MEADOWS: Modeling, Emulation, and Analysis of Data of Wireless Sensor Networks

Qiong Luo, Lionel M. Ni, Bingsheng He, Hejun Wu, and Wenwei Xue

Department of Computer Science

The Hong Kong University of Science and Technology

Clear Water Bay, Kowloon

Hong Kong, China

{luo, ni, saven, whjnn, wwxue}@cs.ust.hk

Abstract

In this position paper, we present MEADOWS, a software framework that we are building at HKUST (The Hong Kong University of Science and Technology) for modeling, emulation, and analysis of data of wireless sensor networks. This project is motivated by the unique need for intertwining modeling, emulation, and data analysis in studying sensor databases. We describe our design of basic data analysis tools along with an initial case study on HKUST campus. We also report our progress on modeling power consumption for sensor databases and on wireless sensor network emulation for query processing. Additionally, we outline our future directions on MEADOWS for discussion and feedback at the workshop.

1. Introduction

Sensor networks have created exciting opportunities for data management [2], especially for in-network query processing [1][5][11][18], because these networked sensor nodes form a large-scale, dynamic, and distributed database with each node acquiring, processing and transmitting data simultaneously. However, studying in-network sensor query processing is a challenging task due to the unique features of sensor networks. These unique features include: (1) each sensor node has limited computation, communication, and storage capabilities as well as limited power supply; (2) sensory units and communication channels are lossy and error-prone; and (3) deployed sensor nodes are embedded in the physical world, scattered geographically, and often mobile. To facilitate the study of sensor databases in general and in-network query processing in specific, we propose MEADOWS, a software framework that we are building at HKUST (The Hong Kong University of Science and Technology) for modeling, emulation, and analysis of data of wireless sensor networks.

Copyright 2004, held by the author(s)

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, 2004

Modeling, emulation, and data analysis for sensor networks is essential for studying in-network query processing systematically. On one hand, studying query processing techniques in real sensor networks with real applications has been fruitful and has a high practical impact [11]. On the other hand, the tight integration of sensor networks with the physical world, the high uncertainty in sensory data, and the high deployment cost make it hard to produce general and complete results through field studies only. Consequently, it is highly desirable to perform in-depth analysis of sensory data from field studies and to model and emulate sensor networks in controlled environments.

Let us give a real-world example to illustrate the usefulness of MEADOWS. This example is an experimental monitoring application that we deployed near a frog pond on HKUST campus in the spring of 2004. We used the MICA2 Motes made by Crossbow [4] for the sensor nodes and TinyDB [15] as well as other software running on the motes to collect sensory data. In TinyDB, the data collection process is the execution of declarative, SQL-like queries, which eases application development and allows for performance optimization. However, answering some important questions about the query processor for the application is difficult or infeasible through a simple field study. Specifically, some of these questions are as follows:

(1) We have only ten sensor nodes available for the application. How many do we really need and what geographical deployment topology do we use to observe important phenomena such as trends in temperature, humidity, and frog croaks around the frog pond?

(2) If we collect sensor readings every 30 seconds, what is the status of power consumption at each node over time and when will the batteries run out?

(3) If we change the type of sensor nodes (e.g., CPU, radio channel, sensing units), the sensor network routing scheme, or the data collection queries, what are the resulting answers to questions (1) and (2)?

In MEADOWS, we attempt to answer these questions through data analysis, modeling, and emulation. We show that we can determine the number of sensor nodes needed and the geographical deployment scheme by performing data analysis (Section 2). We also show that we can realistically estimate power consumption in various scenarios by including real-world factors into modeling and emulation (Sections 3 and 4). In addition, the integration of data analysis, modeling, and emulation helps answer the questions better than merely employing one of these three approaches in isolation. Our ultimate goal is to enable various studies on sensor databases and sensor query processing.

To date, modeling, emulation, and data analysis of sensor networks for query processing is still at an early stage. Our work in MEADOWS is only an initial step in this direction. In this early report, we present a case study of preliminary sensor network data analysis in Section 2, a hierarchical power consumption model for sensor databases in Section 3, and a sensor network emulator for query processing in Section 4. We draw conclusions and list future directions in Section 5.

2. Analysis of Sensor Network Data

In this section, we focus on real-world sensory data and discuss a case study of collecting and analyzing the data from a small network of sensors deployed outdoors on the HKUST campus. The purpose of this case study is to explore how data analysis can help answer questions about sensor query processors. In addition, we aim to gain insights for data analysis tool design.

2.1 Overview

Analysis of real-world data provides a realistic basis for modeling and emulation. Because sensor networks are designed to be tightly embedded in the physical world, collecting and analyzing real-world sensor network data is both challenging and worthwhile. Even though there have been a few projects on outdoor deployment of sensor networks [14], we have not yet seen studies that answer questions about query processors. Therefore, as a first step of our framework development, we conducted a field study with this specific goal in mind. The scale of the study was small due to our resource limitation. However, the study is sufficient for the purpose of producing an initial design of data analysis tools.

The case study is the frog pond monitoring application we briefly described in the Introduction. The frog pond is located at the northeastern corner of the campus. Throughout the late spring, the frogs in the pond croak loudly all day long. We chose the frog pond as it has this interesting phenomenon as well as other outdoor microclimate characteristics (e.g., close to the sea and two pagodas).

