1st International Workshop on

Data Management for Sensor Networks

In Conjunction with VLDB 2004

Toronto, Canada
August 30, 2004

Editors:
Alexandros Labrinidis
Samuel Madden

Sponsored by Intel

Message from the Program Chairs

August 4, 2004

The past few years have seen substantial amounts of computer science research on sensor networks. Other subfields have had a number of workshops on the topic (e.g., the Workshop on Wireless Sensor Networks and Applications (WSNA) in 2002 and 2003 and the Sensor Networks Protocols and Applications (SNPA) Workshop in 2002 and 2003, both of which are systems/networking focused). Furthermore, there are now at least two major conferences – the Conference on Information Processing in Sensor Networks (IPSN), started in 2002, and the ACM Conference on Sensor Systems (SenSys), started in 2003. These conferences have published a small number of database papers, but there is no forum for discussion on early and innovative work on data management in sensor networks.

We believe that the Workshop on Data Management for Sensor Networks (DMSN'04) fills a significant gap in the database community by bringing interested researchers together to identify research challenges and opportunities. Specifically, the workshop focuses on data processing and management in networks of remote, wireless, battery-powered sensing devices (sensor networks). The power-constrained, lossy, noisy, distributed, and remote nature of such networks means that traditional data management techniques often cannot be applied without significant re-tooling. Furthermore, new challenges associated with acquisition and processing of live sensor data mean that completely new database techniques must also be developed.

The workshop represents a wide range of topics, including: data replication and consistency in noisy and lossy environments, database languages for sensor tasking, distributed data storage and indexing, energy-efficient data acquisition and dissemination, in-network query processing, integration of sensor network data into traditional and streaming data management systems, networking support for data processing, techniques for managing loss, uncertainty, and noise, query optimization, and privacy protection for sensory data.

As a response to the Call for Papers, the DMSN'04 workshop received 38 abstracts, of which 25 materialized as full papers by the submission deadline. During the review process, each paper was reviewed by at least three PC members or external reviewers, resulting in the acceptance of 15 papers.

We are grateful to many people who contributed to the content and organization of the workshop. First of all we would like to thank the steering committee: Panos Chrysanthis, Mike Franklin, Johannes Gehrke, and Joe Hellerstein. Their advice and support proved invaluable. We are also grateful to the Program Committee members and the external reviewers for helping us put together a high-quality program for the workshop. Intel Corporation's generous donation enabled us to support activities that would not have been possible with just the registration income, like the best paper award and student travel grants. We would also like to thank Surajit Chaudhuri of Microsoft Research and the CMT team for allowing us to use the Conference Management Toolkit service and for their assistance. Finally, we would like to thank the VLDB'04 organizing committee and in particular John Mylopoulos, Alberto Mendelzon, S. Sudarshan, Mariano Consens, Grant Weddell, and Iluju Kiringa.

Alexandros Labrinidis and Sam Madden

August 2004

DMSN ’04 Organization

Program Chairs:
Alexandros Labrinidis, *Univeristy of Pittsburgh*
Samuel Madden, *MIT*

Steering Committee:
Panos Chrysanthis, *Univeristy of Pittsburgh*
Michael J. Franklin, *UC Berkeley*
Johannes Gehrke, *Cornell University*
Joseph M. Hellerstein, *Intel Research and UC Berkeley*

Program Committee:
Philippe Bonnet, *University of Copenhagen*
Luc Bouganim, *INRIA*
John Byers, *Boston University*
Ugur Cetintemel, *Brown University*
Panos Chrysanthis, *University of Pittsburgh*
Isabel Cruz, *University of Illinois at Chicago*
Michael J. Franklin, *University of California, Berkeley*
Minos Garofalakis, *Bell Labs*
Johannes Gehrke, *Cornell University*
Phil Gibbons, *Intel Research*
Ramesh Govindan, *University of Southern California*
Carlos Guestrin, *Intel Research and CMU*
Joseph M. Hellerstein, *Intel Research and UC Berkeley*
Wei Hong, *Intel Research*
Zachary G. Ives, *University of Pennsylvania*
Christian S. Jensen, *Aalborg University*
George Kollios, *Boston University*
Alexandros Labrinidis, *University of Pittsburgh*
Qiong Luo, *HKUST*
Samuel R. Madden, *MIT*
Sharad Mehrotra, *University of California at Irvine*
Silvia Nittel, *University of Maine*
Sunil Prabhakar, *Purdue University*
Mema Roussopoulos, *Harvard University*
Timos Sellis, *National Technical University of Athens*
Anthony Stefanidis, *University of Maine*
Matt Welsh, *Harvard University*
Adam Wolisz, *Technical University of Berlin*
Vladimir Zadorozhny, *University of Pittsburgh*
Feng Zhao, *Microsoft Research*

External Reviewers
Reynold Cheng
Rashmi Chitrakar
Qi Han
Iosif Lazaridis
Kostas Patroumpas
Roberto Tamassia
Xingbo Yu

Contents

Spatio-temporal Techniques

BINOCULAR: A System Monitoring Framework

Fatih Emekci† Sezai E. Tuna‡ Divyakant Agrawal† Amr El Abbadi†

Department of Computer Science† Department of Elec. and Comp. Engineering‡
University of California Santa Barbara
Santa Barbara, CA 93106, USA
{fatih,agrawal,amr}@cs.ucsb.edu† emre@ece.ucsb.edu‡

Abstract

Recent advances in hardware technology facilitate applications requiring a large number of sensor devices, where each sensor device has computational, storage, and communication capabilities. However these sensors are subject to certain constraints such as limited power, high communication cost, low computation capability, presence of noise in readings and low bandwidth. Since sensor devices are powered by ordinary batteries, power is a limiting resource in sensor networks and power consumption is dominated by communication. In order to reduce power consumption, we propose to use a linear model of temporal, spatial and spatio-temporal correlations among sensor readings. With this model, readings of all sensors can be estimated using the readings of a few sensors by using linear observers and multiple queries can be answered more efficiently. Since a small set of sensors are accessed for query processing, communication is significantly reduced. Furthermore, the proposed technique can also be beneficial at filtering out the noise which directly affects the accuracy of query results.

1 Introduction

Due to advances in miniaturization, low power, and low cost design of sensors, large-scale sensor networks are being deployed to monitor systems. Examples include environment monitoring on Great Duck Island and James Reserve [2, 7]. In sensor networks, each sensor can be modeled as a full fledge computer with computational, communication, and sensing capabilities. However, these sensors are subject to several constraints such as limited power, high communication cost, low computation capability, presence of noise in readings and low bandwidth. Because of these constraints, techniques for distributed systems, databases, and data stream management cannot be applied directly to sensor networks. In particular, any system dealing with sensor generated data needs to pay attention to these constraints.

There have been many related research efforts in the database and data stream management areas. Traditional database management aims to reduce the query response time using indexes. On the other hand, the main goal in the context of data streams is to reduce the storage and computational cost and give fast approximate answers to queries. However, monitoring a system (a system can be any measurable phenomenon in the physical world) with queries is quite different from query processing over data streams and database management systems. The cost of query execution in sensor networks is not only bounded by computational and storage costs but also bounded by data collection cost. In data stream and database management systems, however, data collection cost is not taken into account explicitly; instead it is assumed that data is already available. This assumption is quite reasonable in database and data stream management systems which are built on wired systems that do not have energy and bandwidth constraints. This, however, is not true in sensor networks where each sensor is run by ordinary batteries and has energy and bandwidth constraints which directly affect the quality of monitoring.

Recently, there are several proposals to deal with sensor generated data aiming to reduce the cost of data collection to prolong the lifetime of the sensors. In [5], researchers proposed the Fjords architecture for managing multiple queries over many sensors. They collect readings of all sensors and try to compute common subexpressions among queries only once. There are several researches try to compute queries in-network such as [10, 6, 11, 12]. In general, in-network aggregation can reduce the power usage by pushing part of the computation into the network. However, these works only consider aggregation queries and do not consider multi-queries. Lazaridis and Mehrotra [4] proposed to compress the raw data at each sensor node, then the compressed data is sent to the basestation when the precision is out of bound. Goel and Imielinski [3] proposed a prediction technique to monitor environment by

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

applying MPEG techniques in prediction.

When monitoring the physical environment, there are physical rules relating to data originating from different data sources (there is a physical rule between readings of sensors), which is different from data streams and databases. Most of the time, these physical rules can be discovered and modeled using correlations among sensor readings. Once this model is known, the query processor can use this model to observe the environment by collecting data from a few sensors instead of all of them. Furthermore, this model can be used to reduce the noise in the measurements. Our main observation behind this work is that if two sensors are close to each other, then there is a physical rule between their measurements. And this physical rule can be discovered with temporal, spatial and spatio-temporal correlations among sensor readings. For example, if two sensors are 100 meters apart from each other then their temperature measurements are correlated. Therefore, if these correlations are determined and modeled with historical data, the query processor can use that model to estimate the readings of all sensors using the readings of a few sensors. Formally, if a system is identified and modeled using a linear model, then that linear model can be used to observe all readings using only a subset of the sensors.

Hence, the properties of BINOCULAR can be summarized as follows:

- BINOCULAR is a monitoring system where users pose continuous queries to monitor the physical environment. Therefore, it is a multi-query processing platform.
- BINOCULAR models the readings of sensors as a linear system to observe readings of all sensors with a small set of sensor readings. Therefore, it is an energy efficient monitoring system.
- BINOCULAR improves the quality of the answers of queries by reducing the noise over sensor readings using linear observers.
- BINOCULAR balances energy consumption among sensors while extending their lifetime.

The rest of the paper is organized as follows: Section 2 formalizes the problem of monitoring systems with queries and gives a solution overview. Linear observers are introduced in Section 3. Section 4 describes the proposed query processing technique. Section 5 reports the results of our preliminary experimental evaluations. Section 6 concludes the paper, presents future work and discusses open research problems.

2 Problem Formulation and Solution Overview

Given a set of sensors BINOCULAR divides them into two types: *working* and *sleeping sensors*. In order to estimate the readings of all sensors, BINOCULAR only collects data from the working sensors and uses a system model to estimate the readings of the sleeping sensors. A system model expresses an estimate based on the current readings of the working sensors (u_k) and the current estimate of the sleeping sensors (x_k). In a linear system model this is expressed by a linear relationship between x_{k+1} and (x_k, u_k) based on a linear correlation using system matrices A and B. This can be expressed as follows:

$$x_{k+1} = Ax_k + Bu_k, \quad (1)$$

where $x \in R^{n\times 1}$ is the *state* (the estimated readings of the sleeping sensors), $u \in R^{m\times 1}$ is the *input* (the actual readings of the working sensors), $A \in R^{n\times n}$ is the system matrix, $B \in R^{n\times m}$ is the input matrix, m is the number of the working sensors and n is the number of the sleeping sensors. Matrices A and B are derived using a system identification toolbox e.g, Matlab [8], which uses some historical data set to derive them. Thus, queries can be thought of as a function of these states and inputs. If the correlations among sensor readings can be captured by a perfectly linear model (e.g, illumination), then we can estimate the exact readings of all sensors using the working sensors. However, in most cases these correlations cannot be captured by a linear model but need to be approximated by a linear model (e.g, temperature readings). In this case we have to collect readings of all sleeping sensors periodically (every D time units) in order to avoid error accumulation. The appropriate value of D can be derived from historical data.

However, if the readings of sleeping sensors can be estimated by a small subset of them (*linear observers*), then collecting readings from that subset is enough to estimate the readings of all sleeping sensors. This scheme requires the system model to be continuously used to derive the estimated readings of all sensors based on the readings of the working sensors. The linear observers are activated periodically every D time units for a short period to recalibrate the errors in the system model. In this paper we use a *linear observer* to estimate the readings of all sleeping sensors. A linear observer is a linear system built from the original system model given by (1). Given a system model in the form of (1) and a vector of readings of a subset of the sensors y_k of size p ($y_k = Cx_k$ where $C \in R^{p\times n}$ and $C(i,j)$ is 1 if sensor j is in the observer set and it is the only 1 in that row), our goal is to determine whether all sleeping sensors can be observable via the y_k, referred to as the observer. If it is possible to observe, then we use the linear observer specified by the $y_k = Cx_k$

Example Consider an environment monitored by three sleeping sensors and one working sensor with a system model:

$$x_{k+1} = Ax_k + Bu_k, \quad (2)$$

where x_k is a 3×1 matrix such that $x_k(i)$ is the estimated reading of sleeping sensor i and u_k is 1×1 matrix and $u_k(1)$ is the actual reading of the working sensor at time k. A and B are matrices given by a system identification toolbox based on historical data. If this model is accurate

then actual readings of sleeping sensors need not to be collected, because all of the readings of the sleeping sensors can be estimated from the reading of the working sensors. However, if the model is a linear approximation, then the error given by the system model will accumulate over time. In order to decrease such an error we collect readings of the three sleeping sensors every D time units. However, if the readings of these three sensors are observable by any of them, then the readings of the other two sleeping sensors can be estimated by collecting data from that observer. Then the question becomes: Is this system model observable via $y_k = Cx_k$ where $C = [0\ 0\ 1]$ (the reading of the third sensor)? If it is observable then we can construct a linear observer and use that to estimate the readings of the other two sleeping sensors.

The problem of monitoring systems with queries can be formulated as follows: Given a set of historical readings of sensors, and a set of continuous queries to monitor,

- build a scheduler to schedule sensors as a working or a sleeping sensor,
- discover a linear model between the readings of working sensors and sleeping sensors,
- construct linear observers and an observer scheduler

to execute queries while reducing and balancing the energy consumption.

In this paper, we assume that the working and sleeping sensors are statically assigned. In addition, we assume working sensors are not subject to energy constraints. However, these assumptions are not realistic. Hence, we need a scheduler to schedule sensors as a working or a sleeping sensor. The choice of working sensors to model the system more accurately and a scheduling technique needs further research.

3 Formal Model For Observers

Given a system model and a C matrix where $y_k = Cx_k$, we can construct a linear observer if the pair $(A,\ C)$ is an observable pair which is defined as follows [1]:

Definition 1 *The pair $(A,\ C)$ is said to be an observable pair if the matrix*

$$\begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

is full column rank.

Theorem 1 states how and why a linear observer can be constructed with an observable pair $(A,\ C)$.

Theorem 1 *Given a system model and a y_k as follows:*

$$x_{k+1} = Ax_k + Bu_k \quad (3)$$
$$y_k = Cx_k\ , \quad (4)$$

The states of all sleeping sensors can be observed via y_k using the following system called linear observer if the $(A,\ C)$ pair is observable:

$$\hat{x}_{k+1} = A\hat{x}_k + L(y_k - C\hat{x}_k) + Bu_k \quad (5)$$

where L is an observer matrix.

Proof 1 *If we define the error as $e := x - \hat{x}$, then we can write using (3), (4), and (5)*

$$\begin{aligned} e_{k+1} &= x_{k+1} - \hat{x}_{k+1} \\ &= Ax_k - A\hat{x}_k - L(y_k - C\hat{x}_k) \\ &= A(x_k - \hat{x}_k) - LC(x_k - \hat{x}_k) \\ &= (A - LC)(x_k - \hat{x}_k) \\ &= (A - LC)e_k\ . \end{aligned} \quad (6)$$

Equation (6) tells us that if all the eigenvalues of the error gain matrix *$A - LC$ are with magnitude strictly less than unity, i.e., $A - LC$ is* Schur, *then we can claim $|e_k| \to 0$ and hence $\hat{x}_k \to x_k$ as $k \to \infty$. Now, the question is can one always pick L so that $A - LC$ has this property. It is possible to pick such an L only if the $(A,\ C)$ pair satisfies the observability condition [1]. Indeed, if $(A,\ C)$ is an observable pair then the eigenvalues of $A - LC$ can be arbitrarily placed (all of them can be set to zero for instance) with a proper choice of L and the system called observer given by Equation (5) can be used to estimate all of the states.*

Suppose we are given the system matrix A and we can measure two of the sleeping sensors, say $x(1)$ and $x(3)$, where $x = [x(1)\ x(2)\ \ldots\ x(n)]^T$. Then we can construct an observer with a decaying observer error if $(A,\ C)$ pair is observable where

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \end{bmatrix}$$

Therefore, we can estimate all of the states by collecting the readings of sleeping sensors 1 and 3 via the linear observer in Theorem 1.

Using the same observer will drain the energy of the sensors designated as observers. Therefore, we should find a set of observers and switch among them periodically. In order to do this, we should have an *observer determination technique*. The goal of the observer determination process is to give a set of C matrices such that the system is observable via $y_k = Cx_k$. The number of possible different observers is 2^N, where N is the number of sleeping sensors. For small N, it is possible to test all C matrices whether (A, C) is observable or not. However, for large N it is impossible to test all of the possible 2^N cases. Therefore, we need a heuristic to find a set of observers in polynomial time.

Discussion So far, we use the system model to estimate the readings of sleeping sensors with the readings of the working sensors. However, the linear observer given by

(5) can be used to estimate the readings of all the sleeping sensors with readings of the working sensors and sensors in that observer. Although it seems inefficient in terms of energy consumption, it may be beneficial to get more accurate results. We leave these experimental evaluations as future work. In general, we need a set of observers and an observer scheduler to select a linear observer to estimate the readings of all the sensors. Given a set of observers, we can not switch them at any time. Formally, suppose we have a set of observers defined by $\{L_1, L_2, \ldots, L_m\}$ and $\{C_1, C_2, \ldots, C_m\}$. Let the resultant error gain matrices be $\{H_1, H_2, \ldots, H_m\}$, where $H_i = A - L_i C_i$ is Schur for all $i \in \{1, 2, \ldots, m\}$. Although each error gain matrix is Schur, there is no guarantee that the error decays to zero under arbitrary switching between observers. Intuitively, if we stay at an observer long enough before switching to another one then we still should have a decaying observer error. Therefore, we need an observer scheduler to schedule observers appropriately. Throughout the paper, we assume that we are given a set of observers and a time matrix T such that given two observers O_i and O_j, T_{ij} is the time to switch from O_i to O_j.

