (e.g., window size). Table 3 summarizes the power consumption of some basic operations, e.g., computing the average or sending data.

Parameters like the network size m or the average hop number h depend on the network. In general, sensor networks are built in an ad-hoc fashion, that is we have to find a way to estimate the average number of hops. A simple approach is to use a broadcast message and sending it once to all nodes within the sensor network. Finally, the nodes reply indicating the number of hops needed to contact them. The sampling rate r_s and, if necessary, the number of leader nodes m_l is known from the network description.

An essential parameter influencing the overall power consumption is an operator's selectivity σ_j. The selectivity depends on the operator type and the data distribution. An approach to approximate the data distribution (and consequently the operator selectivity) is the usage of statistics obtained from previous queries. In case we do not have any statistics, we have to estimate the costs using default values. Figure 6 shows the estimated costs for our example query using our cost model. This example shows that in general in-network processing is not always the best solution. For instance, if the number of bursts increases, i.e., if the selectivity of the burst op-

measurement	time	energy
compute average of 10 values	$52.3\mu s$	$0.272\mu J$
compute average of 100 values	$245\mu s$	$1.274\mu J$
single addition	$2\mu s$	$0.010\mu J$
single division	$27\mu s$	$0.140\mu J$
single multiplication	$16.2\mu s$	$0.08\mu J$
sending 1 byte	$4.85ms(2.33 - 6.95ms)$	$240.19\mu J(121 - 361\mu J)$
sending 10 bytes	$4.9ms(2.8 - 7.4ms)$	$252.93\mu J(146 - 385\mu J)$

Table 3: Average energy consumption measured on real sensors

erator is near 1, a central detection of bursts minimizes power consumption. Figure 6 also shows that a complete in-network region detection is only recommendable if the burst operator's selectivity exceeds 0.2.

Assume previous observations indicate a burst probability below 0.2, then AnduIN chooses the plan decomposition 3 of Figure 5. That is, both the projection and the burst detection will be processed in-network and the region detection will be processed within the stream engine. In case the observations indiate a burst probability higher than 0.2, all operations will be processed within the sensor network.

6. CONCLUSION

In this paper, we have presented AnduIN, a system developed to alleviate operating wireless sensor networks. AnduIN combines characteristics of data stream management systems (DSMS) and in-network query processors (INQP). In order to optimize interaction between these two components, AnduIN offers an optimizer to identify which parts of a query should be processed by the sensor nodes and which parts by the query processor. The paper focuses on application-oriented practical aspects that a user working with such a system has to deal with. To improve usability, we have proposed a flexible declarative approach for query processing and deployment. In our case study, we illustrate the general steps of query processing and decomposition using AnduIN. We also show that in general in-network processing is not the most power efficient solution.

Processing data partially in-network also offers a lot of interesting challenges. As described in Section 5, the optimal query plan depends on the data distribution. In case the data distribution changes, we have to reorganize the running plan. Currently, the decomposition of AnduIN only considers the overall network power consumption. In future work we also plan to take optimization criteria such as data throughput and quality into account.

7. REFERENCES

[1] ABERER, K., HAUSWIRTH, M., AND SALEHI, A. Infrastructure for data processing in large-scale interconnected sensor networks. In *MDM* (2007), pp. 198–205.

[2] ARASU, A., BABCOCK, B., BABU, S., DATAR, M., ITO, K., MOTWANI, R., NISHIZAWA, I., SRIVASTAVA, U., THOMAS, D., VARMA, R., AND WIDOM, J. STREAM: The Stanford Stream Data Manager. 19–26.

[3] ARASU, A., BABU, S., AND WIDOM, J. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Tech. rep., University of Stanford, 2003.

[4] BONNET, P., GEHRKE, J., AND SESHADRI, P. Towards sensor database systems. In *Mobile Data Management* (2001), pp. 3–14.

[5] CARNEY, D., ÇETINTEMEL, U., CHERNIACK, M., CONVEY, C., LEE, S., SEIDMAN, G., STONEBRAKER, M., TATBUL, N., AND ZDONIK, S. Monitoring streams: a new class of data management applications. In *VLDB '02* (2002), VLDB Endowment, pp. 215–226.

[6] CHANDRASEKARAN, S., COOPER, O., DESHPANDE, A., FRANKLIN, M., HELLERSTEIN, J., HONG, W., KRISHNAMURTHY, S., MADDEN, S., RAMAN, V., REISS, F., AND SHAH, M. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR '03* (2003).

[7] COOPER, O., EDAKKUNNI, A., FRANKLIN, M., HONG, W., JEFFERY, S., KRISHNAMURTHY, S., REISS, F., RIZVI, S., AND WU, E. HiFi: A unified architecture for high fan-in systems. In *In VLDB* (2004), Demo, pp. 1357–1360.

[8] CULLER, D., ESTRIN, D., AND SRIVASTAVA, M. Overview of sensor networks. *IEEE Computer 37*, 8 (2004), 41–49.

[9] FASOLO, E., ROSSI, M., WIDMER, J., AND ZORZI, M. In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE 14* (2007), 70–87.

[10] FRANKE, C., KARNSTEDT, M., KLAN, D., GERTZ, M., SATTLER, K.-U., AND KATTANEK, W. In-Network Detection of Anomaly Regions in Sensor Networks with Obstacles. In *BTW 2009* (2009).

[11] HANDY, M., HAASE, M., AND TIMMERMANN, D. Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In *IEEE Conference on Mobile and Wireless Communications Networks (MWCN)* (2002), pp. 368–372.

[12] HEINZELMAN, W., CHANDRAKASAN, A., AND BALAKRISHNAN, H. Energy-efficient communication protocol for wireless microsensor networks. *Hawaii International Conference on System Sciences 8* (2000), 8020.

[13] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom '00* (New York, NY, USA, 2000), ACM, pp. 56–67.

[14] KARNSTEDT, M., KLAN, D., PÖLITZ, C., SATTLER, K.-U., AND FRANKE, C. Adaptive burst detection in a stream engine. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing* (New York, NY, USA, 2009), ACM, pp. 1511–1515.

[15] KRISHNAMACHARI, B., ESTRIN, D., AND WICKER, S. B. The impact of data aggregation in wireless sensor networks. In *ICDCSW '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 575–578.

[16] KULIK, J., HEINZELMAN, W. R., AND BALAKRISHNAN, H. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks 8* (1999), 169–185.

[17] LINDSEY, S., AND RAGHAVENDRA, C. Pegasis: Power-efficient gathering in sensor information systems. *Aerospace Conference Proceedings, 2002. IEEE 3* (2002), 1125–1130.

[18] MADDEN, S., FRANKLIN, M., HELLERSTEIN, J., AND HONG, W. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 131–146.

[19] MADDEN, S. R., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. TinyDB: an acquisitional query processing system for sensor networks. *ACM TDS 30*, 1 (2005), 122–173.

[20] YAO, Y., AND GEHRKE, J. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record 31*, 3 (2002), 9–18.

[21] ZHOU, Y., ABERER, K., SALEHI, A., AND TAN, K.-L. Rethinking the Design of Distributed Stream Processing Systems. In *The 4th International Workshop on Networking Meets Databases* (2008), IEEE Computer Society, pp. 182–187.