We deployed a small number of sensor nodes in two groups near the frog pond. We collected one-day of sensory data during four two-hour periods. We pre-processed the data by adding labels (e.g., timestamps) and converting data formats (e.g., from raw sensor readings to more human-friendly engineering units). We analyzed the data by examining patterns, exceptions (outliers), and correlations. Finally, we discuss our design of data analysis tools as well as the insights gained from the case study.

2.2 The Case Study

We deployed two groups of MICA2 motes in the two pagodas near the frog pond (Figures 1 and 2). Mote 0's of both groups were sink nodes connected with a laptop through a serial cable. Group 1's Motes 1-5 used the MTS310CA sensor boards, which detect temperature, light, noise level, acceleration and magnetic value. Group2's Motes 1-2 used the MTS420CA weather sensor boards, which measure temperature, light, acceleration, humidity and barometric pressure. We used TinyDB [15] to collect data from Group 1 and a modified Xlisten program from the TinyOS Sourceforge CVS directory [17] to collect data from Group 2, due to the applicability of the software to different types of sensor boards. In addition, we logged battery voltage of both groups for data conversion and analysis.

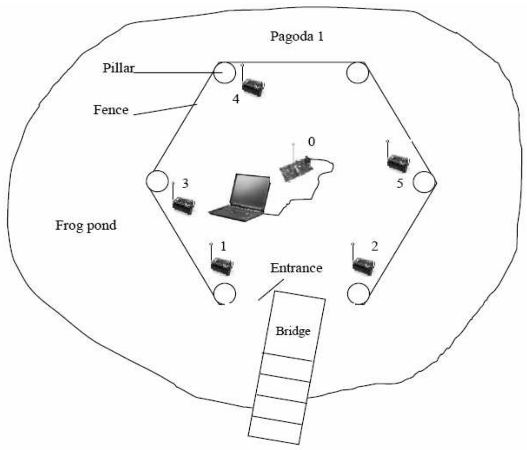

Figure 1: Deployment of Group 1 Motes

It was a cloudy day and rained intermittently. We collected data during the following four 2-hour periods: 6:30-8:30, 12:30-14:30, 17:30-19:30, and 22:00-24:00. We set the sampling period of each reading to be 30 seconds and collected thousands of readings per group. We show three figures (Figures 3-5) as representative examples.

The noise readings of all sensor nodes in Group 1 were similar at any point in time. We picked two motes that differed most in the readings, Motes 1 and 5, shown in Figure 3. These readings mainly captured frog croaks. They indicate that frogs croaked most actively in the early morning and least actively at noon time. There is a gap of a few minutes in the morning readings, which was due to a crash of our data logging program and its subsequent recovery.

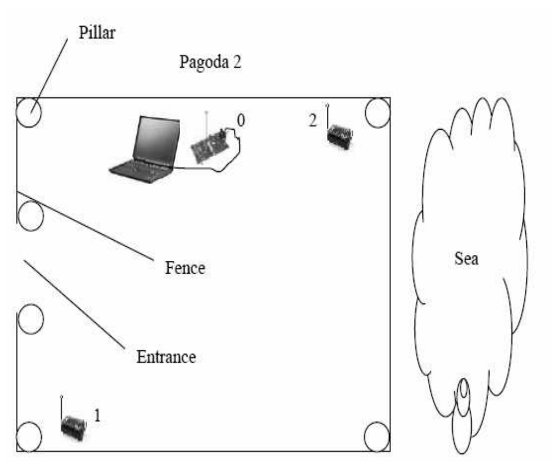

Figure 2: Deployment of Group 2 Motes

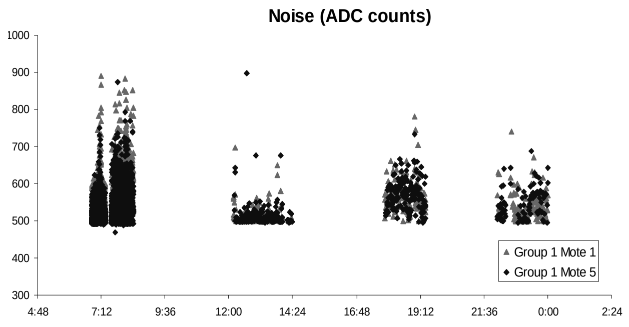

Figure 3: Group 1 Noise Readings

The humidity readings of Group 2 remained at the level of around 90% most of the time (Figure 4). Readings of abnormally high humidity (larger than 130%) at Mote 1 were detected in the early morning, because rain drops accidentally splashed onto Mote 1 when we took it out of a box and deployed it. The water made the humidity sensor at Mote 1 malfunction and thus return abnormally high readings. This kind of physical problem for motes is common and recoverable [14]. After being dried, the humidity sensor returned to normal operation.

The temperature readings of the two groups varied slightly within each group (21-24°C in Group 1 and 21-23°C in Group 2). As illustrated in Figure 5, the temperature measured by Group 2 motes was often slightly higher than that measured by Group 1 motes (except around noontime), even though the two pagodas were close to each other (within 20 meters). We think there are two possible reasons for this difference: (1) the temperature sensors of the two groups have different hardware characteristics since they are made by different companies, and (2) the microclimates in the two pagodas had a slight difference due to their different geographical locations.