4 Query Processing

The readings of the sensors are needed to answer queries. Recall there are two methods to estimate the readings of the sensors:

- **Method 1:** Use the system model continuously and access any of the observers for a short time every D time units to recalibrate the errors in the system model.
- **Method 2:** Use only observers such that at any time only one of the observers is accessed and the estimated readings of all of the sleeping sensors are derived from that observer.

The job of the query processor is to schedule observers to balance energy consumption among sensors. The observer scheduler chooses the observer with the highest score where the score is defined as the $Average\ Energy\ Consumption\ to\ Collect\ Readings\ of\ the\ Observer$ divided by the $Average\ Available\ Energy\ of\ Sensors\ in\ the\ Observer$. At any time the observer scheduler chooses the observer with the lowest cost and the highest energy.

Observer scheduling is quite straightforward in Method 1. The job of the observer scheduler is to select the observer with the highest score every D time units and execute it for a short time to estimate readings of all sleeping sensors in *Method 1*.

Since the observer scheduler cannot switch observers at any arbitrary time, observer scheduling is not straightforward in *Method 2*. An example of observer scheduling for Method 2 is shown in Algorithm 1, which uses an eager approach to schedule observers assuming the T matrix is known. Let O_i and O_j be the two observers with the highest scores and T_{ij} is the time needed to execute O_i before switching from O_i to O_j. Algorithm 1 uses O_i for T_{ij} time units. After that, it calculates the new scores and repeats the same process based on the new scores.

Algorithm 1 Observer Scheduler Algorithm

```
1: Input:
2: T: Time matrix shows time needed before switching observers;
3: O: Set of observers O = O_1, ..., O_o
4: Procedure:
5: while There is a query do
6:    Calculate the score of each observer in O
7:    Find two observers O_i and O_j with highest scores
8:    Use O_i for T_ij
9: end while
10: End Procedure
```

5 Preliminary Results

The main motivation behind BINOCULAR is modeling the correlations among the sensor readings and use that model in query processing. Thus, we conducted some preliminary experiments over a real temperature dataset from the Tropical Atmosphere Ocean Project [9] to show correlations among the readings of sensors can be modeled. We took the average daily temperature readings of 20 sensors for 1000 days. After that we randomly select one sensor as a working sensor and the remaining as sleeping sensors. Based on the first 300 days we build a system model and try to measure the average errors of readings of sensors for each time interval during the remaining 700 days In addition to this, we collect actual readings of all sleeping sensors at time 300 in order to avoid the error accumulation (we did not use observers). The average error for time t is calculated as follows: $((1/19) * \sum_{i<20} \mid \hat{s_i} - s_i \mid / s_i) * 100$ where $\hat{s_i}$ is the estimated reading and s_i is the actual reading of a sensor i at time t. Results are shown in Figure 1, where the x axis represents time and the y axis represents the average percentage error. With the system model and one working sensor, we can estimate the reading of all sensors within 3.5 percent error on the average (this is $\approx \mp 1^o C$). This preliminary result shows that we can model the readings of sensors and use that model in order to process queries resulting in significant savings in energy consumption. In this study, for example, we need only one sensor to be working instead of all of the 20 sensors.

6 Conclusion and Future Work

In this paper we presented our system monitoring framework, BINOCULOR, and also showed some preliminary results. BINOCULAR uses a linear model between working sensors and sleeping sensors to answer queries while using a small set of sensors. We introduced the notion of linear observers to account for the fact that the linear model will always be an approximation of the physical environment. By using the linear observers, the modeling error can be reduced exponentially over time. This results in less communication cost and prolongs the lifetime of sensors. Although we presented the general framework, there are still open research questions. These include:

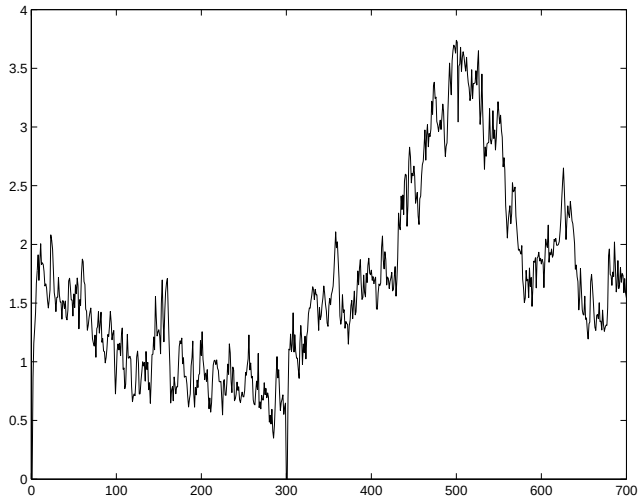


Figure 1: Average error (in percentage) of sensor readings

- How to choose working sensors and balance energy usage among all sensors?
- How to determine the set of observers in large scale networks?
- When and how to switch observers to balance energy consumption?
- How to adapt the system model to changes which is needed for systems with mobile sensors?

References

[1] Pans J. Antsaklis and Anthony N. Michel. *Linear Systems*. The McGraw-Hill Companies, Inc.

[2] A. Cerpa, J. Elson, D. Estrin, L. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. *In ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, pages 88–97, 2001.

[3] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from mpeg. Technical Report DCS-TR-438, Rutgers University, June 2001.

[4] Iosif Lazaridis and Sharad Mehrotra. Capturing sensor-generated time seires with quality guarantees. *International Conference on Data Engineering (ICDE 2003)*, pages 429–, March 2003.

[5] S. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. *International Conference on Data Engineering (ICDE 2002)*, pages 555–566, March 2002.

[6] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *5th Symposium on Operating Sytsem Design and Implementation*, December 2002.

[7] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. *In ACM Workshop on Sensor Networks and Applicaions*, 2002.

[8] Matlab. http://www.mathworks.com/products/sysid/.

[9] M. J. McPhaden. Tropical atmosphere ocean project. *Pacific marine environmental laboratory*. http://www.pmel.noaa.gov/tao/.

[10] Anantha Chandrakasan Wendi Rabiner Heinzelman and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *HICSS*, January 2000.

[11] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[12] Yong Yao and Johannes Gehrke. Query processing for sensor networks. *In CIDR 2003*, January 2003.

Adaptive Sampling for Sensor Networks

Ankur Jain

Computer Science
University of California, Santa Barbara
Santa Barbara CA 93106
ankurj@cs.ucsb.edu

Edward Y. Chang

Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara CA 93106
echang@ece.ucsb.edu

Abstract

A distributed data-stream architecture finds application in sensor networks for monitoring environment and activities. In such a network, large numbers of sensors deliver continuous data to a central server. The rate at which the data is sampled at each sensor affects the communication resource and the computational load at the central server. In this paper, we propose a novel adaptive sampling technique where the sampling rate at each sensor adapts to the streaming-data characteristics. Our approach employs a Kalman-Filter (KF)-based estimation technique wherein the sensor can use the KF estimation error to adaptively adjust its sampling rate within a given range, autonomously. When the desired sampling rate violates the range, a new sampling rate is requested from the server. The server allocates new sampling rates under the constraint of available resources such that KF estimation error over all the active streaming sensors is minimized. Through empirical studies, we demonstrate the flexibility and effectiveness of our model.

1 Introduction

As sensor networks grow in size, bandwidth allocation becomes increasingly critical. A sensor network needs to allocate its bandwidth to maximize total information gain. A desirable bandwidth allocation scheme should distribute the given bandwidth such that it is sensitive to streaming data characteristics, query precision, available resources (communication, power, CPU), and sensor priority (data from some sensors might be more important than others) [9, 2]. We can further motivate this research using the following two examples.

- Wireless sensor-networks are being used for habitat monitoring applications. In [11], sensors registering light, temperature, and sound are deployed in burrows of Storm Petrels (a seabird) for monitoring purposes. During the day time, the burrows are expected to be empty, and thus we can have a low sampling rate. However, if some unusual measurements are recorded at some burrows (say abrupt increase in sound levels), it would be desirable to collect samples from them more frequently than the other burrows.

- In video surveillance applications like [6], multiple cameras are mounted at key locations to monitor activities of vehicles and people in a parking lot. If a camera shows a vehicle exhibiting unexpected behavior (random swirling, speeding), the camera's sampling rate should be increased by decreasing the sampling rates of the other cameras that are not observing abnormal behavior.

A naïve solution to the above-mentioned problems is over-sampling [12]. However this comes at increased cost of resources, namely:

- *CPU* — The CPU at the central server might have to process unnecessary data from numerous sources, but this would not affect the result significantly.

- *Network Bandwidth* — The communication channel would be transmitting unnecessary data. Moreover, in cases of low bandwidth networks the option of over-sampling might not be available at all.

- *Power Usage* — Power conservation is critical for wireless sensors. Over-sampling leads to increased power consumption of a sensor's measuring devices, radio transmitter, and processing unit.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

There has been a significant amount of work in the sensor-network resource management. The key aspect that differentiates this work from the prior efforts lies in data collection (sensing). We adjust sampling rates (sensing rates) at sensors to adapt to data characteristics. Traditional methods (e.g., load-shedding [17] and adaptive precision setting [13]) collect data at a peak sampling rate and then determine whether collected data should be dropped to conserve resources. Even though the filtering and load-shedding approaches can reduce bandwidth consumption in the transmission phase, excessive sampling rates incur high cost in data collection and processing (to determine what data to drop) at the sensors. The adaptive sampling scheme proposed in this paper adjusts the data collection rate according to data characteristics. Therefore, resources are conserved and better utilized working only on data relevant to the queries.

Our *general* and *adaptive* sampling approach adjusts the sampling interval SI (the time interval between two consecutive samples) collectively. At the sensors, the SI is adjusted depending on the streaming data characteristics. The remote source is allowed to modify the sampling interval independently within a specified *Sampling Interval Range* (SIR). If the desired modification in the SI is more than that allowed by the SIR, a new sampling interval is requested from the server. At each sensor, we use the Kalman Filter estimator to predict the future values of a stream based on those seen so far. Large prediction errors signify unexpected behavior of the streaming data or an interesting event. The sampling interval is adjusted based on the prediction error. At the server, new sampling intervals are allocated to the requesting sensors based on available bandwidth, network contention, and streaming source priority.

We consider a simple network model to conduct the experiments, where all the streaming sources connect to a single network channel. The server continuously monitors the usage of this network channel and allocates new bandwidth based on its availability. These kinds of networks are prevalent in video surveillance, object tracking and process control (automated meter reading, building automation). Extending our current architecture to multi-hop sensor networks is a part of future research.

The main contributions of our work can be summarized as below:

- We propose a model which, is *adaptive* to adjust the sampling rate based on the input data characteristics and *general* to map to linear (as well as non-linear) problems without many major modifications.
- Our method utilizes the given bandwidth judiciously such that more important sources get more bandwidth by reducing the bandwidth of less important ones.
- Our method allows the *capability* at the remote site to adjust the sampling rate (to a certain extent) independently without the central server mediation to improve response time.
- Finally, we propose an *optimal estimation* scheme (Kalman Filter) that can be used on the sensor side to assess data arrival characteristics.

2 Related Work

The resource management problem in data streaming has been studied mainly from the perspective of data filtering [5, 13]. It has been shown that using adaptive precision bounds [13], unusual *trends* in the streaming data can be captured (the data is updated to the server only when it falls out of an adaptive precision bound) at low communication costs. However, due to uniform sampling, the approach does not have the capability to utilize a given bandwidth to maximize the information gain.

The adaptive sampling approach proposed in [10] considers only the network channel contention while adjusting the sampling rate. The sensors check for the network channel contention before putting the data on it and reduce the sampling rate if the contention and data-tuple drop rate is high. This reduces the overall load on the network channel and achieves a better delivery rate at the server. The proposed approach does not utilize the network channel judiciously, and it uses adaptive sampling only when the network channel becomes congested and requires load-shedding.

The use of adaptive sampling and bandwidth management in sensor networks has been very well motivated in [12, 3, 14, 9]. However a scalable method applicable in a distributed environment is still not available.

As we have discussed in Section 1, the problem of adaptive sampling is not the same as that of load-shedding [17]. First, to the best of our knowledge, none of the load-shedding techniques have yet used prediction/estimation models. Second, while load-shedding modules are activated only when the load on the system increases beyond what it can handle, adaptive sampling modules are executed during the lifetime of a stream. In the event of network congestion, the load-shedding module would reduce the data *transmission* rate of the sensor by randomly dropping tuples, whereas an adaptive sampling technique would reduce the data *collection* rate in such a way that higher priority data receive a higher proportion of the available bandwidth.

3 The Kalman Filter

The Kalman Filter was introduced in 1960 by R. E. Kalman [7] as a recursive solution to the discrete-data linear filtering problem. Since then, it has found application in the fields of data smoothing, process es-

timation, and object tracking, to name a few. The traditional Kalman Filter is a linear algorithm that estimates the internal state of a system based on a prediction/correction paradigm. Below, we provide a brief overview of the Kalman Filter's mathematical formulation, for more details refer [18].

The Kalman Filter comprises a set of mathematic equations that provide a recursive solution to the least-squares method. The system model is represented in the form of the following equations:

$$\boldsymbol{x_{k+1}} = \boldsymbol{\phi_k x_k} + \boldsymbol{w_k} \tag{1}$$

$$\boldsymbol{z_k} = \boldsymbol{H_k x_k} + \boldsymbol{\nu_k} \tag{2}$$

where

$\boldsymbol{x_k} =$ *state vector of the process*
$\boldsymbol{\phi_k} =$ *state transition matrix relating* x_k *to* x_{k+1}
$\boldsymbol{w_k} =$ *process model noise*
$\boldsymbol{z_k} =$ *measurement vector*
$\boldsymbol{H_k} =$ *matrix relating system state and measurement vector*
$\boldsymbol{\nu_k} =$ *measurement noise*
$k =$ *discrete time index*

The prediction $\hat{\boldsymbol{x}}_{\boldsymbol{k}}$ is based on a linear combination of previous prediction/estimation and the weighted prediction error. This error is called *innovation* $\boldsymbol{\psi_k}$, which is calculated as follows:

$$\boldsymbol{\psi_k = z_k - H_k \hat{x}_k^-}. \tag{3}$$

The value of the weight is called *Kalman Gain* $\boldsymbol{K_k}$ which is adjusted with each measurement. The prediction is calculated as follows:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}_{\boldsymbol{k}}^- + \boldsymbol{K_k \psi_k}. \tag{4}$$

Applying the least-square method we get

$$\boldsymbol{K_k} = \boldsymbol{P_k^- H_k^T}(\boldsymbol{H_k P_k^- H_k^T} + \boldsymbol{R_k})^{-1}. \tag{5}$$

$$\boldsymbol{P_k} = (\boldsymbol{I} - \boldsymbol{K_k H_k})\boldsymbol{P_k^-}. \tag{6}$$

where, $\boldsymbol{P_k}$ and $\boldsymbol{R_k}$ are the error covariance and measurement noise covariance matrices respectively (the superscript denoting the *a priori* state of the matrices).

The advantage of using the Kalman Filter is that it gives satisfactory results even when we cannot model the process accurately (i.e., when the values of matrices $\boldsymbol{\nu_k}$ and $\boldsymbol{\phi_k}$ are unknown) and that the innovation sequence can be used to evaluate the performance of the estimation process.

There is a wide spectrum of filtering solutions available which work on the estimation/correction paradigm and can be substituted for Kalman Filter in our proposed architecture. However, we support the use of Kalman Filter as it can be easily customized to provide good results on a wide range of streaming sensor data and produce *unbiased* estimates even when the incoming data have high variance. Biased algorithms (like Exponential Weighted Moving Average, EWMA) might not be the best choice when incoming data has high variance. Error estimates can be further improved using more sophisticated solutions like Particle Filter [8] or condensation (conditional density propagation) [4] as they work on non-Gaussian noise processes and multi-modal state propagation. Such algorithms are likely to provide better results as real-life data are not Gaussian, however this performance upgrade comes at increased cost of computational resources. Most of the sensing devices have limited computational capacity and selecting the best filtering solution is subject to the availability of the resources. The advantage of using Kalman Filter here is that the computational complexity can be easily manipulated by adjusting the number of state variables in the state propagation equation.

4 Our Approach

We now present our adaptive sampling approach in a distributed stream environment. We consider an environment where numerous sensors continuously stream updates to a central server. For example, a system of sensors that continuously measure the location of a moving object in two dimensions (one sensor for each object). Our adaptive approach would distribute the available bandwidth automatically in such a way that sensors monitoring objects showing increased activity have shorter time intervals between successive (low sampling interval) measurements whereas those with reduced activity have longer time intervals (high sampling interval). This way, the trajectory generated at the server by interpolating the measurements from the sensors would be closer to the original trajectory than that obtained by performing uniform sampling.

To maintain simplicity, we do not assume the presence of any data filtering or load-shedding modules in our discussion. Thus, the data-sampling interval is the same as the data-transmission interval of the sensor. We interpret the sampling interval as the number of time units between two successive measurements.

There are two main modules in the system, one on the sensor side and the other on the server side. Due to the space limitations, we describe each of them only briefly. To simplify our discussion, we assume in this paper that the tuple size over all the sources is the same, and hence the bandwidth consumption is directly proportional to the sampling interval at the streaming sources.

4.1 Source Side Module

Let SI_i denote the current sampling interval at source S_i (i is the i^{th} source) which, is the number of time units between two consecutive measurements. Let SIR_i denote the range within which, the sampling interval can be adjusted by the source without any server mediation and SI_i^{last} denote the latest value of sampling interval received from the server. (We currently assume static SIR_i's.) Let $SI_i^{desired}$ denote the desired sampling interval based on the KF prediction error. Sensor S_i need not contact the server for additional bandwidth provided that

$$(SI_i^{last} - SIR_i/2) \leq SI_i^{desired} \leq (SI_i^{last} + SIR_i/2). \quad (7)$$

If $SI_i^{desired}$ satisfies Equation 7 then SI_i takes the value of $SI_i^{desired}$. This scheme helps the source to capture unexpected data trends immediately as the server grants over the network could be delayed due to network congestion or unavailability of resources. Each data tuple sampled by S_i is forwarded to a Kalman Filter KF_i which, provides with the innovation $\boldsymbol{\psi_t^i}$ value (Section 3). The estimation error δ_i at any instant t is then calculated as:

$$\delta_t^i = sqrt(trace(\boldsymbol{\psi_t^i}(\boldsymbol{z_t^i})^{-1})^2). \quad (8)$$

We multiply the innovation (error in prediction) by the inverse of the measurement matrix to get the fractional error ($\boldsymbol{\psi_t^i}$ and $\boldsymbol{z_t^i}$ are column matrices). We take the square of the matrix to eliminate any negative values. Finally the square root of the trace gives the fractional error over all the variables in the measurement matrix.

S_i maintains a sliding window of size W_i that holds the last W_i values of the estimation error. If n_j^i is the j^{th} element of the sliding window at S_i (n_1^i being the latest element), total error Δ_i over the sliding window is calculated as:

$$\Delta_i = \frac{\sum_{j=1}^{j=W_i} n_j^i/j}{\sum_{j=1}^{j=W_i} 1/j}. \quad (9)$$

Equation 9 ensures that the newer values in the window have higher weight.

User parameters λ_i and θ_i control the dynamics of SI_i. Each time Δ_i is calculated, a new sampling interval SI_i^{new} is generated as follows:

$$SI_i^{new} = SI_i + \theta_i * (1 - e^{f_i}). \quad (10)$$

where $f_i = \frac{\Delta_i - \lambda_i}{\lambda_i}$. Equation 10 ensures sharp fall and gradual rise in the sampling interval due to the exponential factor that helps improving the response time of the system. If SI_i^{new} satisfies Equation 7, then the sampling interval is assigned this new value; otherwise, a new sampling interval is requested from the server. The source requests the change is the sampling interval ΔSI_i such that

$$\Delta SI_i = SI_i^{new} + SIR_i/2. \quad (11)$$

In addition the source also sends the fractional error f_i for each request of decrease in the sampling interval.

4.2 Central Server module

We now discuss the sampling rate allocation policy at the central server. The allocation algorithm is executed each time a request for decrease in sampling interval (increase in the sampling rate) is received from a streaming source. The server maintains a variable R_{avail} that holds the amount of communication resource available at any time. When a source reports about an increase in its sampling interval, the server immediately adds the proportional amount of resource units to R_{avail} and sends an acknowledgment to the source. Any request for a decrease in a sampling interval is added to a job-queue that is processed continuously by a separate thread.

Each job J_p in the job-queue has 5 attributes which, are described below:

1. *Fractional error* f_p is received from the source when it sends a request.
2. *Request* Req_p is the units of resource requested.
3. *History* h_p is the age of the request in the job-queue. Its value is incremented by unity each time the job-queue is processed.
4. *Grant* g_i is the fraction by which, the Req_p has been satisfied so far.
5. *Query Weight* w_p the weight of the streaming source from the query evaluator.

Assuming that the error f_p is reduced to zero if resource request J_p is satisfied completely, we can formulate a linear optimization problem, minimizing the total error over all the jobs. If J_p is allocated A_p units of resources, then the residual error after satisfying the job is proportional to $(1 - A_p/Req_p)$. Jobs having higher f_p, h_p, and w_p are given more priority than others, whereas the priority varies inversely with g_p. We normalize each attribute by dividing it by the sum of its value in all the jobs in the job-queue. Thus the objective function can be formulated as:

$$\min_{A_p} \left(\frac{f_p}{\sum f_p} * \frac{h_p}{\sum h_p} * \frac{w_p}{\sum w_p} * \frac{\sum g_p}{g_p} * \left(1 - \frac{A_p}{Req_p}\right)\right) \quad (12)$$

$$\text{s.t.} \begin{cases} \sum A_p \leq & R_{avail} \\ 0 < A_p \leq & Req_p \end{cases} \quad (13)$$

Constraints in Equation 13 ensure that the sum of the allocated resources is less than that of the total available and that each grant is less than its request. Once

the optimization problem is solved, the resource units are distributed to the requesting sources and the job-queue attributes are updated accordingly.

5 Results

In this section we present the preliminary results of our distributed adaptive sampling system. We performed the experiments on data produced by the oporto realistic spatio-temporal data generator [15]. We recorded the trajectories (in 2 dimensions) of 12 shoals produced by the generator for $3,000$ time units. Oporto produces data with uniform distribution and some of the trajectories were more complex than the others.

We implemented our system and conducted the experiments on a Pentium III processor workstation with 256MB of RAM on a 10/100 Mbps LAN. The coding was done on JDK 1.2.4, using JAMA [1] matrix package for matrix operations and OR-Objects [16] package to solve the LP problem.

We initialized different streaming sources with different trajectories but the same initialization parameters using a linear KF model [5]. All the sources had to wait until the sliding window was full. We ran the simulation until one of the sources had read all the $3,000$ records. The tuples received at the server with their timestamps were then used to create the complete trajectory using linear interpolation for both X and Y coordinates. We evaluated the performance of our system based on an *effective resource utilization* (ERU) metric ξ which, is calculated as

$$\xi = \eta * m \tag{14}$$

where m is the fraction of messages exchanged between the source and the server, to the total number of tuples read by the source, where η is the mean fractional error between the actual trajectory and that generated by interpolation. While calculating m we considered the number of tuples forwarded by the source, messages for bandwidth allocation and acknowledgment messages from the server to the source. In all the experiments θ_i=2, the initial sampling interval was five tuples and none of the sources were allowed to skip more than 12 tuples in the adaptive sampling module. We studied the affect of the number of sources, sliding window size W_i and λ_i on the ERU. Results shown in Figures 1, 2, and 3 were obtained using $W_i = 5$ and $\lambda_i = 0.6$.

Figure 1 shows the mean fraction of messages forwarded to the main server against the number of sources. In this figure m is low for a small number of sources, but as the number of sources increases, it rises and stabilizes around 0.12. In all the cases the number of messages is less than or equal to that sent using uniform sampling.

Figure 2 shows that the fractional error using adaptive sampling is always less than that using uniform sampling, except when the number of sources is one. This is because we initialize the experiments with same

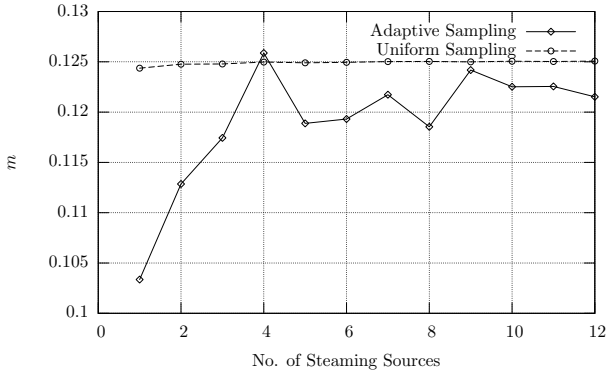


Figure 1: m on varying # of streaming sources

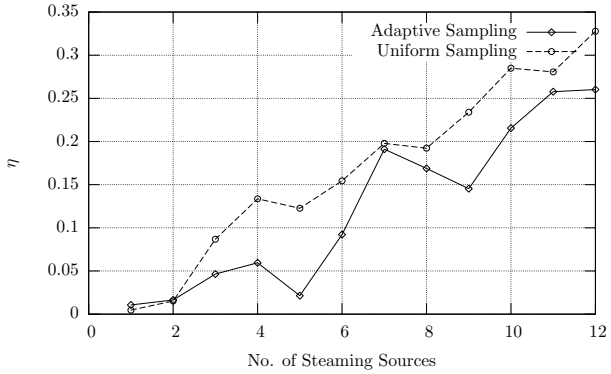


Figure 2: η on varying # of streaming sources

sampling interval for both uniform and adaptive methods. Thus if the number of sources is one, then we cannot beat the uniform sampling method.

Figure 3 shows change in ERU on varying the number of sources. The trend is similar to that in Figure 2. We observe that our approach outperforms the uniform sampling method even when the number of sources is high. There are some unusual results when the number of sources is five and seven. This is because the trajectories for these sources of input data may be unusually simple/complex.

Figure 4 shows the affect of parameter λ_i on the ERU. Resource utilization is high for very low values of λ_i because although the error rate would be low, the number of messages would be very high. We observed lower ERU when λ_i varied around 0.4 and 1.2. This is because at lower values of λ_i the error is low and thus ERU is low, on slightly higher values, although the error is high, the value of m drops down significantly enough to reduce the resource utilization below that of uniform sampling. However at further increasing the value of λ_i, η_i starts to dominate and the ERU starts to increase.

The effect of varying the sliding window size is shown in Figure 5. It is observed that at low values

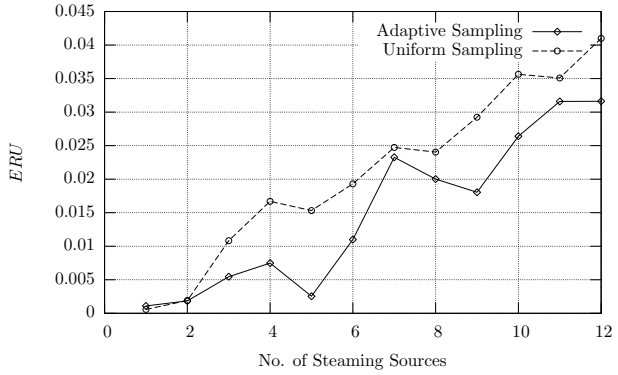


Figure 3: ERU on varying # of streaming sources

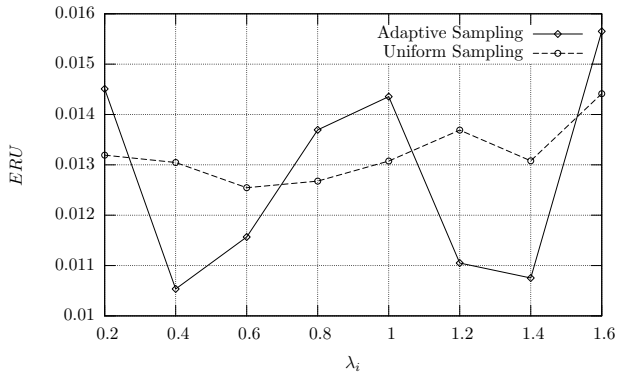


Figure 4: ERU on varying λ_i

of W_i give better ERU. As the window size increases, the ERU approaches constant value.

6 Conclusions and Future Work

In this paper we have proposed an adaptive sampling technique based on a Kalman Filter estimation of error as an alternative to commonly used uniform sampling techniques. We motivated the need for adaptive sampling techniques in a sensor network environment, where network bandwidth is a valuable resource. Adaptive sampling was shown to be desirable not only to conserve resources but also to improve the overall quality of results (minimize the fractional error between the actual and the interpolated results).

We discussed some of the preliminary results in Section 5 to show the effectiveness of our approach. We observed that when we choose the input parameters judiciously, our system can provide performance upgrade as much as three to four times as compared to uniform sampling (Figure 3). We have also shown the effect of different input parameters on the system performance which, suggests that further research needs to be conducted to enable us to choose optimal parameters for the system.

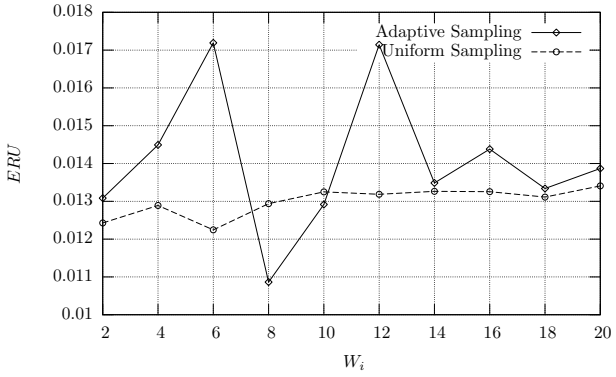


Figure 5: ERU on varying W_i

Our preliminary results are encouraging but further research is indicated in the following directions:

- Extending the current architecture to multi-hop sensor networks.
- Choosing appropriate window size.
- Developing efficient techniques to compute the error over the sliding window. In some cases exponential decay methods might provide better results.
- Developing efficient algorithms to reduce the request/acknowledge message overhead between the server and the sources. (Currently the message overhead is high.)
- Developing algorithms to incorporate adaptive SIRs in the current system.
- Testing the system performance on more real life data sets.

References

[1] R. F. Boisvert, B. Miller, R. Pozo, K. Remington, J. Hicklin, C. Moler, and P. Webb. Jama : A java matrix package.

[2] P. Bonnet, J. E. Gehrke, and P. Seshadri. Towards sensor database systems. In *Second Intl. Conf. on Mobile Data Management*, Hong Kong, January 2001.

[3] B. Hull, K. Jamieson, and H. Balakrishnan. Bandwidth managment in wireless sensor networks. In *Intl. Conf. on Embedded Networked Sensor Systems*, Los Angeles, California, USA, November 2003.

[4] M. Isard and A. Blake. CONDENSATION – conditional density propagation for visual tracking. *International Journal Computer Vision*, 1998.

[5] A. Jain, E. Chang, and Y. F. Wang. Adaptive stream resource management using kalman filters. In *Proceedings of the 2004 ACM SIGMOD Intl. Conf. on Management of Data*, 2004.

[6] L. Jiao, Y. Wu, G. Wu, E. Y. Chang, and Y. F. Wang. The anatomy of a multi-camera security serveillance system. *ACM Multimedia System*, 2004.

[7] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] M. K.Pitt and N. Shephard. Filtering via simulation: auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, June 1999.

[9] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. QUASAR: Quality aware sensing architecture. *ACM SIGMOD Record*, 33(1):26–31, Mar. 2004.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD Intl. Conf. on Management of Data*, pages 491–502. ACM Press, 2003.

[11] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *ACM Intl. Workshop on Wireless Sensor Networks and Applications,WSNA*, Atlanta, Georgia, USA, September 28 2002.

[12] A. D. Marbini and L. E. Sacks. Adaptive sampling mechanisms in sensor networks. In *London Communications Symposium*, London, UK, 2003.

[13] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc.of ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, California, USA, June 2003.

[14] J.-Y. Pan and S. S. amd Christos Faloutsos. Fastcars: Fast, correlation-aware sampling for network data mining. In *GLOBECOM 2002 - IEEE Global Telecommunications Conf.*, pages 2167–2171, Taipei, Taiwan, November 2002.

[15] J.-M. Saglio and J. Moreira. Oporto: A realistic scenario generator for moving objects. *GeoInformatica*, 5(1):71–93, 2001.

[16] D. Systems. OpsResearch: OR-Objects. http://www.opsresearch.com.

[17] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in data streams. In *29th Intl. Conf. on Very Large Data Bases (VLDB)*, pages 309–320, Berlin, Germany, September 2003.

[18] G. Welch and G. Bishop. *Introduction to Kalman Filter*. http://www.cs.unc.edu/∼welch/kalman, 2002.

Predictive Filtering: A Learning-Based Approach to Data Stream Filtering

Vibhore Kumar, Brian F Cooper, Shamkant B Navathe

College of Computing, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332-0280
{vibhore, cooperb, sham}@cc.gatech.edu

Abstract

Recent years have witnessed an increasing interest in filtering of distributed data streams, such as those produced by networked sensors. The focus is to conserve bandwidth and sensor battery power by limiting the number of updates sent from the source while maintaining an acceptable approximation of the value at the sink. We propose a novel technique called Predictive Filtering. We use matching predictors at the source and the sink simultaneously to predict the next update. The update is streamed only when the difference between the actual and the predicted value at the source increases beyond a threshold. Different predictors can be plugged into our framework, and we present a comparison of the effectiveness of various predictors. Through experiments performed on a bee-motion tracking log we demonstrate the effectiveness of our algorithm in limiting the number of updates while maintaining a good approximation of the streamed data at the sink.

1. Introduction

Advances in networking and sensor technology have made it possible to access sensor data as it is gathered, and this in turn has fueled the development of applications that use a continuous stream of data. To manage this data, several data stream management systems have been developed, including STREAM [4], NiagaraCQ [6], TelegraphCQ [7] and Aurora [5]. We consider distributed environments in which remote data sources continuously stream updates to a stream processing installation. These environments incur a significant communication overhead in the presence of rapid update streams. Limiting the number of updates can significantly reduce this overhead.

We present a novel approach to limiting stream updates, called Predictive Filtering. In many scenarios like motion tracking and network monitoring, approximate data values can be tolerated by the stream applications [2]. When the data values do not change randomly, prediction algorithms can be used to approximate the next update when it occurs without actually streaming the data. In our approach the sink requesting data from a stream source specifies to the source a certain precision constraint that needs to be satisfied. Predictors are then deployed both at the source and the sink that adapt to evolving data patterns in the stream. An update is streamed only when the difference between the actual and the predicted value at the source increases beyond the threshold or the precision constraint; otherwise the sink uses the predicted update. In the case of streams with a known update rate, the sink knows when the next update should be predicted; otherwise an update-beacon (described later) is used to signal the occurrence of a new update at the source.

As with all previous prediction techniques our approach is also limited to data streams that show some pattern in the updates because predictions are based on the previous updates and patterns. Our approach is well suited for streaming sensor data, since many sensors track phenomena with an inherent pattern, such as temperature, motion, and so on. We next give an example of one particular application of our approach.

1.1 Example Application: Location Tracking and Collision Prevention

Consider a scenario where a number of fast moving objects are being tracked to maintain location information. Tracking the location of fast moving objects incurs a large amount of communication overhead because of the large number of updates required per unit time to track an object. Our predictive filtering approach takes advantage of the fact that motion does not tend to be random. For example, airplanes follow air routes and bees

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

move in a way that communicates information to other bees. As long as the objects being tracked stay on the predicted course, few updates will have to be streamed. Moreover, our filters can allow us to look beyond the current update and predict with some probability the future positions of the tracked objects. This can aid in applications that may need to react early to certain conditions, such as two airplanes passing too close to each other.

1.2 Related Work

An approach to data stream filtering was suggested by Olston, Jiang and Widom [1], which makes use of adaptive filters for processing continuous queries with precision guarantees. However, the approach makes no attempt to predict the next update and is thus similar to our approach with the predictor always predicting the next update to be equal to the last update. We present results comparing our techniques to their approach in Section 4. Quantitative guarantees regarding the precision of approximate answers is dealt with in [3]. One possible application for monitoring of environmental conditions using wireless sensors is discussed in [8, 9, 10].