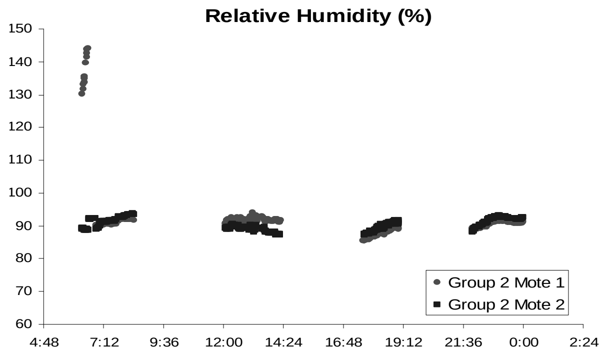

Figure 4: Group 2 Humidity Readings

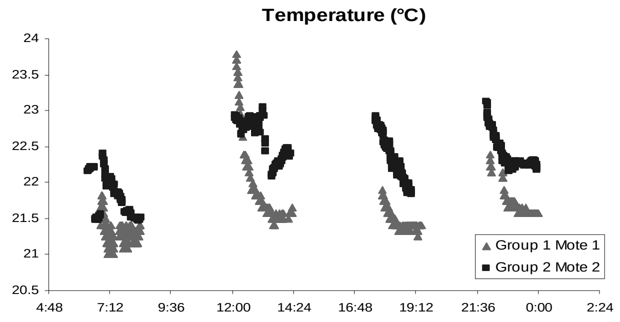

Figure 5: Temperature Readings of Two Groups

2.3 Discussion

From our data analysis, we suggest that the application only use one Mote per pagoda for a small-scale case study around the frog pond, since the readings within each group were similar and there were slight differences between the two groups that were deployed in different geographical locations (pagodas). Moreover, if the application scenario changes and more questions about the query processor are asked, we need to have a set of general data analysis tools to answer these questions.

Based on our experience with the frog pond case study, we propose the following three requirements for a sensory data analyzer.

(1) The analyzer should have data acquisition functions that are fault-tolerant and adaptive, since the sensory data collection process determines the quality of sensory data. The fault-tolerance requirement is because hardware malfunctioning is common in field studies, as we experienced. It is thus desirable that a data collector is able to recover, to migrate the work from a failed node to a normal node, and to resume the work. The adaptivity requirement is to take advantage of the patterns and regularities captured in sensor readings. For instance,

continuous quantities such as temperature can be measured with a sampling frequency adapted to the changes in the temperature readings in order to improve power efficiency while keeping the quality of sensory data unaffected.

(2) The analyzer should have a set of basic functions for data pre-processing and post-processing operations. Data pre-processing is to further ensure the quality of the data for analysis. Data post-processing is mainly for the presentation of analytical results. For example, the function convert() converts sensor readings from raw ADC counts to human-friendly engineering units, the function calibrate() performs hardware-specific calibration of the readings, and the function plot() plots data points and curves following user-defined criteria for analytical summaries.

(3) As the core of the analyzer, the sensory data analysis functions include pattern and outlier detection, and correlation of multiple sensory attributes or multiple sensor nodes. We further discuss these two types of functions as follows:

First, detecting patterns and outliers in single-node single-attribute sensory data is the basic analytical operation. For instance, given the temperature readings of one sensor node, the basic analytical information about these readings must include a summary of the range, the trend, and the outliers of the data. As a result of measuring natural phenomena, sensory data has inherent patterns as well as outliers. Moreover, outliers are sometimes due to real events in the environments and sometimes due to system errors. It is necessary to pay special attention to outlier analysis.

Second, correlation analysis gives insight into sensory data, because each sensor node has multiple sensory attributes and multiple sensor nodes work concurrently in a geographical region. The inherent correlations between natural phenomena as well as the temporal and spatial correlations of sensor nodes are useful for both sensor query processing and application deployment. For example, when an application is detecting transient changes such as a sudden increase in the noise level, it can utilize the spatial correlation of a cluster of adjacent nodes to detect the noise with a high fidelity. In other words, an increase in the detected noise level could be the result of a real event and/or a system error. But if multiple nearby nodes report the same event, the probability of the change occurring as a result of a system error is much lower than that of it occurring as a result of a real event.

In summary, analytical results from real-world sensory data, such as patterns, outliers, and correlations, can help answer questions about query processors as well as improve query processing. In addition, data analysis can interact with modeling and emulation to better serve the purpose of studying query processing. On one hand, analytical results serve as a realistic basis for modeling and emulation; on the other hand, modeling and emulation can be used for guiding and cross-validating data analysis.

3 Modeling Power Consumption

Having presented a case study of sensory data analysis, next we turn to modeling sensor databases. Due to the short time period (eight hours) and resource constraints (no oscilloscope on site) of the field study, we were unable to obtain detailed power consumption statistics. Since power efficiency is a major issue in sensor query processing, we examine this issue through modeling and emulation.