1.3 Roadmap of the paper

The rest of this paper is divided into 4 sections; Section 2 gives a brief description of various prediction techniques explored in this paper. Section 3 contains the details to incorporate the prediction techniques into our stream framework. Section 4 contains an evaluation of our approach and its performance when compared to existing data stream filtering algorithms. In Section 5 we conclude by discussing some possible future directions and challenges in the domain.

2. Overview of Algorithm & Prediction Techniques

The choice of a prediction technique is highly dependent on the nature of the data stream under consideration. Using linear extrapolation one can easily approximate a linear data stream that monotonically increases, decreases or remains constant for sufficiently large intervals of time; such a scenario happens when tracking fast moving objects that tend to stay on course. In situations where linear extrapolation is not appropriate because of the rapidly fluctuating or more complex patterns of behavior, enhanced prediction techniques like double-exponential smoothing can be used. In some cases like streaming stock market data, statistically modeling a system for predicting such updates might not be possible. In such scenarios neural network-based time series prediction methods can serve the purpose. However, besides the nature of data stream, the update rate may also affect the choice of the prediction technique. We can tolerate computational delays for streams with slow update rates; but for streams with faster update rates, techniques with less computational overhead have to be used to ensure delivery of all updates. We now briefly describe each of these approaches.

The basic components of our approach are shown in Figure 1. We maintain two predictors; one at the source, and other at the sink, that are exact copies of each other. The predictors contain three components; the 'Predict' component that is responsible for predicting the update based on past updates; 'Learn', which performs the learning in case of an incorrectly predicted update, and finally the 'Update Trigger' that causes periodic generation of an update in case of regular streams or causes the generation of an update on arrival of an update-beacon. The update-beacon is a small message that occurs in lieu of the actual update to signal the sink that an update has occurred. Update-beacons are important for streams with irregular update rates in which the occurrence of next update cannot be determined until it actually occurs. An update-beacon is not required for regular streams, but for irregular streams we must tolerate some communication overhead imposed by update-beacons (which is less than actually propagating the updates) in order to know when updates should be predicted. It may be possible to extend our predictors to predict the update rate as well, although we have not yet

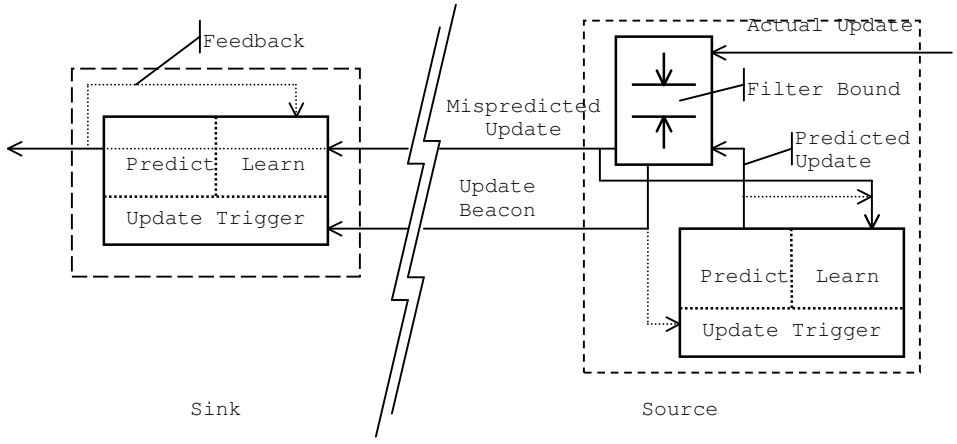


Figure 1. Showing the components in Predictive Filtering

```
/* source side component is invoked for each update */

THRESHOLD threshold;            // user defined
BOOLEAN regular_stream;         // true if stream is periodic

source_update(UPDATE actual_update)
{
    UPDATE predicted_update;
    predicted_update = predict();
    if(difference(actual_update, predicted_update)>threshold){
        stream(actual_update);
        learn(actual_update);
    }
    else{
        if(!regular_stream){
            signal_update_beacon();
        }
    }
}
```

Figure 2. Source side update component

```
/* sink side component is invoked by trigger update with NULL *
 * (either a timer or update-beacon) or by an arriving update   */

sink_update(UPDATE actual_update)
{
    If(actual_update == NULL){    // no update arrived
        actual_update = predict(); // need to predict the update

    }
    else{
        learn(actual_update);
    }
}
```

Figure 3. Sink side update component

examined this possibility. The filter-bound at the source is a user specified parameter that encapsulates the degree of approximation that is tolerable.

The procedures at the source and the sink for handling updates / update-beacons are shown in Figure 2 and Figure 3 respectively. The `learn()` and `predict()` procedures are specific to the type of prediction technique being used. The following sub-sections contain details about these procedures.

2.1 Linear Extrapolation

Linear Extrapolation provides a technique that imposes a very low overhead but performs sufficiently well for predicting a wide range of data streams. Given two updates at time t_n and t_{n+1} the update at time t_{n+2} is given by the following expression:

$$u(t_{n+2}) = \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} t_{n+2} + \{u(t_n) - \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} t_n\}$$

Thus, the next update is predicted to be on a straight line connecting the previous two updates. When the stream is not changing rapidly we can predict more than the next update i.e. the d^{th} update into the future is calculated using the following expression:

$$u(t_{n+d}) = m \times t_{n+d} + c$$

$$where \;\; m = \frac{u(t_{n+1}) - u(t_n)}{t_{n+1} - t_n} \;\; and \;\; c = \{u(t_n) - m \times t_n\}$$

For linear extrapolation, learning consists only of tracking the previous two updates.

2.2 Double Exponential Smoothing

Double exponential smoothing-based prediction (DESP) [11] models a given time series using a simple linear regression equation where the y-intercept c and slope m are varying slowly over time. An unequal weighting is placed on these parameters that decays exponentially through time so newer observations get a higher weighting than older ones. The degree of exponential decay is determined by the parameter $\alpha \in [0:1)$. The method makes use of two smoothing statistics:

$$S(u(t_n)) = \alpha\, u(t_n) + (1-\alpha) S(u(t_{n-1}))$$

$$S'(u(t_n)) = \alpha\, S(u(t_n)) + (1-\alpha) S'(u(t_{n-1}))$$

Using these smoothing statistics c and m can be estimated as c´ and m´ by applying the following equations

$$m'(t_n) = \frac{\alpha}{1-\alpha}(S(u(t_n)) - S'(u(t_n)))$$

$$c'(t_n) = 2S(u(t_n)) - S'(u(t_n)) - t_n m'(t_n)$$

Given these estimates and with some algebraic manipulation the next update is predicted time d into the future with

$$u(t_{n+d}) = (2 + \frac{\alpha d}{1-\alpha}) S(u(t_n)) - (1 + \frac{\alpha d}{1-\alpha}) S'(u(t_n))$$

In this case, the predictor learns by refining c and m over time based on updates.

2.3 Artificial Neural Network based Predictors

Another popular technique used for prediction is Neural networks [12, 13, 14]. Compared to the previous two techniques, they tend to be more adaptable and flexible, since they can effectively model complex non-linear mappings and a broad class of problems due to their non-parametric nature. Their topology and weights are adaptable; therefore they are able to learn, which makes neural networks well suited for applications like prediction, system identification, and classification in many problem domains. One of the drawbacks of neural networks is that they need a sufficiently large data set to

```
/* source side component is invoked for each update */

THRESHOLD threshold;          // user defined
BOOLEAN regular_stream;       // true if stream is periodic

source_update(UPDATE actual_update)
{
     UPDATE predicted_update;
     predicted_update = predict();
     if(difference(actual_update, predicted_update)>threshold){
          stream(actual_update);
     }
     else{
          if(!regular_stream){
               signal_update_beacon();
          }
     }
}
```

Figure 4. Modified source side update component

```
/* sink side component is invoked by trigger update with NULL *
 * (e.g. a timer or update-beacon) or by an arriving update       */

sink_update(UPDATE actual_update)
{
     If(actual_update == NULL){     // no update arrived
          actual_update = predict(); // need to predict the update

     }
     else{
          learn(actual_update);
          update_source_predictor();
     }
}
```

Figure 5. Modified sink side update component

train the network, but this is what makes them even more suitable for stream-based applications that present enormous amounts of data. Another advantage of using neural networks is that they can be used to predict categorical data streams, which may have certain elements that cannot be mathematically approximated. However, the inherent mathematical complexity involved in training neural networks and then in the prediction may limit the usefulness of this approach.

Since we do not want to overload the stream data-source with computations for predicting the next update, we use a modified algorithm in this scenario. The learning component in case of neural network based predictors is present only at the sink and the predictor at the source is periodically updated to ensure similar predictions as the sink. The modified source and sink procedures are shown in Figure 4 and Figure 5 respectively. Further modifications to the algorithm include updating the sink predictor with batched updates after a threshold number of mispredictions. This allows us to limit the number of

```
/* E-Code Equivalent of source side linear predictor */

{
     int predicted_val;
     predicted_val = filter_data.m * filter_data.x + filter_data.c;
     if((predicted_val – input.val)>=filter_data.threshold ||
       (input.val – predicted_val)>=filter_data.threshold)){
          //update the filter_data
          filter_data.m = (filter_data.last_val – input.val);
          filter_data.c = input.val – filter_data.m * filter_data.x;
          filter_data.x = filter_data.x + 1;
          filter_data.last_val = input.val;
          return 1;
     }
     filter_data.x = filter_data.x + 1;
     filter_data.last_val = predicted_val;
     return 0;
}
```

Figure 6. E-Code representation of source side linear predictor

source predictor updates and produces better on-line training of the neural network. We have so far experimented with simple three-layer feed-forward neural networks and made use of the error back-propagation method to train the network; units with sigmoid function were used to construct the network. We are gathering results for neural-network based predictors as part of our ongoing work.

3. Implementation Details

In this section we deal with the issues concerning the deployment of predictive filters. We have implemented a distributed stream management framework using the ECho publish-subscribe middleware [18, 19] developed at Georgia Tech. The predictive-filtering algorithm discussed in this paper was implemented as a part of our stream management framework. The ECho middleware supports channels that facilitate the flow of data between the source and the sink. One of the important features of the ECho-Channel framework is its ability to dynamically compile and deploy filters written in E-Code, a highly portable subset of C, at remote sites to process data at the source. We have used this ability of the framework to enable deployment of predictors at the source. More details about ECho and the E-Code Language can be found in [18].

The E-Code equivalent of a source-side predictor using linear extrapolation is shown in Figure 6. The variables *input* and *filter_data* are implicitly available to the function. The *input* variable contains the update. The *filter_data* variable contains configuration and state information, including the slope, y-intercept, last update and the update iterator. A return value of 1 causes the update to be transmitted to the sink while a 0 results in non-transmission of the update.

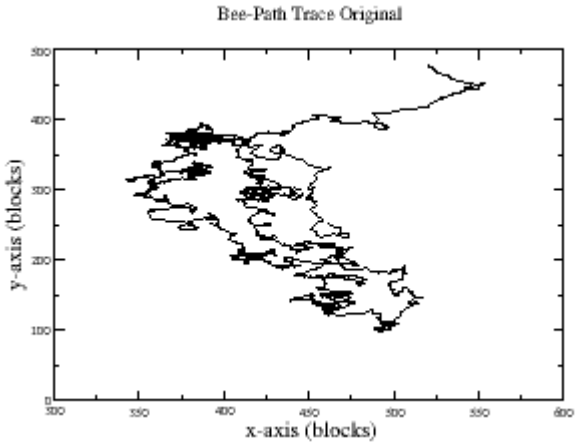


Figure 7. The original path followed by the Bee.

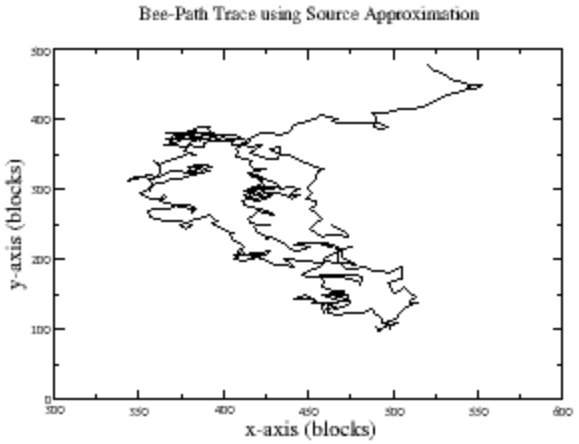


Figure 8. Bee-Path Trace using Source Approximation

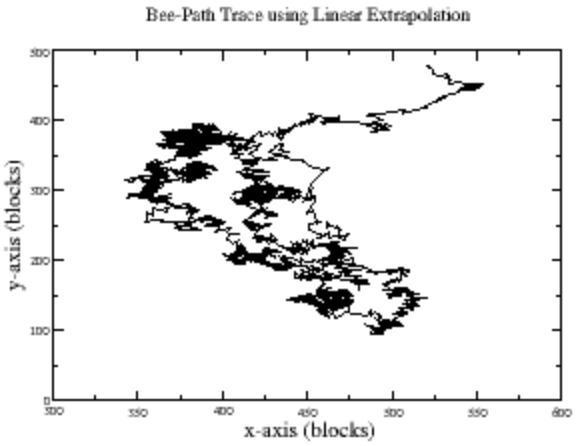


Figure 9. Bee-Path Trace using Linear Extrapolation

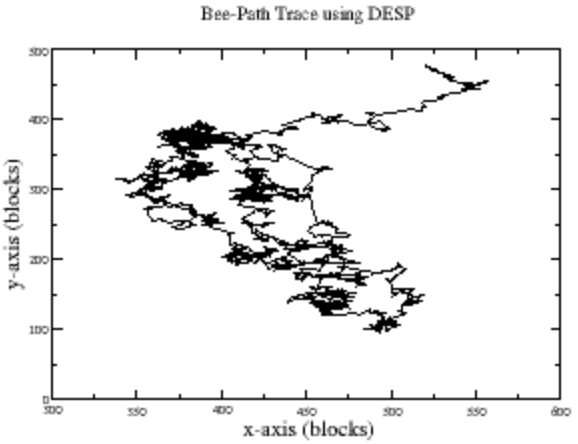


Figure 10. Bee-Path Trace using Double Exponential Smoothing

Table 1. Comparison of various predictive filtering techniques for constant Filter-Bound = 5

Filtering Technique	Number of Updates Propagated	RMS Error
No Filtering	7846	-
Source Approximation[1]	542	3.98
Linear Extrapolation	672	2.62
Double Exponential Smoothing	454	2.07

Another advantage of using ECho channels is that the sink can remotely access the *filter_data* configuration information available to the source-predictor. This facility helps in remote maintenance of this structure and proves helpful in implementation of neural network based predictors in which the source predictor needs to be updated by the sink.

4. Experimental Results

We evaluated the performance of our technique and its applicability by designing a motion tracking system that records a log of various fast-moving objects. The temporal data feed was visual data collected by the BioTracking group at Georgia Tech [20]. The group video recorded activity of bees around a beehive for 10 days, 12 hours per day; and then processed the video using an image-processing algorithm to track the individual bees. We chose this data stream because the high rate of change in a bee's trajectory allows us to examine our techniques for predicting very complex data streams.

The original motion log of a single bee is shown in Figure 7, which depicts 7846 updates from the bee-path

Table 2. Comparison of various predictive filtering techniques for RMS-Error ~ 4

Filtering Technique	Number of Updates Propagated	Filter-Bound
No Filtering	7846	-
Source Approximation [1]	542	5.00
Linear Extrapolation	464	6.30
Double Exponential Smoothing	359	7.80

trace. In our experiment, we limited the number of updates for tracking the bee by using various prediction techniques discussed above. Figure 8 shows the path traced by the bee using the source approximation technique discussed in [1]. Note that the path is very sparse and misses some details. Figure 9 shows the results obtained by using a linear extrapolation predictor. The figure shows more of the zig-zag nature of the original curve as the filter tries to predict and approximate the correct position of the bee. The results of using a double exponential smoothing based predictor are shown in Figure 10 and smooth edges and even more detail are shown in the figure.

Table 1 shows the actual number of updates propagated from the source to approximate the position log for the various prediction techniques when filter-bound is kept constant. It also contains the root-mean-squared (RMS) difference (error) between the updates predicted by or received at the source and the actual updates. The RMS measures the quality of the approximated data; lower error is better. The number of updates required for linear extrapolation is more than that required for source approximation but the corresponding RMS error is considerably lower. However, the double exponential smoothing technique is the best both in terms of the number of propagated updates and the RMS error. Table 2 shows the actual number of updates required by each technique to approximately deliver the same quality in terms of RMS error at the sink. The table clearly shows the advantage of using prediction-based methods for filtering the updates. It maybe noted that better filtering techniques allow for relaxed filter-bound to achieve the same quality of sink updates.

Experiments with neural-network based predictors are being conducted as part of our ongoing work.

5 Conclusion and Future Work

We have presented a novel approach to limiting the number of updates in streaming data environments by predicting values rather than streaming them. We have described the basic algorithm, which can be used in conjunction with a number of prediction techniques. Our initial experiments suggest that our approach can produce high quality data at the sink while effectively limiting the number of updates that must be sent. Our approach is applicable to a large number of streaming data applications typically present with sensor networks that deal with regular data: e.g. network monitoring data, traffic data and stock market data, and so on. We are currently exploring the possibility of predicting categorical data streams using neural network based predictors. We understand that the techniques are limited to data with numeric values; if the data is from a domain with discrete strings or categories as values, a modification of the proposed techniques will be needed. This is left as part of our future work.

References

[1] C Olston, J Jiang, J Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In proceedings of the ACM SIGMOD International Conference on Management of Data, 2003.

[2] R Min, M Bharadwaj, S Cho, A Sinha, E Shih, A Wang and A Chandrakasan. Low-power wireless sensor networks. In proceedings of the Fourteenth International Conference on VLSI Design, India, January 2001.

[3] H Yu and A Vahdat. Efficient numerical error bounding for replicated network services. In proceedings of the Twenty Sixth International Conference on Very Large Databases, Cairo, Egypt, September 2000.

[4] S Babu, J Widom (2001) Continuous Queries over Data Streams. SIGMOD Record 30(3):109-120

[5] D Carney, U Cetintemel, M Cherniack, C Convey, S Lee, G Seidman, M Stonebraker, N Tatbul, S Zdonik. Monitoring Streams: A new class of data management applications. In proceesings of the twenty seventh International Conference on Very Large Databases, Hong Kong, August 2002.

[6] J Chen, D DeWitt, F Tian, Y Wang. NiagaraCQ: A scalable continuous query system for internet databases. In proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, May 2000.

[7] S Chandrasekaran, M Franklin. Streaming Queries over Streaming Data. In proceesings of the twenty seventh International Conference on Very Large Databases, Hong Kong, August 2002.

[8] J M Kahn, R H Katz, K S J Pister. Next century challenges: Mobile Networking for "smart dust". In the proceedings of the ACM/IEEE International Conference

on Mobile Computing and Network Monitoring (MobiComm-99), Seattle, Washington, August 1999.

[9] S Madden, M J Franklin. Fjording the stream: An Architecture for queries over streaming sensor data. In the proceedings of the 18th International Conference on Data Engineering, San Jose, California, February 2002.

[10] G J Pottie, W J Kaiser. Wireless integrated network sensors. Communications of the ACM, 43(5):551-558, May 2000

[11] Joseph J LaViola Jr. Double exponential smoothing: an alternative to Kalman filter-based predictive tracking. Proceedings of the workshop on Virtual environments, Zurich, 2003.

[12] C. Lee Giles, Steve Lawrence, A. C. Tsoi. Noisy Time Series Prediction using a Recurrent Neural Network and Grammatical Inference. Machine Learning Journal, volume 44, 2001.

[13] Amalia Foka. Time Series Prediction Using Evolving Polynomial Neural Networks. MS Dissertation.

[14] S. Bengio, F. Fessant, and D. Collobert. A Connectionist System for Medium-Term Horizon Time Series Prediction. In International Workshop on Applications of Neural Networks to Telecommunications, Stockholm, Sweden, 1995.

[15] G. Cybenko. Approximation by superposition of sigmoidal functions. Mathematics of Control, Signal and Systems, 2:303--314, 1989.

[16] F. Fessant, S. Bengio, and D. Collobert. On the Prediction of Solar Activity Using Different Neural Network Models. Annales Geophysicae, 1995.

[17] F. Fessant, S. Bengio, and D. Collobert. Use of Modular Architectures for Time Series Prediction. Neural Processing Letters, 1995.

[18] Greg Eisenhauer, Fabian Bustamente and Karsten Schwan. A Middleware Toolkit for Client-Initiated Service Specialization. Proceedings of the PODC Middleware Symposium - July 18-20, 2000.

[19] Greg Eisenhauer. The ECho Event Delivery System. Technical Report GIT-CC-99-08, College of Computing, Georgia Institute of Technology, Atlanta.

[20] http://borg.cc.gatech.edu/biotracking/

Confidence-based Data Management for Personal Area Sensor Networks *

Nesime Tatbul[§] Mark Buller[†] Reed Hoyt[†] Steve Mullen[†] Stan Zdonik[§]

[§] Brown University
{*tatbul, sbz*}*@cs.brown.edu*

[†] U.S. Army Research Institute of Environmental Medicine
{*mark.j.buller, reed.hoyt, stephen.mullen*}*@us.army.mil*

Abstract

The military is working on embedding sensors in a "smart uniform" that will monitor key biological parameters to determine the physiological status of a soldier. The soldier's status can only be determined accurately by combining the readings from several sensors using sophisticated physiological models. Unfortunately, the physical environment and the low-bandwidth, push-based personal-area network (PAN) introduce uncertainty in the inputs to the models. Thus the model must produce a confidence level as well as a physiological status value. This paper explores how confidence levels can be used to influence data management decisions. In particular, we look at power-efficient ways to keep the confidence above a given threshold. We also contrast push-based broadcast schedules with other schedules that are made possible by two-way communication.

1 Introduction

Data management has traditionally been reserved for large complex software environments in which huge amounts of data must be processed with limited resources. Modern database management systems (DBMS) that run on large back-office servers are the most well-known embodiment of this kind of technology. Researchers have recently realized that similar technologies are needed in smaller environments in which resource limitations are also an issue [9, 4]. In sensor-based applications, bandwidth and battery power are typically the scarce resources.

This paper looks at a real sensor-based application in which results are computed along with a confidence value. The data management game that we play here is to set transmission parameters (statically or dynamically) in order to achieve the highest confidence only when the application requires it. Our techniques use strategies that are informed by the confidence models to conserve bandwidth and power. We discuss these ideas and some possible approaches in terms of a military physiologic sensing application. Our main contribution is in the way that confidences can be used in this particular application.

The warfighter's workplace has unique occupational challenges: from mission demands, the environment, and combat injuries. Modern dismounted soldiers commonly engage in intense, mentally and physically demanding 3-10 day missions, often in rugged terrain or complex urban settings. Warriors carry heavy loads and are often food and sleep-restricted. Environmental conditions can vary widely in terms of ambient temperature, humidity, wind speed, barometric pressure, and the like. In non-war mode the military can suffer over 120 heat casualties a year [1]. Under or over hydration can decrement physical and cognitive performance, and increase the risk of heat injury, hyponatremia, or death [14, 15, 16]. Added to the harsh environment is the possibility of receiving a wound. Once a warfighter has become a casualty, it is critical that treatment is received quickly during the "golden hour", which is the short period of time when proper medical treatment can mean the difference between life and death. It has been suggested that 20% of these deaths could be prevented with rapid intervention [13]. Therefore, wearable physiological and medical status monitoring can play an important role in: sustaining physical and mental performance, reducing

* This work has been supported in part by the Military Operational Medicine Research Program, the Combat Casualty Care Research Program, and the Telemedicine and Advanced Technologies Research Center, US Army Medical Research and Material Command, Ft. Detrick, MD, 21704-5014; and by the NSF, under the grants IIS-0086057 and IIS-0325838. The opinions or assertions contained herein are private views of the authors and are not be construed as official or as reflecting the views of the US Army or Department of Defense. Citations of commercial organizations and trade names in this report do not constitute an official Department of the Army endorsement or approval of the products or services of these organizations.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

the likelihood of non-battle injuries such as heat stroke, and provide remote notification and medical status of a casualty.

In this paper, we first describe the Warfighter Physiologic Status Monitoring (WPSM) application in detail in Section 2, where we also present an example scenario and show how the sensor network behaves under this scenario. We discuss potential data management techniques that would improve the existing network in Section 3. We present some preliminary simulation results in support of these discussions. Section 4 summarizes related work in the area. We conclude the paper by discussing future directions in Section 5.

2 The WPSM Application

2.1 The Sensor System

The Medical Research and Material Command (MRMC) under its Warfighter Physiologic Status Monitoring - Initial Capability (WPSM-IC) program is developing what is essentially a wellness monitor for each soldier. This system is comprised of a medical hub which hosts a personal area network of physiologic and medical sensors and a number of algorithms. The algorithms estimate the state of the warfighter in the following areas: *Thermal*, *Hydration*, *Cognitive*, *Life Signs*, and *Wound Detection*. Each area has four potential states that are coded by color. *Green* represents normal-no action is required; *Yellow* means requires attention; *Red* calls for immediate action; and *Blue* indicates a system fault. For each area's state, the hub also estimates a confidence level. Confidence refers to the accuracy level of the state estimated by a model.

The states for each medical and physiologic area are based upon input to the state algorithms from a number of sensors distributed around a warfighter's body, uniform and equipment, as well as outputs from other algorithms resident in the medical hub. Figure 1 shows a schematic of the current WPSM-IC sensor system and the physical placement of sensor equipment on a warfighter. The ingestible thermometer pill is network-enabled, and measures the temperature of the stomach and intestines, which is usually a good indication of body core temperature. The fluid intake monitor measures the amount of fluid consumed through a bladder-style canteen. The life sign detection sensor (LSDS) is an integrated system with multiple parameters and algorithms including heart rate, respiration rate, body orientation, actigraphy [1], and skin temperature. The LSDS also has an integrated ballistic impact detection device which provides an alert when on-body acoustic signals are detected that indicate the probability that a ballistic projectile has impacted the warfighter. The sleep performance watch treats sleep as a consumable quantity, measures it, and uses an algorithm to equate this to apparent cognitive readiness. The soldier also carries a GPS and other technologies which report his geographic location.

[1] Actigraphy is a measure of activity patterns [7].

Figure 1: WPSM-IC Sensor System

The sensors are connected to the medical hub by a proprietary wireless RF network [12]. The network was developed with a number of key requirements unique to a military operational environment. The network needed to be very low power, to allow miniature physiologic sensors to run for weeks without the need of battery recharge or replacement, and also have an ability to reject cross talk and interference from similar networks borne by other soldiers when congregated in close proximity to each other. In addition, the network had to provide a low profile signature to avoid detection.

The current network uses a detuned (low detectability) 40MHz radio frequency (RF) carrier. Digital data are transmitted from sensors to the medical hub utilizing a pseudo random push transmission scheme. Sensors are factory set with an identification number (ID) and random number table seed. Sensors are supplied operating in a deep sleep mode and are activated through an infrared (IR) port, by a medical hub. Activation associates a particular sensor with a particular hub. The sensor in a series of initial transmissions sends its transmission schedule (based upon its ID and random number seed) and clock information to the hub. Knowing this information, the hub is able to keep itself in a sleep mode, powering up fully only when it knows to expect a transmission from an associated sensor. This reduces power consumption in the hub ($\sim$0.1% duty cycle) and also guards, to some degree, against cross talk from other sensors. This "push-only" scheme has the benefits of allowing sensors to only carry transmission circuitry which is activated on a known schedule, rather than both a transmitter and receiver. In a "polled" scheme, a sensor would need to constantly power the receiver circuitry to listen for data polls, and hence consume more power. Sensors in the current network sample every 15 seconds and transmit data at 2400 baud on average every 15 seconds. The transmission interval can vary from 3 seconds to 27 seconds according to the pseudo random schedule with each transmission time interval having an equal probability of occurrence. Each sensor message is 240 bits long.

Model	Skin Temp.	Heart Rate	Actigraphy	Geo-Location	Resp. Rate	Pill	# Sensors
TSkin	√						1
Threshold	√	√					2
Model1			√				1
Model2			√	√			2
Model3		√	√	√	√		4
TCore						√	1

Table 1: Models for estimating thermal state

2.2 Example Scenario: Estimating the Thermal State

In this paper, we focus more closely on the warfighter thermal state, and the sensors and models which allow thermal state and its confidence to be determined. In what follows, we describe an example scenario for estimating the thermal state of a soldier.

The best and most confident method to assess thermal state is direct measurement of core body temperature by using the network-enabled ingestible pill. When core body temperature is greater than $39.5°$C, there is a high probability that the warfighter is in thermal strain. However, this method is impractical for continual use. Thus, these devices are reserved for use during high thermal stress missions, while encapsulation in nuclear, biological, and chemical protective suits, and/or if use is indicated by other algorithms or medics.

When a core temperature pill is not being used, WPSM-IC plans to use variants of two basic types of models to provide an estimate of thermal state. The simplest model is the Threshold Model [2] that takes inputs from two sensors measuring skin temperature and heart rate. Under very low and high skin temperatures, the confidence in states produced by this model is higher than otherwise. For mid-values of temperature, knowing heart rate values improves confidence. The second model is a first principles model similar to the USARIEM Scenario Model [6], that takes metabolic rate, environmental conditions, clothing configurations and biometric data as inputs to estimate core body temperature. Metabolic rate and the environmental conditions are key drivers of this model. From the current system, metabolic rate can be derived independently from heart rate, respiration rate, actigraphy, and geo-location readings in multiple ways with different confidence levels. Based on these, Table 1 summarizes six alternative models to estimate thermal state together with the sensors they are using. TSkin Model is a simplified version of the Threshold Model, using only the skin temperature sensor. The Threshold Model additionally uses the heart rate sensor. Models 1-3 represent variants of the first principles model where metabolic rate is derived using different sets of sensors: Model1 uses just actigraphy; Model2 uses both actigraphy and geo-location; Model3 uses actigraphy, geo-location, heart rate and respiration rate. Finally, TCore Model uses the core temperature pill. Each alternative model has complex algorithms that map sensor values to physiologic states with certain confidence levels. The details of these algorithms are outside the scope of this paper.

Our thermal state estimation problem consists of three major dimensions that determine the confidence levels:

1. **Model:** The first factor is the model, and hence the set of sensors, that participate in the state computation. Input from a greater number of sensors usually increases the confidence in the state. This is not true when the core temperature pill is used. However, the core temperature pill is unique in that it is a consumable sensor, with a costly logistics and resupply train.

2. **Latency:** The second factor is the latency of sensor messages. As readings get older, their relevance and usefulness to the models and state algorithms decay. Thus, a latency decay function or "shelf-life" is defined for each sensor. This function maps latency values measured in seconds to decay coefficients. For our example scenario, all sensors are simply assumed to have the following exponential decay function[2]:

$$2^{-(\lceil latency/15 \rceil - 1)}, \text{ where } latency > 0$$

   For example, a heart rate reading of age 20 seconds has a decay coefficient of 0.5, i.e., a state computation that uses this heart rate value would have its confidence level degraded by 0.5. When multiple sensors are involved in a model computation, we simply use their average latency to compute the decay coefficient. If sensors had different latency decay functions, then we would take an average of their individual decay coefficients.

3. **State:** Finally, the third determinant of confidence is the output state. For our thermal state estimation problem, the Green state can be determined with higher certainty than the Yellow and Red states.

Next, we present confidence assignments on two of the dimensions, Model and State. The latency dimension is based on the decay function provided above.

As mentioned earlier, physiologic models are also affected by the physical environment. In Table 2, we illustrate a detailed work environment scenario. The first two columns of this table show nine different environment-activity combinations. Work environment conditions are

[2]In general, it is more realistic to choose different decay functions for different sensors. For example, heart rate readings would certainly age faster than ambient temperature readings.

		TSkin			Threshold			Model1			Model2			Model3			TCore
Env.	Work	G	Y	R	G	Y	R	G	Y	R	G	Y	R	G	Y	R	G/Y/R
cool	low	80	76	72	90	85.5	81	95	90.25	85.5	95	90.25	85.5	95	90.25	85.5	100
warm	low	80	76	72	90	85.5	81	95	90.25	85.5	95	90.25	85.5	95	90.25	85.5	100
hot	low	60	57	54	80	76	72	95	90.25	85.5	95	90.25	85.5	95	90.25	85.5	100
cool	med	70	66.5	63	90	85.5	81	95	90.25	85.5	95	90.25	85.5	95	90.25	85.5	100
warm	med	50	47.5	45	70	66.5	63	80	76	72	90	85.5	81	95	90.25	85.5	100
hot	med	40	38	36	60	57	54	70	66.5	63	80	76	72	90	85.5	81	100
cool	high	40	38	36	60	57	54	90	85.5	81	95	90.25	85.5	95	90.25	85.5	100
warm	high	20	19	18	40	38	36	60	57	54	70	66.5	63	80	76	72	100
hot	high	5	4.75	4.5	20	19	18	50	47.5	45	60	57	54	75	71.25	67.5	100

Table 2: Work Environment models to estimate thermal state and their confidence levels

measured independently from the soldier (e.g. through a weather station) and they are external to the soldier's personal area sensor network. However, they directly affect the confidence achieved by the models. For each environment-activity combination, confidence levels for six alternative models are shown. Note that these values are representative values. Each model can estimate the Green (G) state with the highest confidence. If a Yellow (Y) is computed, this confidence degrades by 0.95; if a Red (R) is computed, it degrades by 0.90. TCore Model is an exception as its confidence for all states is perfect due to its being a direct measure of thermal state. Note that as the environment moves from cool to hot, and as activity moves from low to high, more types of sensors may be needed to maintain a high confidence about the soldier's thermal state.

The application requires different confidence levels depending on soldier's state. Table 3 shows the required thresholds for our example. If a Green state is reported, its confidence has to be at least 50. If a Yellow state is observed, a confidence value of at least 70 is required. Finally, if soldier's state is reported to be Red, a confidence value of at least 80 has to be provided. In other words, the application requires higher confidence for more important events. The goal is to operate the sensor network in such a way that it delivers state estimations with sufficient confidence levels.

State	Confidence Threshold
Green	≥ 50
Yellow	≥ 70
Red	≥ 80

Table 3: Required confidence thresholds for each state

2.3 The Push-Only Transmission Scheme

We simulated the existing push-only sensor network on CSIM [11]. We ran the alternative models of the example scenario through the simulator, using one model and one environment-work pair at a time. We assumed that the soldier is in the Green state. We make the following important observation: Models requiring more sensors do not always achieve better confidence levels. Models periodically compute states based on what has most recently been received from the participating sensors. When more sensors are present in the network, the frequency of packet collisions and message drops increases. When the most recent measurement from a sensor is missing, state computation at the hub has to rely on a stale earlier reading from that sensor. As mentioned before, stale data degrades the confidence level associated with each instance of model output. Figure 2 shows how each model behaves under three of the environment-activity conditions. Model3, using four sensors to estimate metabolic rate, achieves better confidence than other models (except TCore) in the (warm, high) and the (hot, high) cases which represent relatively high intensity conditions. To generalize this notion, delivering high-confidence for different detection goals (i.e., thermal stress, wound detection, etc) demand different models.

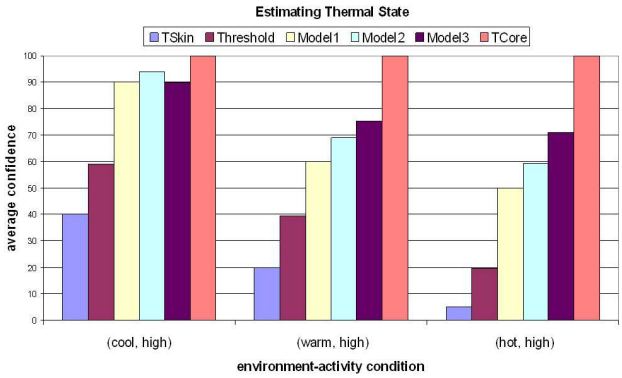


Figure 2: Simulation of the push-only scheme

3 Confidence-based Data Management

In the WPSM context, data management largely concerns the scheduling of data transmissions. Frequent transmission can in principle improve latency, but over zealous transmission can waste power and increase the odds of a collision (i.e., lost data). In what follows, we discuss techniques for optimizing this tradeoff. We use confidence modeling as the primary way to inform these decisions.