3.1 Overview

Power efficiency is a major issue in sensor networks, since sensor nodes are battery-powered and it is difficult or infeasible to recharge deployed sensor nodes in practice. There has been work on the power efficiency of sensor nodes [6][13], sensor networks [8][10], and senor query processing techniques [1][3][11][18]. However, it remains unclear how to evaluate systematically the power efficiency of sensor databases. The main reason is that there are many intertwined factors that affect power consumption in a sensor database system: sensor node computation, wireless transmission, and various query processing techniques. Therefore, we propose to represent these factors in a general model to study the power consumption of sensor databases.

We group these factors into a three-level hierarchy (Figure 6): the sensor database, the sensor network, and the sensor node. The sensor node model captures the power consumption characteristics of a single sensor node and provides a quantitative approach to estimate the power consumption of a single sensor node by the operations of the node. The sensor network model groups the main factors in wireless communication that affect power consumption. It adapts the quantitative approach provided by the sensor node model to a network environment. The sensor database model formalizes the main factors of database workloads that affect power consumption in a sensor network and further improves the accuracy of power consumption estimation for database workloads.

As a result, our hierarchical model can estimate the power consumption of a sensor query processing workload in a unified and general way. We can instantiate each level of model with specific real-world factors and realistically estimate power consumption of query workloads. For instance, we can use the MICA2 hardware specification for the sensor node, a typical network routing scheme for the sensor network, and a monitoring query used in our frog pond application for the database workload.

In the remainder of this modeling section, we use UML (Unified Modeling Language) style illustrations for modeling (Figures 6-9). A big box with a small square on top represents a *package*, e.g., “Sensor Database Model”. A package can contain other packages. A dashed line with an arrow stands for the "uses" relationship. A solid line with an arrow stands for the "has" relationship.

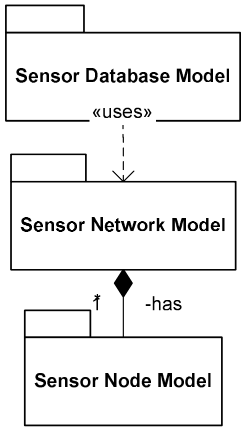

Figure 6: Model Hierarchy

3.2 The Model

We show our hierarchical power consumption model in Figures 7, 8, and 9 and describe them briefly. For brevity, all formulas are omitted and will be available in a technical report.

In Figure 7, we represent the configuration of a smart sensor node as a package of six types of units: the processor, the RAM, the flash memory, the wireless transmission unit, the battery, and the sensing data units. A *configuration* contains the important units (in terms of power consumption) of a sensor node and the parameters for the power consumption estimation of the units. The parameters starting with "pc" represent the unit power consumption, e.g., "pcInstruction" of the processor stands for power consumption per instruction. We define several operations in a sensor node (not shown in Figure 7): sensing (sampling), listening, sending (transmitting), receiving, discarding, and processing. We estimate the power consumption of a sensor node during a period of time by summing up the power consumption of all operations during that period. For each operation, the power consumption is calculated using a linear battery model [13]. Clearly, our sensor node model accommodates a wide range of sensor nodes with various hardware characteristics.

In Figure 8, we model a sensor network with the canonical topology, the routing scheme, and the model metrics. The canonical topology is represented as an undirected graph with its k-ary spanning tree. The routing scheme is responsible for building the spanning tree on the graph. For instance, in the flooding scheme, we can build the spanning tree by traversing the graph via Breadth-First Search. Finally, the model metrics include per-node metrics (the number of neighbors per node and the number of children per node in the spanning tree) as well as network-wide metrics (expansion, resilience, and distortion). Note that a node's neighborhood is determined by the wireless signal transmission range in the deployment whereas a node's children are determined by the routing tree. Since different routing schemes have different power consumption characteristics, our sensor network model aims to provide insights for designing power-efficient routing schemes.

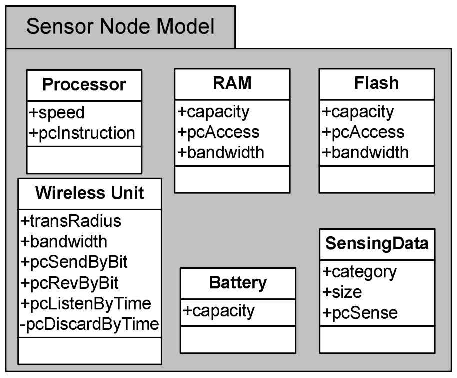

Figure 7: Sensor Node Package

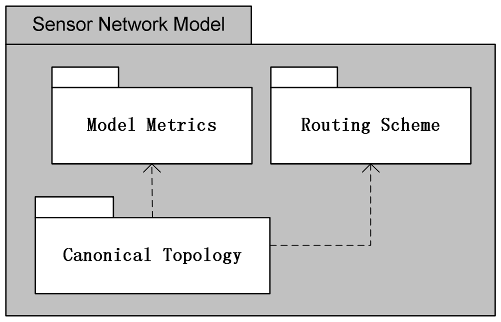

Figure 8: Sensor Network Package

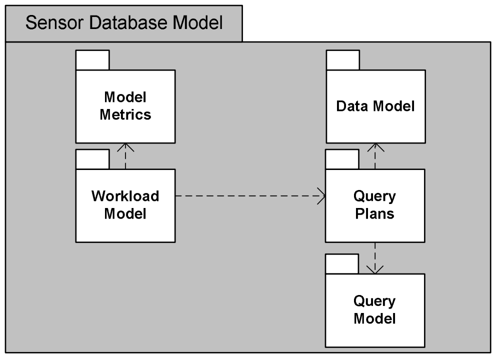

Figure 9: Sensor Database Package

In Figure 9, the sensor database model consists of the data model, the query model, the query plans, the workload model, and the model metrics. Our data model is relational and our query model is TinySQL-style extended SQL [11] with clauses specifying sampling rate