Model	Average Confidence	% Drop
Model 1	64.92	0
Model 2	72.73	1.98
Model 3	77.75	5.79
All Models	79.22	5.71

Table 4: Model redundancy simulation results

3.1 Exploiting Redundancy

Physiologic states can be estimated with higher certainty by allowing redundancy at several levels.

Model Redundancy. All alternative models to estimate a particular state can run concurrently. As we have demonstrated, various factors like sensor values and latency decay may cause one model to achieve higher confidence than another. By running the models simultaneously, one can obtain multiple state estimations at different levels of certainty and the one with the highest confidence can be picked. Table 4 shows preliminary results from a model redundancy simulation for a changing work environment scenario. We again assume that the soldier is in the Green state. Models 1-3 are run both separately and all together. The work environment is initially set to (cool, high) and then gradually changed to (warm, high) and (hot, high). When all models are redundantly run together, the average confidence is the highest. Models 1 and 2 have fewer drops due to fewer sensors sharing the channels. *Model 3* and *All Models* use four sensors and they both experience higher percent message drop due to collisions. Note that *All Models* loses around the same percent of messages as *Model 3* alone, but achieves higher average confidence.

Data Redundancy. Sensor readings can be transmitted multiple times. A sensor message not only contains the most recent reading, but also the previous reading as well. This type of redundancy is useful when the model to be computed not only requires the most recent sensor value, but a valid sensor reading every certain time period. This increases the probability that a reading will get through.

Obviously, allowing redundancy has drawbacks in terms of resource consumption. Running all models at the same time increases network traffic and message loss. Similarly, repeating readings in multiple messages increases message lengths, thus consuming bandwidth and expending additional battery power. Therefore, the degree of redundancy has to be adjusted based on a tradeoff between desired level of confidence, variability of the conditions affecting confidence, and resource consumption.

3.2 Adjusting Sampling Rates

In the current deployable network, sensors come with factory-set transmission schemes. Thus, their sampling and transmission periods are not adjustable. However, we believe that confidence levels and network lifetime could be considerably improved by dynamically adjusting these sensor parameters to match the requirements of the physiological models. Thus, we foresee a need to incorporate two-way communication into future sensor designs. Of course, we must be able to show that the extra power needed to run the receiver is worth it.

In general, sensors reporting with high frequency feed low-latency values into the models, but messages are more likely to get dropped due to collisions. In the extreme case, high data rates can translate into high latency as well as extensive energy consumption. On the other hand, low-frequency transmissions seldom get dropped and use power economically, but they may not refresh the models as often as needed. Each sensor's sampling rate should be adjusted between these two extremes based on model requirements.

One thing to consider is the sharing between running models. There are five different areas where state estimation is needed. Each area may also run multiple models concurrently. Each model requires readings from a certain subset of the sensors. Sensors could be ranked based on how many models they are feeding. Also, importance of a state could be considered. For example, Wound Detection may be more important than Cognitive State. Sensors involved in Wound Detection should have higher rank. Sensors of high rank should have shorter sampling and reporting periods.

A second consideration is the latency decay functions of the sensors. A cumulative latency decay function could be defined based on functions of all sensors involved in a model computation. This function would indicate how often that model has to be refreshed to preserve its confidence level. As mentioned before, some sensors can have stricter latency requirements than others. For example, heart rate readings age faster than temperature readings. This implies that the heart rate sensor must update more often, illustrating the notion that refresh periods are application dependent.

3.3 Bi-directional Data Communication

The sensor network used in the described application is designed to be push-only, where data flows in a single direction, from sensors to the hub. Sensors do not have any receivers, but only transmitters. The rationale behind this kind of a setup is threefold. First, it uses less power since no sensor wastes battery by listening to the network. Second, message loss is small since collisions are expected to occur less frequently. Last but not least, push-only sensors are much cheaper to build. However, this design limits many potential optimizations that could be performed at the receiver hub.

The receiver hub is the only point in the network that has a complete view of all the sensors and all the physiological models with their confidence requirements. As such, it can make the best judgement about how to deliver high confidence states in an efficient way. However, in a push-only scheme, it has no control over sensor transmissions. The hub must be able to "pull" from the sensors as needed.

With a two-way communication model, we can accumu-

late minimal sensor readings in order to populate the lower-confidence models. Typically, the amount of data and the latency requirements are lower for low confidence results. In this situation, if we get an alert for a thermal stress event with a low confidence, we can then contact the sensors to collect more data in order to feed the higher confidence models. Thus, we only spend bandwidth and power when it is needed. In other words, in the normal operating case, it is best to run lean at the expense of confidence. When an important but low confidence event is observed, we expend more resources to confirm or deny it. We now illustrate this point on our work environment example presented in Section 2.2. As shown in Table 3, our application has different confidence requirements depending on the soldier's state. These requirements can be met in multiple ways using alternative models. For example, if the soldier is in the Green state and under the (hot, high) condition, Models 1-3 and TCore Model can deliver enough confidence (≥ 50). Among these models, Model1 is the most desirable one. First of all, Model1 uses only one sensor. Thus, network bandwidth does not have to be shared with other sensors. The network lifetime with one sensor would be much longer as the energy consumption at the hub is proportional to the the number of sensors it is communicating with. Finally, the actigraphy sensor used by Model1 is a much cheaper alternative than using the core temperature pill. If we apply this heuristic of "using as few sensors as possible" to all condition and state combinations in Table 2, we end up with model preferences shown in Table 5.

To show the performance benefit of this heuristic, we considered a scenario where the soldier is in a (hot, medium) environment and is initially in the Green state. Then his state gradually changes to Yellow and Red. Model3 delivers enough confidence for all of these states. Therefore, we ran one simulation where only Model3 is used. In a second simulation, we started out with Model1 and changed to Model2 only when soldier entered Yellow state, when Model1 can not deliver enough confidence. Similarly, when the soldier's state changed to Red, we switched from Model2 to Model3 so that confidence is above the required threshold. This second run simulates the behavior of a hub pulling from sensors as necessary. Initially, it only pulls from the actigraphy sensor; then the geo-Location sensor is added; and finally, heart rates and respiration rates are pulled. We further assumed that the main determinant of network lifetime is the battery at the hub which is about 1800 mAHrs. Additionally, we assume that each sensor that is turned on has a current draw of 50mA; i.e, if this sensor is left on for an hour, it will consume 50mAHrs of the total 1800mAHrs battery. Then we compared these two simulations in terms of network lifetime. The first simulation runs out of hub battery in 9 hours whereas the second one can survive more than 14 hours. This simple scenario clearly demonstrates how a pull-based model could conserve energy based on model and situation-specific confidence requirements.

In a way, bi-directional communication enables switching between alternative estimation models dynamically. As such, it is a much more efficient alternative to the redundancy approach proposed in Section 3.1.

Env.	Work	G	Y	R
cool	low	TSkin/Model1	TSkin/Model1	Model1
warm	low	TSkin/Model1	TSkin/Model1	Model1
hot	low	TSkin/Model1	Model1	Model1
cool	med	TSkin/Model1	Model1	Model1
warm	med	TSkin/Model1	Model1	Model2
hot	**med**	**Model1**	**Model2**	**Model3**
cool	high	Model1	Model1	Model1
warm	high	Model1	Model3	TCore
hot	high	Model1	Model3	TCore

Table 5: Model preferences based on the number of sensors

Two-way communication is also more flexible than the sampling rate adjustment approach discussed in Section 3.2. Sensor transmission rates can effectively be adjusted by changing the pull frequency at the hub.

4 Related Work

There is a growing body of research on sensor network data management. TinyDB [18] and Cougar [4] are two example query processing systems for multi-hop sensor networks. These systems emphasize in-network processing of declarative queries to reduce data communications and battery usage. TinyDB especially focuses on acquisitional aspects of query processing like where, when and how often data should be collected from the sensors [9]. Sensor sampling rates are adjusted based on event and lifetime specifications of queries. Cougar uses sensor update and query occurrence probabilities for view selection and location on top of a carefully constructed aggregation tree [4]. Scheduling techniques to overcome collisions in the sensor network are also explored in this project. These systems are designed to serve monitoring applications that span a larger or difficult to reach geographical area than the personal area case, where multi-hop sensor communication is a necessity (e.g., habitat monitoring).

More relevant to our problem are quality-driven approaches. As an example, TiNA exploits temporal coherency tolerance specifications of users in in-network processing to trade off between result quality and energy conservation [17]. Sensor readings are reported only if they differ from an earlier value by a certain amount. Another example is the QUASAR project [8], which also exploits applications' tolerance to imprecision to minimize resource consumption. As a more closely related work to ours, a model-driven approach for data acquisition in sensor networks has been recently developed by Deshpande et al [5]. A probabilistic model of the sensor network data is created based on a history of readings from sensors and correlations between them. Queries can be approximately answered based on this model. If confidence requirements can not be met by the model alone, then the sensors in the network need to be queried. The model is also refined as

more readings are received. In the application that we consider, multiple complex models exist to estimate physiological states of a soldier. Each model uses a different set of sensors. These models and their confidence levels are well-defined. Rather than building and refining the models, we concentrate on efficient data acquisition from sensors to estimate states with acceptable confidence using alternative models.

There is some related work on data management for personal area sensor networks as well. For example, a recent work proposes a query processing system for healthcare bio-sensor networks [3]. Patient heart rates are monitored using electrocardiogram (ECG) and accelerometer sensors. Multiple ECG sensors have to be worn for a complete measurement of the electrical activity of the patient's body. Furthermore, if the patient moves, ECG signals may be corrupted. Therefore, readings from an accelerometer sensor have to be correlated with ECG readings for a more reliable result. This application has similar sensor network uncertainty concerns as ours. However, the focus of this work is on query processing at the base station. We believe our confidence-based approach could be used at the data acquisition phases of this system to improve query results. In the same domain, CodeBlue is a wireless communications infrastructure for medical care applications [10]. It is based on publish/subscribe data delivery where sensors worn by patients publish streams of vital signs and geographic locations to which PDAs or PCs accessed by medical personnel can subscribe. Secure and ad hoc communication, prioritization of critical data, and effective allocation of emergency personnel in case of mass casualty events are major emphases of this project.

Finally, wireless sensor networks are also a subject of recent research in the networking community. Of particular relevance to our work are MAC (Medium Access Control) protocols that determine when and how the network nodes coordinate to share a broadcast channel [19]. Collision avoidance is a major concern in these protocols. S-MAC is one such protocol where sensor nodes periodically sleep to reduce energy consumption by avoiding idle listening [20]. While such protocols make the underlying network more reliable in power-efficient ways, they are unaware of the application-specific requirements, like confidence levels in our WPSM-IC application.

5 Future Directions

In this paper, we presented a challenging sensor network application which can highly benefit from various data management strategies as evidenced by our initial simulation results. We are currently working on making these strategies operational on the real network. In the future, we are planning to extend this work in several directions. A potential research direction involves treating sensor readings as continuous waveforms with integrity constraints. If sensor values could be noisy or erroneous, earlier values could verify or deny confidence of the latest value. We could also decide when to pull from sensors based on what values have recently been received. If recent heart rate readings suggest that the heart rate could not have gone beyond normal threshold since the last reading, then we do not need to receive a new heart rate report.

WPSM-IC is currently concerned with dismounted warriors and the management of their personal area networks. The goal at this point is to create a summary of the soldier's physiological state at the hub. In the future, this state information would be disseminated to other battlefield units. This might include mobile medics who are deployed in the theater of operation or to advanced field hospitals that are prepared to deal with both prevention of potential casualties as well as management of known casualties of various kinds. The information that is uploaded beyond the individual soldier would be used for some form of remote triage.

The remote triage problem, of course, comes with its own technical challenges. Similar reports from more than one co-located soldier might be an indication of a particular kind of attack. Physiological status reports from many soldiers can be used to prioritize treatment. In these cases, the medic might find that the reported confidence level is not high-enough to warrant the deployment of an ambulance. Instead, the medic may contact the soldier's hub to amplify the confidence to some given level. This might require a great expenditure of resource, but in an emergency, the investment is likely worth it.

References

[1] Army Medical Surveillance Activity. `http://amsa.army.mil/`.

[2] L. Berglund, M. Yokota, and M. Kolka. Non-Invasive Physiological Hyperthermia Warning System. Technical report, USARIEM, 2004 (in process).

[3] C.-M. Chen, H. Agrawal, M. Cochinwala, and D. Rosenbluth. Stream Query Processing for Healthcare Bio-sensor Applications. In *IEEE ICDE Conference*, April 2004.

[4] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. The Cougar Project: A Work-In-Progress Report. *Sigmod Record*, 32(4), December 2003.

[5] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB Conference*, Toronto, Canada, September 2004.

[6] A. P. Gagge, A. P. Fobelets, and L. G. Berglund. A standard predictive index of human response to the thermal environment. *ASHRAE Transactions*, 92(2B):709–731, 1986.

[7] R. Hoyt, M. Buller, J. DeLaney, D. Stultz, K. Warren, M. Hamlet, D. Schantz, W. Matthew, W. Tharion, P. Smith, and B. Smith. Warfighter Physiologic Status Monitoring (WPSM): Energy Balance and Thermal Status During a 10-Day Cold Weather US Marine Corps Infantry Officer Course Field Exercise. Technical Report T-02/02, DTIC Number A396133, USARIEM, October 2001.

[8] I. Lazaridis, Q. Han, X. Yu, S. Mehrotra, N. Venkatasubramanian, D. V. Kalashnikov, and W. Yang. QUASAR: Quality-Aware Sensing Architecture. *Sigmod Record*, 33(1), March 2004.

[9] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *ACM SIGMOD Conference*, San Diego, CA, June 2003.

[10] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2004.

[11] Mesquite Software, Inc. CSIM18 Simulation Engine. `http://www.mesquite.com/`.

[12] Mini Mitter, Inc. Physiological and Behavioral Monitoring for Humans and Animals. `http://www.minimitter.com/`.

[13] R. F. Bellamy. The Causes of Death in Conventional Land Warfare: Implication for Combat Casualty Care Research. *Military Medicine*, 149:55–62, 1984.

[14] S. J. Montain and E. F. Coyle. Fluid ingestion during exercise increases skin blood flow independent of increases in blood volume. *Journal of Applied Physiology*, 73(3):903–910, 1992.

[15] S. J. Montain and E. F. Coyle. Influence of graded dehydration on hyperthermia and cardiovascular drift during exercise. *Journal of Applied Physiology*, 73(4):1340–1350, 1992.

[16] S. J. Montain and M. N. Sawka and W. A. Latzka and C. R. Valeri. Thermal and cardiovascular strain from hypohydration: Influence of exercise intensity. *International Journal of Sports Medicine*, 19(2):87–91, 1998.

[17] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation. In *3rd ACM MobiDE Workshop*, September 2003.

[18] TinyDB: A Declarative Database for Sensor Networks. `http://telegraph.cs.berkeley.edu/tinydb/`.

[19] W. Ye and J. Heidemann. Medium Access Control in Wireless Sensor Networks. In C. S. Raghavendra, K. M. Sivalingam, and T. Znati, editors, *Wireless Sensor Networks*. Kluwer Academic Publishers, 2004.

[20] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM 2002)*, New York, NY, September 2002.

Approximately Uniform Random Sampling in Sensor Networks

Boulat A. Bash John W. Byers Jeffrey Considine

Computer Science Department
Boston University
{boulat, byers, jconsidi}@cs.bu.edu

Abstract

Recent work in sensor databases has focused extensively on distributed query problems, notably distributed computation of aggregates. Existing methods for computing aggregates broadcast queries to all sensors and use in-network aggregation of responses to minimize messaging costs. In this work, we focus on uniform random sampling across nodes, which can serve both as an alternative building block for aggregation and as an integral component of many other useful randomized algorithms. Prior to our work, the best existing proposals for uniform random sampling of sensors involve contacting all nodes in the network. We propose a practical method which is only approximately uniform, but contacts a number of sensors proportional to the diameter of the network instead of its size. The approximation achieved is tunably close to exact uniform sampling, and only relies on well-known existing primitives, namely geographic routing, distributed computation of Voronoi regions and von Neumann's rejection method. Ultimately, our sampling algorithm has the same worst-case asymptotic cost as routing a point-to-point message, and thus it is asymptotically optimal among request/reply-based sampling methods. We provide experimental results demonstrating the effectiveness of our algorithm on both synthetic and real sensor topologies.

1 Introduction

In the emerging research area of sensor databases, a central challenge is to develop cost-effective methods to extract answers to queries about conditions inside the sensor network. One typical sensor database scenario involves sensor elements that are prone to failure, are highly resource-constrained, and must communicate across a lossy network. Sensor networks comprised of small battery-powered motes are a representative instantiation of this scenario [7]. In such an environment, *aggregation queries* are particularly effective, as they are robust to node and link failures, can be resilient to incorrect or outlying responses, and are amenable to the use of in-network processing to minimize messaging cost. For these queries, approximate answers typically suffice, especially in light of the very high cost of ensuring 100% reliability in communications in sensor networks. Recent work has focused on computation of aggregates using a request/reply model in which a query is broadcast to a region of interest, individual sensors make best-effort replies, and responses are aggregated in-network en route to the origin of the query [3, 11, 20].

In this paper, we argue that there is a rich and relatively under-explored set of classic statistical methods that have not yet been extensively studied in the domain of sensor databases. In particular, we propose a more careful study of random sampling methods, which have long been used in other domains to approximately compute aggregates such as MEDIAN, AVG, and MODE [2, 12, 13]. Random sampling is a particularly good fit for approximate aggregation queries in the sensor network domain in light of the potentially modest messaging cost. While we view random sampling as especially useful in the context of data management and data aggregation problems, we also note that it is an integral component of other useful randomized algorithms that are potentially applicable to sensor networks, including randomized routing [18].