EPOCH and query lifetime *LIFETIME*. The query plans describe the execution plans of queries with selection, projection, and aggregation operators. The model metrics include the number of tuples, the size of each tuple, and the *reduction factor* of each operation (selection, projection, or aggregation). A reduction factor is defined as the ratio of the output data size to the input data size of the operator. Finally, the workload model estimates power consumption of the query workload in the sensor network.

To estimate the power consumption of a query workload, we consider both the local computation cost and the network traffic cost, which depend on the complexity of the handling and the volume of data handled. We developed algorithms for estimating sensor network lifetime in terms of power consumption in the static deployment and dynamic deployment, respectively. In the static deployment, the routing tree does not change as long as the network topology does not change. In contrast, the routing tree changes dynamically in the dynamic deployment. The algorithms estimate the power consumption for each node and identify the *weak points* in the sensor network. A weak point is a node whose power consumption is higher than others in the sensor network.

The algorithm for the static deployment works as follows:

(1) Generate a k-ary spanning tree based on the selected routing scheme. If it fails, the algorithm stops.

(2) Generate the query plan of the query workload on the sensor network and estimate the reduction factors for selection, projection and aggregation as needed.

(3) Estimate the power consumption of each node for this query workload as time goes, and identify the weakest point until it runs out of power.

(4) Remove the dead weak point from the network and repeat the previous steps starting from step (1).

For the dynamic deployment, we modify the algorithm for the static deployment by adding a time period *round*. At the end of each round, even though no nodes have run out of battery power, there is still a router reassignment process. Similar to the algorithm for the static deployment, the algorithm for the dynamic deployment estimates the lifetime of the deployment until the sensor network is disconnected.

3.3 Initial Validation Results

We validated our model using a typical sensor node configuration, two representative routing schemes, and a simple query workload. The sensor node configuration followed the MICA2 [4] Motes hardware specification. The two representative routing schemes we compared were LEACH [8] and flooding (Figure 10). LEACH identifies clusters of nodes and selects leader nodes of clusters in a round-robin fashion for packet merging (or called "partial aggregation" in networking terms, but not the "aggregation", e.g., *SUM()*, in database terms). The query workload we tested was a simple aggregation query: *"SELECT MAX(temperature), humidity FROM sensors GROUP BY humidity EPOCH 30 seconds"*.

The "sensors" virtual table had a schema of *{humidity, temperature, timestamp}* with a fixed length of 4 bytes per attribute. We assumed each packet contained a header of 20 bytes. With the temperature and humidity attributes in the query result, each packet contained 28 bytes. We also assumed that each sensor node covered a circular with a radius of 20 feet. The average distance between a sensor node and the sink node (Mote 0) was assumed to be 500 feet. We used LEACH's assumption that the unit power consumed in sending is proportional to the distance.

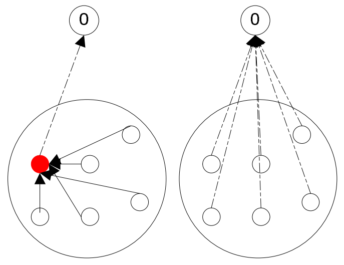

Figure 10: LEACH (left) versus Flooding (right)

Figure 11 shows the predicted average node lifetime in a network of *N* (ranging from 6 to 24) nodes resulting from our model. Our model predicts that LEACH results in a five-fold improvement in power efficiency over flooding whereas in the original LEACH paper this factor was eight. One major reason for this difference is that we considered the power consumption of database workloads as well as that of individual sensor nodes in addition to networking.

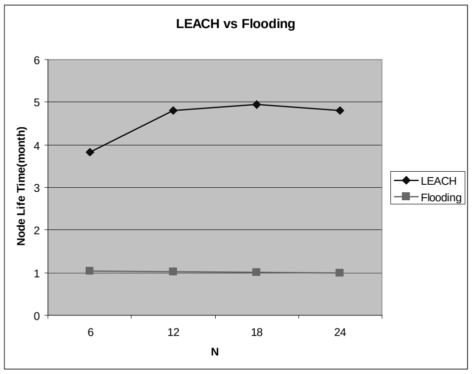

Figure 11: Predicted Average Node Lifetime

Since the number of nodes was small and there were at most two hops in LEACH in our study, the effect of database-style in-network aggregation (e.g., executing MAX() at a leader node) was insignificant. We are considering more complex and larger-scale cases for validation, in which in-network aggregation makes a difference [1][11][18].

3.4 Discussion

As shown in the preliminary results, our modeling can estimate power consumption of query processing workloads realistically using real-world factors such as sensor node hardware configuration, representative routing schemes, and typical queries in monitoring applications. To further improve our model, we consider the following three extensions:

(1) Extend the estimation of reduction factors for power-aware query processing. For example, our data analysis shows that patterns and correlations are common in sensory data. If a query processor takes advantage of these patterns and correlations and performs pattern-aware or correlation-aware data acquisition, we can extend the estimation method of reduction factors for these techniques.

(2) Extend the estimation of the node neighborhood in the sensor network model by considering the synchronization characteristics of transmission. A neighborhood of a node is a basic topology element in a multi-hop networking environment; transmission between nodes can be synchronous or asynchronous. We modeled transmission to be synchronous as is commonly assumed by existing work. To achieve a more accurate estimation, we plan to cover asynchronous transmission as well.

(3) Extend the database workload model to handle joins. Joins are a complex operation in sensor databases, which involves factors such as where and how to perform the join. The reduction factor alone seems to be insufficient for modeling the power consumption characteristics of a join operation.

4 Emulation for Query Processing

Modeling is useful for defining the problem space and quantifying the effects of multiple factors, as shown in our hierarchical power consumption model in Section 3. Nevertheless, dynamic behaviors of programs, for instance, parallel execution of query processing code on multiple sensor nodes, are often hard to abstract and to model. Under such situations, emulation is useful for observing the execution process. In this section, we present an emulator for sensor query processing.

4.1 Overview

Currently, it is difficult to study in-network query processing on real sensor networks, not only because the deployment is expensive and hard to maintain, but also because the resource constraints in a sensor network limit the collection of detailed statistics about the system's running status. Both simulation and emulation can ease these problems, either by representing the logical views and actions of the target system (simulation) or by executing the code with the same control flow as that of the target system (emulation).

We propose an emulation environment, VMN (Virtual Mote Network), for studying sensor query processing. It is a mix of simulation and emulation. We use TinyOS [16] modules to emulate the application execution environment in each VM (Virtual Mote). We simulate the radio channel and the sensing units of each VM following the MICA2 [4] hardware specification. The sensory data, which is fed into the virtual sensing units as the input of VMN, is generated from real-life data such as that collected in our frog pond monitoring application (Section 2). Finally, the execution of query processing code on each VM and the network topology are emulated on networked PCs.

Our VMN is different from the two existing sensor network simulators, TOSSIM [9] and EMStar [7], in that VMN utilizes networked PCs to emulate networked motes in parallel and has execution time and power consumption models for query processing applications. Other simulators such as ns-2 [12] and Sensorsim [13] or emulators such as EMPOWER [19] lack the execution environment of smart sensor nodes.

4.2 The Emulator

Our VMN (Figure 12) emulates a real network of MICA2 motes running TinyOS. PC 0 acts as the virtual base station, which runs VM 0 to emulate the sink node (Mote 0) in the real sensor network and runs the real application client (in this case, the TinyDB GUI) to communicate with VM 0. Each of the PCs 1 to n emulates multiple virtual motes except VM 0. Virtual motes communicate with each other through the virtual channel, which is implemented on top of the UDP (User Datagram Protocol) on a LAN (Local Area Network) and simulates a real radio channel with bit errors and delays.

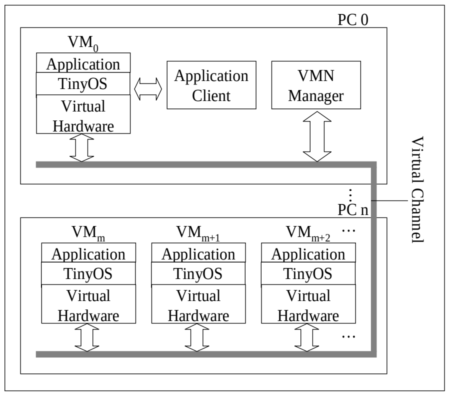

Figure 12: Architecture of a VMN

Each VM (Figure 13) emulates a MICA2 mote running TinyOS. We partition a VM into upper and the lower layers. The upper layer includes (i) the application, (ii) the senders and receivers of Active Messages (AM), UART (Universal Asynchronous Receiver/Transmitter, or RS232 serial communication) packets and radio packets, and (iii) the VM manager for emulation control and statistics collection on the node. The lower layer consists of (i) various types of virtual sensors, the virtual UART (for Mote 0 only), and the virtual RFM (Radio Frequency Monolithic), (ii) the virtual drivers for (a), and (iii) the virtual clock. This partitioning scheme identifies the components that are pertinent to program execution and then puts these components into the upper layer.

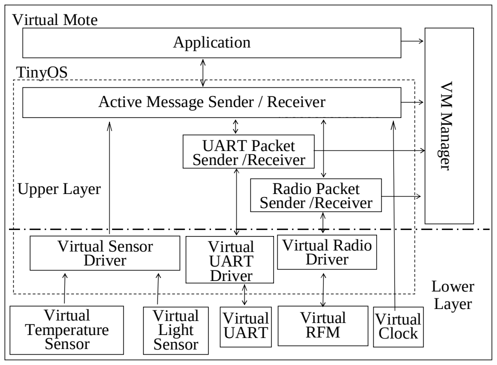

Figure 13: Architecture of a VM

Connecting multiple VMs, the virtual channel simulates wireless network effects using three software modules: the bit error module, the collision module and the delay module (shown in Figure 14).