In the context of sensor networks, a natural abstraction is *spatial sampling*, i.e. sampling from geographical locations within the network uniformly at random. On a 2-D network with bounded spatial extent, such an objective can easily be realized by picking an (x, y) coordinate from within the space at random and using geographical routing to route to the node closest

The authors were supported in part by NSF grants ANI-9986397, ANI-0093296, ANI-0205294, and EIA-0202067.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

to that point. While this is desirable for many applications, such as computing spatial averages [6], many other applications and database queries prefer to ignore geometry and instead wish to *sample uniformly from the set of nodes*. Examples include querying average sensor battery life, counting the number of nodes that are currently capable of executing a given sensing task, determining the 95th quantile of sensor CPU utilization, or estimating the number of sensors that will fail within the next day. Our focus is to develop practical algorithms for uniformly sampling from a set of sensor nodes with low messaging cost.

Since it is well-known that nodes in a sensor network often have highly irregular placements, spatial sampling will produce non-uniform samples of the nodes [5]. Our work relies on spatial sampling as a starting point, but uses practical methods for smoothing, or regularizing, the non-uniform samples to produce approximately uniform node samples. The key idea is to have each sensor node compute and maintain the area of its Voronoi cell. A uniform node sample is then realized by sending a sequence of spatial samples until one is "accepted". A targeted node in the network "accepts" by responding to a given spatial sample with an appropriate probability normalized by its Voronoi cell size, otherwise it "rejects". The specifics of this normalization depend on global statistics on the number of nodes in the network and on an appropriate k-quantile of Voronoi cell sizes across the network. We argue that these statistics can be updated infrequently and consistently. Ultimately, this application of von Neumann's rejection method [19] results in approximately uniform node samples.

As sketched above, our algorithm for generating a random sample has a messaging cost that is typically bounded by the messaging cost of a small constant number of spatial samples in the expectation. This cost is low since the messaging cost of computing a spatial sample is akin to routing a point-to-point message using a geographic routing method such as GPSR [8]. In the worst case, such a message traverses the diameter of the network. In contrast, the best existing methods for node sampling, which can compute an exactly uniform sample, necessitate contacting all nodes in the network [13]. We note that the additional infrequent global update costs incurred by our algorithm can be amortized by the potentially vast number of samples that can be taken between updates.

The remainder of the paper is organized as follows: Section 2 formalizes the uniform sampling problem and the limitations of existing methods. We summarize the building blocks of our proposed method in Section 3. Our rejection-based sensor sampling algorithm is presented in Section 4. Then in Section 5 we describe the practical implementation issues, and Section 6 concludes with the broader implications and applications of our work.

2 Sampling: Problems and Methods

We now formally define our sampling problems.

Definition 1 (Uniform random sampling) *An algorithm samples uniformly at random from a set of reachable sensors S if and only if it outputs a sensor ID $s \in S$ with probability $\frac{1}{|S|}$.*

Uniform random sampling is simple if the set of sensor IDs is known in advance and sensors neither fail nor move. However, it is much more challenging in the realistic case where the set of IDs may not be known and the set of reachable sensors dynamically changes over time. For these reasons, we will be content with the following close approximation to uniform sampling.

Definition 2 ((ϵ, δ)-sampling) *An algorithm performs (ϵ, δ)-sampling of a reachable set S if and only if it returns a sample $s \in S$ such that no element of S is returned with probability greater than $\frac{1+\epsilon}{|S|}$ and at least $(1-\delta)|S|$ elements are output with probability at least $\frac{1}{|S|}$.*

By this definition, our goal is to sample from almost all sensors nearly uniformly with tunable parameters ϵ and δ. Our definition allows us to under-sample a small fraction δ of the nodes.

In a sensor network scenario, we typically wish to sample from a set of pairs $\langle k, v \rangle$ where k identifies a particular sensor and is unique within the set, and v is some value associated with the sensor. This value might be a measurement by the sensor, such as the local temperature, or an internal statistic such as the remaining battery life. As motivated earlier, sampling in sensor networks is more challenging since neither a full list of sensors nor direct communication with them is available.

Prior to this work, the following two methods for near-uniform sampling were proposed in the context of sensor and other overlay networks.

Min-wise sampling: In [13], the use of min-wise samples [1] was proposed for sampling a sensor network uniformly at random. Given a hash function h on sensor IDs, they returned the value associated with the ID s such that $h(s)$ is minimal (i.e. $\forall_{s' \in S}(h(s) \leq h(s'))$). Each sensor would then propagate the value associated with the smallest observed $h(s')$. With careful control of the transmissions, this scheme can be implemented with each node in the sensor network sending a constant number of messages, for a total of $\Theta(|S|)$ transmissions. However, since the entire network is involved, this is an expensive operation.

Random walks: Another natural method for sampling is the use of random walks. In the sensor network domain, one could generate a random sample by propagating a request message along a randomly chosen k-hop path starting from the query sink, and sampling

the kth sensor reached. Unfortunately, this procedure would both need to use a large value of k, and would need to compensate for the fact that the method is biased toward drawing samples from near the center of the spatial region where sensors are located, as we demonstrate in Section 5.1.

Our methods follow a rather different line. Like the random walk method, we ultimately seek out a single sensor, but our choice of the route to the sensor avoids many of the dependencies and complications of the random walk approach.

3 Prerequisites

Instead of choosing a path at random, we choose a location in the sensor coordinate space at random and route a probe to its closest sensor using geographic routing techniques. When we partition the coordinate space into regions of ownership by mapping the nearest neighbor regions (Voronoi cells) to sensors, we note that these regions are irregularly sized in most instances. Thus, this naive *spatial sampling* method is very likely to generate a biased sample. Therefore, our last key step is to use von Neumann's rejection method to normalize the samples. We now briefly summarize these three prerequisite ideas.

Geographic routing: If every node in a network is aware of its own coordinates (e.g. via GPS), then it is possible to route to a particular position using entirely local decisions. Most of these local routing decisions can be made in a greedy fashion, simply choosing the neighboring node which has the closest coordinates to the destination. This greedy routing fails when there is an obstruction, or "void", which must be circumnavigated to reach the destination. GPSR [8] provides an elegant solution to this problem with just two states. The default state of GPSR is greedy routing, while the other state follows the perimeters of voids until greedy routing can resume. When a packet reaches its target point, another round of perimeter routing is run to visit each of the immediately surrounding sensors so that it can find the sensor nearest to the target point. For typical topologies in 2-D, geographic routing takes $\Theta(\sqrt{|S|})$ steps.

Voronoi diagrams: Once routing to an arbitrary point is possible, we must also quantify the size of the region of points that are closest to a particular sensor s. Formally, the set of points closer to sensor s than any other sensor is called the *Voronoi cell* of s [4]. In the planar case which we consider, the Voronoi cell of s is a convex polygon containing s, where each edge of this polygon lies on a perpendicular bisector between s and another sensor. The exact boundaries of this Voronoi cell are easily determined exactly by locating all of the sensors in the immediate vicinity of s. The areas of these Voronoi cells have been used previously to weight sensor readings for spatial aggregates [6] and they are easily computable, but it is well known that these areas vary widely when the sensors are placed randomly [16]. This variation leads to a bias in spatial sampling – each sensor is chosen with probability in proportion to $A(s)$, the area of its Voronoi cell. For convenience, we assume the areas are normalized so that they sum to one, and thus $A(s)$ can also be interpreted as the probability a randomly chosen point is closest to s.

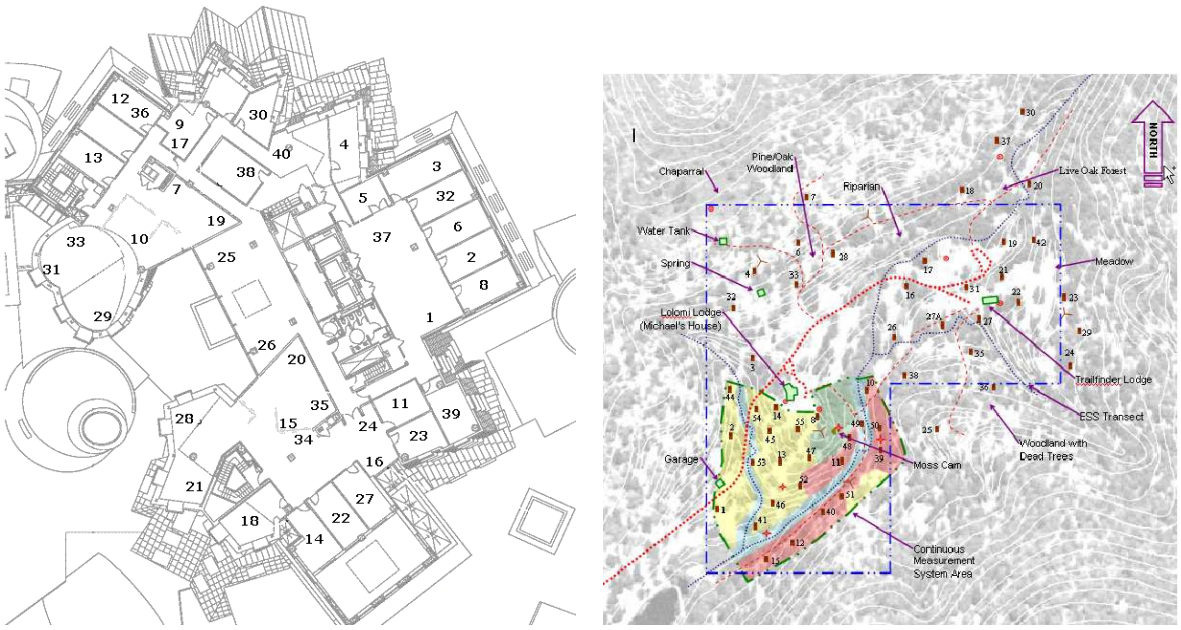
von Neumann's rejection method: Much of the early work on random sampling focused on sampling complex distributions, assuming the ability to sample simpler distributions. A well known example of this is von Neumann's rejection method [10, 19]. Suppose we wish to sample from a distribution with probability density function f (i.e. an event t has probability $f(t)$). If we can sample from a distribution with probability density function g, then we can sample from f as follows. First generate a sample t using g, but only accept and return sample t with probability $\frac{f(t)}{cg(t)}$, where c is a positive constant. If t is not accepted, it is rejected and the process repeats for a new sample t. Assuming that c is chosen so that $\frac{f(t)}{cg(t)} \leq 1$, then the probability of picking a particular event t on the first attempt is $g(t) \cdot \frac{f(t)}{cg(t)} = \frac{1}{c} f(t)$. It then follows that after c expected samples from g, we have one sample from f.

4 Rejection-based Sensor Sampling

We now describe our method to combine ideas of spatial sampling with von Neumann's rejection method to flatten out an irregular probability distribution into a nearly uniform one. For our application, the desired density function is uniform, i.e. $f(t) = \frac{1}{|S|}$, and the distribution which we can sample from, $g(t)$, is the distribution of Voronoi cell areas. One weakness in von Neumann's method for *exactly* reproducing a distribution f is that the constant c must be chosen so that for all events t, $\frac{f(t)}{g(t)} \leq c$. In our application, if there exists a very small Voronoi cell, then c, and hence the expected messaging cost, can be very large. Since we cannot rule out this possibility, we content ourselves for now with generating approximately uniform samples. Later, in Section 5.2, we consider strategies to boost sampling probabilities for the smallest cells to significantly reduce residual sampling bias. We employ the following basic algorithm.

Algorithm 1 (Rejection-based Sampling)

1 The random sampler picks a random location in the sensor field and routes a message to the sensor s closest to this point, using geographic routing and pre-computation of Voronoi cells.



(a) MIT sensor testbed. Reproduced with permission from [17]

(b) James Reserve sensor network. Reproduced with permission from [5]

Figure 1: Maps of real sensor deployments used in our experiments.

2 With probability $\frac{\min(A(s),\tau)}{A(s)}$*, s accepts and reports its value, where* τ *is a threshold to be defined shortly.*

3 Otherwise, s rejects and the random sampler repeats Steps 1-3. The random sampler also returns to Step 1 if it times out waiting for a response.

Intuitively, τ can be thought of as a threshold on Voronoi cell areas, in which we think of any Voronoi cell of area at least τ as *large* and any area less than τ as *small*. By our procedure, all large cells will be selected equiprobably, but small cells will be selected with smaller probability, in proportion to their area. To ensure that Algorithm 1 results in (ϵ, δ) sampling, we must guarantee that the fraction of small cells (sampled non-uniformly) is less than δ, and that the bias introduced by under-sampling small cells results in at most $(1+\epsilon)$-oversampling of large cells. In practice, we set τ to be the area of the cell that is the k-quantile, where $k = \min\left(\delta, \frac{\epsilon}{1+\epsilon}\right)$, and prove the following main result.

Theorem 1 *Running Algorithm 1 with* $k = \min\left(\delta, \frac{\epsilon}{1+\epsilon}\right)$ *and setting* τ *to be the cell area that is the* k*-quantile results in* (ϵ, δ)*-sensor sampling.*

Proof: By our problem definition, it suffices to show that the method ensures that no element of S is sampled with probability greater than $\frac{1+\epsilon}{|S|}$ and at least $(1-\delta)|S|$ elements are sampled with probability at least $\frac{1}{|S|}$. First, we show that all large cells, i.e. cells with area at least τ, are sampled in a given iteration of the sampling algorithm with probability at least $\frac{1}{|S|}$. The probability that a given sensor s is sampled in a particular probe is $p_s = \min(A(s), \tau)$, and thus the probability that a particular probe is successful is $\sum_s p_s \leq |S|\tau$. Now let E_ℓ denote the event that the algorithm ultimately samples from a large cell ℓ.

$$\Pr[E_\ell] = \frac{p_\ell}{\sum_s p_s} = \frac{\tau}{\sum_s p_s} \geq \frac{1}{|S|}.$$

Now since large cells are at least a $(1-\delta)$ fraction of all cells by the setting of $k \leq \delta$, we have that at least $(1-\delta)|S|$ elements are sampled with probability at least $\frac{1}{|S|}$.

Next we show that no element is sampled with probability greater than $\frac{1+\epsilon}{|S|}$. By construction, large cells are sampled with highest probability, so we restrict attention to those cells. Starting from the same probability bound as before:

$$\begin{aligned}
\Pr[E_\ell] = \frac{p_\ell}{\sum_s p_s} &= \frac{\tau}{\sum_{s|A(s)<\tau} p_s + \sum_{s|A(s)\geq\tau} p_s} \\
&= \frac{\tau}{\sum_{s|A(s)<\tau} p_s + \sum_{s|A(s)\geq\tau} \tau} \\
&\leq \frac{\tau}{\sum_{s|A(s)\geq\tau} \tau} \\
&\leq \frac{\tau}{(1-k)|S|\tau}
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{\tau}{(1-\frac{\epsilon}{1+\epsilon})|S|\tau} \\
&\leq \frac{1}{(\frac{1+\epsilon}{1+\epsilon}-\frac{\epsilon}{1+\epsilon})|S|} \\
&= \frac{1}{(\frac{1}{1+\epsilon})|S|} \\
&= \frac{1+\epsilon}{|S|}.
\end{aligned}
$$

Thus the theorem follows. ■

Relating this result back to von Neumann's method, this corresponds to a situation in which $c = \frac{1}{\tau|S|}$. As with the rejection method, the probability that a particular sensor s is picked and accepted on the first attempt is $A(s)\frac{\min(A(s),\tau)}{A(s)} = \min(A(s),\tau)$. It remains to select an appropriate threshold τ for our algorithm.

4.1 Threshold Management

Given user-specified values of ϵ and δ, the threshold τ should be set to the k-quantile of the Voronoi cell areas, where $k = \min\left(\delta, \frac{\epsilon}{1+\epsilon}\right)$ as discussed earlier. The k-quantile can be computed during an initial preprocessing step using recent techniques developed in the sensor database community. In particular, work such as [3, 11] shows how to efficiently count the number of sensors matching some criteria (e.g. with a cell area below a specified threshold) and deriving other simple statistics such as the average cell area. We note that while these values need to be updated to account for dynamic changes within the sensor network, they need not be exact, as bounds on the values suffice for our methods. Therefore, only infrequent updating of these global statistics is needed to maintain consistent and approximately correct values. Updating these statistics can easily be performed either by piggybacking them on the random probes or on various control and maintenance messages. Either way, once these statistics are available, the sampler recomputes τ, and sends it with each probe. Since the sampler's value of τ is included in the query, each sensor deciding to accept or reject a probe acts consistently.

5 Practical Implementation Issues

We now discuss the details of a practical implementation of Algorithm 1. We begin in Section 5.1 presenting experimental results using the basic implementation outlined in Section 4, and then discuss various refinements to improve the uniformity of sampling in Section 5.2.

5.1 Experiments

We experimentally validated our proposed sampling algorithm using three topologies: two from real sensor deployments and one synthetic topology with 2^{15} sensors placed uniformly at random on a unit square. The first real network, illustrated in Figure 1(a), is a testbed deployed at MIT [17]. These sensors were heuristically placed according to expected quality as a vantage point, and proximity to available power outlets. The second real deployment, illustrated in Figure 1(b), is a sensor network for micro-climate monitoring at the James Reserve [5]. These sensors are more concentrated in the lower left, where there is thick foliage.