The bit error module uses an experiential radio signal error data model to generate the bit error rate. The error rate is defined as (the number of error bits received by the receiver) / (the total number of bits sent by the sender). The module maintains a table of two attributes, distance and bit error rate, and generates bit errors randomly at a rate that the table specifies.

The collision module simulates radio signal collision by performing two operations: *carrier sense* and *collision*. Both operations need information about the *virtual time* (the time in the emulated world) and the data transmission status of all VMs. This information is kept in the VMN Manager.

In the carrier sense operation, the collision module asks the network manager whether if a sending VM can hear any VMs that are transmitting data. If so, the sending VM waits a period of time whose length is defined by the network protocols. In the collision operation, the collision module destroys the current bit to be sent on one of two conditions: (1) another VM is transmitting and the sender of this current bit can hear that transmitting VM, or (2) another VM is sending to the same destination as this sender.

Finally, the transmission delay module adds a delay to the virtual time of each packet to be sent.

Having described the three network effect modules, we then describe the transmission process of data on a virtual channel from/to a VM. When outgoing bits are sent from the Virtual Radio Frequency Module (VRFM) of the VM to the virtual channel, they pass through the three modules and stay in a buffer for wrapping (in the lower right corner of Figure 14). When all bits of a packet arrive in the buffer, the virtual channel wraps them into a packet and sends out the packet via UDP. When an incoming UDP packet arrives at the virtual channel, it is put into a queue (lower left of Figure 14) and is decomposed into bits to be sent to the VRFM of the VM via another buffer (on the left of Figure 14).

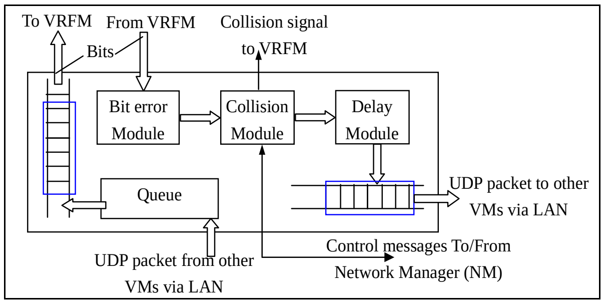
To VRFM
From VRFM
Collision signal to VRFM
Bits
Bit error Module
Collision Module
Delay Module
Queue
UDP packet to other VMs via LAN
UDP packet from other VMs via LAN
Control messages To/From Network Manager (NM)

Figure 14. Virtual Channel

Because VMs run simultaneously, synchronization is needed to ensure that the messages and the operations of VMs are in the same order with that of the target sensor network. The synchronization procedure is as follows: at the startup time, the network manager initializes its table of network status information including the total number of VMs n and the value of the virtual clock of each VM: vt0, vt1… vtn-1. Whenever the VMs run for a predefined interval, T, which is called the *synchronization interval*, they pause and report to the network manager. After every VM has reported to the network manager that its virtual clock has advanced by T, the network manager sends out a broadcast message to inform the VMs to resume running. In addition, the UDP packets on the virtual channel are put in a queue and sorted by their virtual time in ascending order. With the queue and the synchronization interval, the order of operations and messages are ensured to be the same as that on the real network.

4.3 Preliminary Evaluation Results

We have done preliminary evaluation of the VMN with a small number of nodes running a simple query on TinyDB and validated the results of running the query on real MICA2 motes. The query was to report temperature readings of all motes for every epoch of 960ms. This short sampling rate was used to measure the electric currents on real motes at a fine granularity, because the HP 4155A oscilloscope we used was able to measure electric currents at a scale of milliseconds for a period of time of up to two seconds. The two seconds were sufficient for studying the processing of the query because we observed two epochs in each measurement.

We measured the power consumption of this query on a 4-node real mote network using an oscilloscope (HP 4155A) during the query execution (Figure 15). We then ran the query on a 4-node VMN and estimated the power consumption of the query (Figure 16). In our power consumption emulation, we divided the query execution time into several power modes with different operations. These operations are: "Sleeping", "Processing", "Listening", "Sampling" and "Transmitting". Two different operations can occur in one mode, e.g., Processing & Transmitting. The measured electric current in a mode was nearly constant (the range was within +/- 0.3 mA in our experiments).

Figure 15 shows our measurements of four power modes during the query processing in the 4-node real mote network, which were "Listening", "Processing & Transmitting", "Processing & Listening", and "Sampling". Because the sampling rate was short (960ms), the motes did not run into sleeping. In other experiments with a longer sampling rate (>10s), we measured that the average current in sleeping was about 0.0162 mA. All of these results are consistent with the data sheet of MICA2 Motes [4]. These results are also similar to those reported by Madden et al. [11] except one difference is that we did not get the "Snoozing" mode with an average electric current of 4 mA. We are investigating this issue further.

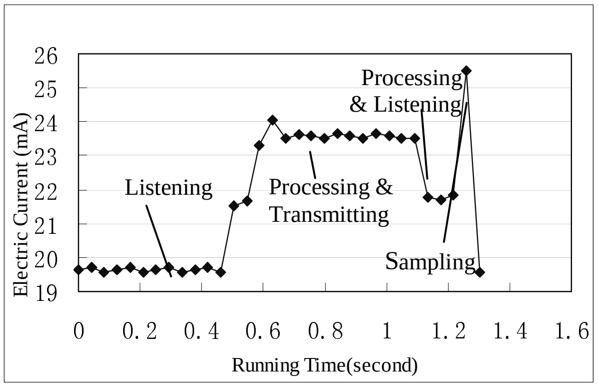

Figure 15: Measured Power Consumption of a MICA2 Mote

Figure 16 shows the estimated power consumption and the estimated query execution time in the 4-node VMN. Compared with the results in Figure 15, the error on query execution time estimation was 1.4-1.34 = 0.06 seconds or 0.06/1.34 = 4.4%. We calculated the power consumption by the sum of (current * running-time), because the number of measurement points was different in the real mote network than in the VMN. The sum of the real measurement was 27.38 mA*seconds, and that of VMN was 28.68 mA*seconds, which resulted in an error rate of 4.72%.

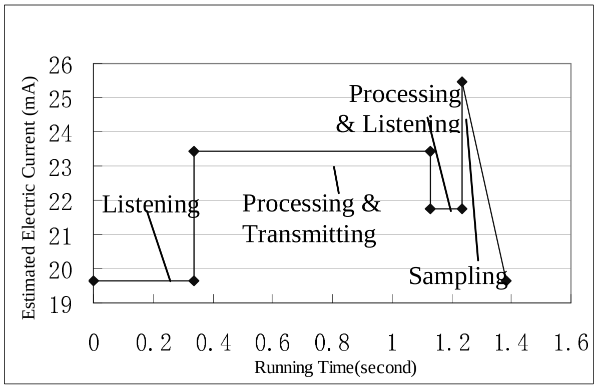

Figure 16: Estimated Power Consumption of a VM

5 Conclusion and Future Work

We have proposed a software framework, MEADOWS, for modeling, emulation, and data analysis of wireless sensor networks. We have reported a case study of real-world data collection and analysis and proposed a preliminary design of data analysis functions for detecting patterns, outliers, and correlations. We have also presented our initial work on a hierarchical power consumption model for sensor databases and on a sensor network emulator using networked PCs. We find that this framework is useful for answering questions about sensor query processing. In addition, the integration of modeling, emulation, and data analysis creates synergy for studying sensor query processing.

Our future work on MEADOWS include (1) implementing our proposed data analysis functions and using the results to cross-validate our modeling and emulation work, (2) conducting more extensive and complex case studies for our sensor database power consumption model and extending the model, and (3) increasing the scale of sensor network emulation and adding node mobility emulation.

Acknowledgement

We collaborated with Pei Zheng at Arcadia University, USA on sensor network emulation. The design of data analysis functions was influenced by discussions with our collaborators at Peking University, China. We thank Jeff Naughton for his helpful comments on the paper. Funding for this work was from Grants HKUST6158/03E, HKUST6161/03E provided by the Hong Kong Research Grants Council (RGC).

References

[1] Jonathan Beaver, Mohamed A. Sharaf, Alexandros Labrinidis, and Panos K. Chrysanthis. Power-Aware In-Network Query Processing for Sensor

Data. The 2nd Hellenic Data Management Symposium, 2003.

[2] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. The 2nd International Conference on Mobile Data Management (MDM), 2001.

[3] Ugur Cetintemel, Andrew Flinders, and Ye Sun. Power-Aware Data Dissemination in Wireless Sensor Networks. The 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access, 2003.

[4] Crossbow Corp. http://www.xbow.com

[5] Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-Query for Wide Area Sensor Network. SIGMOD Conference 2003.

[6] Laura Marie Feeney. An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. Mobile Networks and Applications, 2001.

[7] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. USENIX 2004.

[8] Wendi Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-fficient Communication Protocol for Wireless Microsensor Networks. The 33rd hawaii International Conference on System Sciences, 2000.

[9] Philip Levis, Nelson Lee, Matt Welsh, David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. The 1st International Conference on Embedded Networked Sensor Systems, 2003.

[10] Erran Li and Joseph Halpern. Mimimum-Energy Mobile Wireless Networks Revisited. ICC 2001.

[11] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks. SIGMOD Conference 2003.

[12] NS2. http://www.isi.edu/nsnam/ns/.

[13] Sung Park, Andreas SAvvides, and Mani B. Srivstava. Sensorsim: A Simulation Framework for Sensor Networks. MSWIM, 2000.

[14] Robert Szewczyk, Joseph Polastre, Alan Mainwaring, and David Culler. Lessons from a Sensor Network Expedition. In Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN), 2004.

[15] TinyDB. http://telegraph.cs.berkeley.edu/tinydb/.

[16] TinyOS. http://www.tinyos.net.

[17] Xlisten Program. http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/xbow/tools/src/xlisten/.

[18] Yong Yao and Johannes Gehrke. Query Processing for Sensor Networks. CIDR 2003.

[19] Pei Zheng and Lionel M. Ni. EMPOWER: A Network Emulator for Wireless and Wired Networks. INFOCOM 2003.