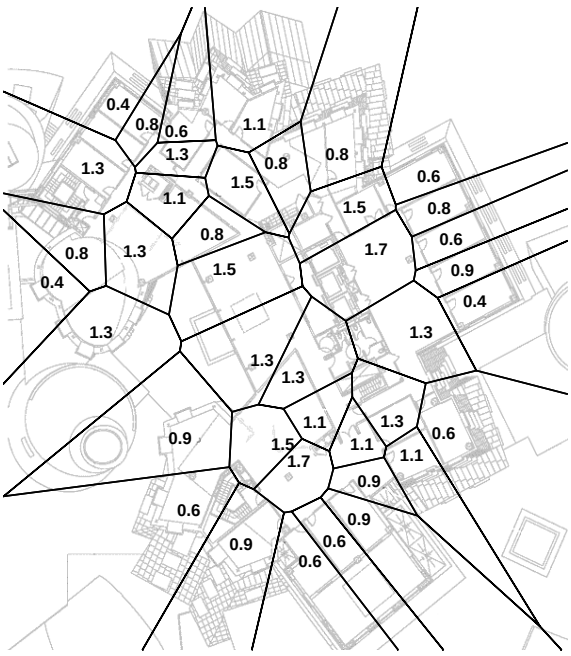
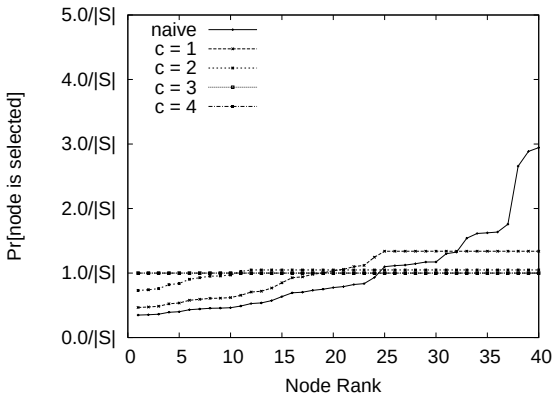


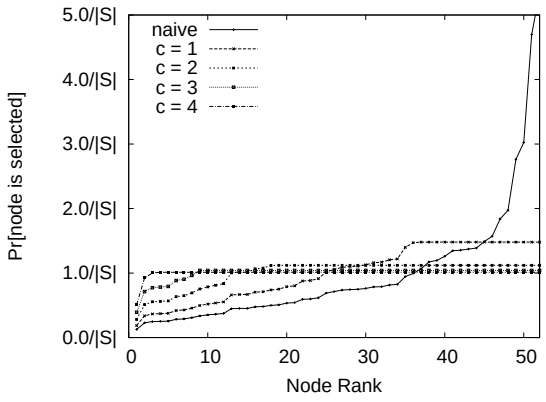
Figure 2: Sample distribution using long random walks along adjacent Voronoi cells. Each sensor's cell is labeled with its probability relative to the mean. For example, a sensor labeled 1.3 is picked with probability $1.3/|S|$.

The objective of these experiments was to demonstrate that we can cheaply obtain a close approximation to uniform sampling. Thus, besides examining ϵ and δ at for various choices of τ, we also examine the expected value of the random variable $\mathbf{Y}$, which is the number of probes sent before a sample is returned. The actual energy costs of our method depend heavily upon the geographic routing protocol in use. Since testing the performance of various geographical routing protocols is beyond the scope of this work, we do not implement geographic routing in our simulation.

First, we confirm our intuition that random walks are unsuitable for near-uniform random sampling. We consider the following random process. Starting at any sensor in the network, a query repeatedly considers the sensors with adjacent Voronoi cells and moves to one chosen uniformly at random. After a sufficient num-



(a) MIT sensor testbed



(b) James Reserve sensor network

Figure 3: Resulting distributions for real testbeds. Nodes are in increasing order of Voronoi cell area.

c	ϵ	δ	$\mathbf{E}[\mathbf{Y}]$
naive	1.9	0.6	1.00
1	0.34	0.45	1.34
2	0.047	0.25	2.09
3	0	0	3.00
4	0	0	4.00
5	0	0	5.00

(a) MIT sensor testbed

c	ϵ	δ	$\mathbf{E}[\mathbf{Y}]$
naive	4.3	0.69	1.00
1	0.48	0.46	1.48
2	0.12	0.23	2.24
3	0.041	0.15	3.12
4	0.012	0.038	4.05
5	0.0072	0.019	5.04

(b) James Reserve sensor network

c	ϵ	δ	$\mathbf{E}[\mathbf{Y}]$
naive	3.8	0.57	1.00
1	0.27	0.41	1.27
2	0.051	0.15	2.10
3	0.017	0.06	3.05
4	0.0079	0.029	4.03
5	0.0042	0.017	5.02

(c) 2^{15} randomly placed points

Table 1: Summary of experimental results

ber of steps to converge on the stationary distribution, the query outputs its current location. Figure 2 shows the Voronoi diagram of the MIT sensor testbed and the relative sampling probabilities of each sensor. As expected, the sensors most likely to be chosen are in the middle of the network, and the sensors least likely to be chosen are on the edges of the network. Sufficiently long random walks on this topology can achieve $(0.71, 0.52)$-sampling. This is better than naive spatial sampling, which would achieve $(1.90, 0.60)$-sampling on the same topology, but our rejection-based methods will give much better results.

Figure 3 shows the results of Algorithm 1 on the real topologies assuming that there are no faults and each sensor knows the area of its own Voronoi cell. The areas of both networks are the areas of their minimum bounding boxes. The threshold τ was set to $\frac{1}{c|S|}$ for $c = 1, 2, 3, 4, 5$, and the naive spatial sampling method is included as a baseline. As c increases and τ decreases, the distribution becomes more uniform and improvements in both ϵ and δ are clearly visible.

Tables 1(a) and 1(b) summarize the parameters of the resulting sampling distributions, along with the expected number of probes for each sample. With the MIT sensor testbed, setting $c = 3$ (equiv. $\tau = \frac{1}{3|S|}$) results in uniform sampling – this is because there are no sensors with less than a third of the average cell area in their Voronoi cell. With the James Reserve network, one sensor has a cell area of slightly more than one tenth of the average, so $c \geq 10$ is necessary for uniform sampling. However, this is the only sensor which is under-sampled for $c \geq 5$.

For comparison, Table 1(c) summarizes the corresponding results for a synthetically generated topology of 2^{15} randomly placed points on a unit square. The smallest Voronoi cell in this topology was slightly smaller than $\frac{1}{98|S|}$, so if exact sampling is desired, an average of $c \geq 99$ probes per sample are needed. However, just setting $c = 5$ achieves $(0.0042, 0.017)$-sampling.

Figure 4 shows the cell size distributions of our test topologies where the impact of human choices on sensor placement is present. First, humans are prone to favor interesting or easily accessible points, resulting in sensors being clustered together, each with below-average area. This is evident in Figure 4: the two real sensor networks have a larger fraction of sensors with below-average Voronoi cell areas than a randomly generated topology. At the same time, humans are unlikely to choose very poor placements where many sensors are extremely close together. Figure 4 also hints at this point, as the smallest Voronoi cells in syntheti-

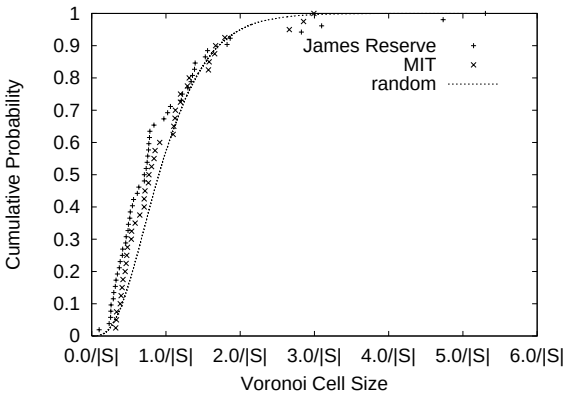


Figure 4: Cell size distributions for random and real testbeds

cally generated networks are significantly smaller than the ones in real topologies.

5.2 Algorithmic Modifications

We now consider a variety of heuristics for improving our baseline algorithm by reducing the impact of small Voronoi cells on the (ϵ, δ)-approximation.

Sleeping: Perhaps the simplest method for handling sensors with very small Voronoi cells is for some of these sensors to sleep. Sleeping sensors are deactivated, and sampling from them is thus rendered impossible. Putting one small cell to sleep will increase the size of adjacent cells (which are also likely to be small), so it is not necessary to put all small cells to sleep to remove their impact. We note that this approach is similar in spirit to some routing schemes which use sleep for power management, particularly in crowded areas [21]. Because the sensed values from the sleeping nodes are unavailable, this approach may not be appropriate for some applications.

Pointers: Another method for increasing the sampling probability of small cells is for larger cells to keep pointers to nearby small cells and forward some rejected probes to those small cells. That is, whenever a large cell would reject a probe, it may instead redirect the probe to a nearby small cell. The probability of forwarding a probe can be negotiated between the cells based on their respective sizes. Essentially, a large cell would donate part of its "unused" area to its small neighbor.

Virtual coordinates: Instead of using real-world geographic coordinates to map points to sensors, we can use virtual coordinates [14, 15], modified to include either a repulsive force between close sensors, or a hard lower bound on the inter-sensor distances. Virtual coordinate spaces also allow the boundaries of the sensor network to be pre-defined, instead of explored via periodic probing [5].

6 Future Work and Conclusions

Uniform random sampling is a standard and useful primitive underlying many algorithmic and statistical methods. Our work focused on the unique constraints imposed by sensor networks, and the problem of cheaply selecting one sensor node uniformly at random. In future work, there are numerous generalizations to consider. Our methods immediately generalize to queries that wish to sample nodes satisfying a geometric predicate, such as those within a region of interest, but we have not yet studied how to efficiently sample from nodes satisfying a non-geometric predicate. Another interesting question is how best to take advantage of parallelism when the number of samples needed or the expected number of attempts is high. Here, distinct probes may traverse common network links, so clever strategies may be able to reduce total transmission costs. We also plan to consider how to optimize sampling for queries which do not fall into a request/reply paradigm. For example, if query patterns are known in advance, such as periodic fixed queries, a more streamlined method for sampling that avoids explicit requests could be implemented in a decentralized fashion. However, our methods may still find use in answering such queries since their "on-demand" nature allows quick responses to unexpected events or failures.

Finally, we note that variants of our sampling methods can be applied much more broadly, outside the context of sensor networks. For example, uniform node sampling is also an important problem in structured P2P networks based on coordinate systems [9]. Variants of our methods apply to these P2P scenarios and provide a simpler and more topology-agnostic alternative to existing methods.

Acknowledgments

We are grateful to Deepak Ganesan and Stanislav Rost for the use of their sensor deployment maps and allowing them to be reproduced here. We thank Phil Gibbons, Kanishka Gupta, and Niky Riga for helpful conversations and feedback on earlier versions of this manuscript.

References

[1] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:630–659, 2000.

[2] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: how much is enough? In *Proc. of ACM SIGMOD '98*, pages 436–447, 1998.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. of the IEEE Int'l Conf. on Data Engineering*, March 2004.

[4] M. de Berg, O. Schwarzkopf, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[5] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with irregular spatio-temporal sampling in sensor networks. In *Proc. of HotNets-II*, November 2003.

[6] C.-C. Han, S. Ganeriwal, A. Boulis, and M. Srivastava. Going beyond nodal aggregates: Spatial average of a physical process in sensor networks. Poster in *ACM SenSys*, Nov. 2003.

[7] J. Hill and D. Culler. Mica: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, Nov/Dec 2002.

[8] B. Karp and H. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *ACM MobiCom*, Aug. 2000.

[9] V. King and J. Saia. Choosing a random peer. In *Proc. of ACM PODC'04*, July 2004.

[10] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 2nd. edition, 1981.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *USENIX OSDI*, 2002.

[12] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. of ACM SIGMOD '98*, pages 426–435, 1998.

[13] S. Nath and P. Gibbons. Synopsis diffusion for robust aggregation in sensor networks. Technical Report ITR-03-08, Intel Research, Aug. 2003.

[14] J. Newsome and D. Song. GEM: Graph EMbedding for routing and data-centric storage in sensor networks without geographic information. In *ACM SenSys '03*, pages 76–88, 2003.

[15] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *ACM MobiCom*, Sept. 2003.

[16] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensornets with GHT, a geographic hash table. *Mobile Networks and Applications*, 8(4):427–442, 2003.

[17] S. Rost and H. Balakrishnan. Lobcast: Reliable Data Dissemination in Wireless Sensor Networks. Under submission.

[18] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277. ACM Press, 1981.

[19] J. von Neumann. Various techniques used in connection with random digits. *U.S. National Bureau of Standards Applied Mathematics Series*, 12:36–38, 1951.

[20] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(3):9–18, 2002.

[21] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *Proc. of IEEE Infocom*, pages 1567–1576, June 2002.

Optimization of In-Network Data Reduction

Joseph M. Hellerstein*† Wei Wang*

*UC Berkeley and †Intel Research Berkeley

{jmh,wangwei}@eecs.berkeley.edu

Abstract

We consider the in-network computation of approximate "big picture" summaries in bandwidth-constrained sensor networks. First we review early work on computing the Haar wavelet decomposition as a User-Defined Aggregate in a sensor query engine. We argue that this technique can be significantly improved by choosing a function-specific network topology. We generalize this discussion to a loose definition of a 2-level optimization problem that maps from a function to what we call a *support graph* for the function, and from there to an aggregation tree that is chosen from possible subgraphs of the physical network connectivity. This work is frankly quite preliminary: we raise a number of questions but provide relatively few answers. The intent of the paper is to lay groundwork for discussion and further research.

1 Introduction

Wireless sensor networks must operate with significant constraints on energy and bandwidth consumption. This presents challenges for interactive analysis of data in sensornets, since data analysts tend to desire a big-picture view of the data before "drilling down" to specific queries. The big-picture queries can range over all the data in the network, but fortunately approximate answers are often sufficient for these purposes. Techniques to provide approximate answers to resource-intensive queries of this sort were explored by a variety of researchers in traditional database scenarios (e.g., [6, 8]).

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

In this paper we explore some initial ideas and challenges in performing online, in-network data reduction in sensor networks. Data reduction techniques can be used to provide synopses or "sketches" that can be used to approximately answer queries. Our main contribution here is not to present specific results, but to rough out a set of ideas and research challenges that we hope the community can explore and define further.

We begin by describing in some detail two techniques for in-network computation of Haar Wavelets. We hinge this discussion on the Haar *support tree*, a logical dataflow specification that describes the ordering constraints on combining data values. We show that an earlier idea for in-network computation of the Haar does not observe the constraints of the support tree, and instead produces biased results. We then consider constraining the network topology to generate a physical *communication tree* that observes the constraints of the logical Haar support tree. We present the surprising observation that a correct communication pattern for the Haar support tree results in a *binomial communication tree* at the network layer. This insight leads to some relatively crisp questions surrounding the optimization of communication topologies for computing Haar wavelets in-network.

Given this specific example as background, we pose a more generic (albeit vaguely defined) family of optimization problems for doing in-network data reduction, by focusing on the general problem of mapping from support graphs to communication graphs for various computations. We also raise various challenges in transferring this algorithmic work to practice.

2 Case Study: Wavelets

Wavelets have been widely used in the database literature as a data reduction technique (a tutorial is presented

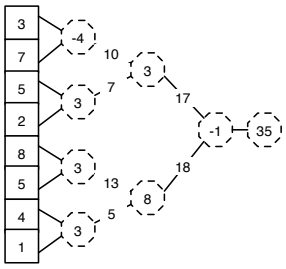


Figure 1: A column of a table and its Haar wavelet support tree (sometimes called an "error tree"). The output of the wavelet transform in this example is [35, -1, 3, 8, -4, 3, 3, 3].

in [11]). Aggregate queries can be answered approximately by running them over compressed wavelets of a raw dataset. Wavelets have a number of attractive properties, including their mathematical simplicity, and their ability to provide "multi-resolution" results by incrementally fetching more of the wavelet from a disk or network.

2.1 A Brief Primer on Haar Wavelets

The Haar wavelet is the simplest and most popular example of the wavelet family. The Haar is also easy to explain; we give a brief sketch here. Given an array of numbers (e.g., one column of a database table), it pairs up the neighboring numbers in odd and even positions (e.g. rows of the table), and transforms them into two different numbers: their sum and their difference. The differences are stored, and the sums are passed into a recursive application of the procedure. The recursion can be visualized as a tree, as in Figure 1[1]. The numbers ("coefficients") stored at each internal node in the tree represent the differences between the overall sum of leaves in the left and right subtrees of the node; the edges are labeled with the sums that are passed up. The root represents the sum of all the entries in the original array. We call this tree the *support tree* of the Haar wavelet: edges in the tree represent data dependencies, where each internal node is computed as a function of its children, and the leaves underneath a node represent the support of the value in that node. The output of the Haar transform can be produced by a breadth-first traversal of the (non-leaf) nodes of the support tree, though in practice there are coding algorithms that do not require constructing and traversing such a tree [16].

The decoding of the transformed data can be done in a straightforward fashion starting from the root and recursing downwards: given the overall sum s at the root, and the difference d at the node below, the overall sums of the left and right subtrees are calculated as $(s+d)/2$ and $(s-d)/2$ respectively, and the process can then recurse to the leaves.

As described, the output of the Haar transform is exactly the same size as the input. However, a simple scheme can be used to lossily compress the wavelet by truncating the list of coefficients. The basic idea is to only keep coefficients with high absolute values[2], and "round" the remaining coefficients to zero. In our example of Figure 1, truncating to the top 3 coefficients gives [35, 0, 0, 8, -4, 0, 0 ,0]. The resulting output array has mostly zero-valued entries, and can be represented compactly via a number of well-known techniques (e.g., via $(position, value)$ pairs for the non-zero entries, or run-length encoding.) Decoding our truncated example wavelet reconstructs the input as [2, 6.75, 4.375, 4.375, 6.125, 6.125, 2.125, 2.125]. Note that wherever a node in the support tree was rounded to zero, the reconstructed leaves in the corresponding subtree moved closer together in value. Dropping coefficients "smooths" differences in the original data.

If the full wavelet encoding is available somewhere – e.g. on a disk, or across a network – then the number of "unrounded" coefficients fetched locally can be increased incrementally in a "multi-resolution" manner, to remove these smoothing effects. Each new coefficient fixes a more subtle smoothing than the previous. This incremental improvement in the reconstruction is one attractive feature of wavelets.

A final side-note is merited regarding the treatment of set-valued data like columns of database tables. Wavelets are a sequence-encoding scheme, preserving the ordering of values in the input. In databases, this input ordering is arbitrary by definition. Given that any ordering is acceptable, an open question is to choose an ordering of the input data for which a wavelet truncated to the top k coefficients is most effective. For numeric data, sorting the table is a natural option; an extension of this idea for inte-

[1]The example builds a 1-d wavelet. Multi-d wavelets are analogously built with trees of fan-in 2^d.

[2]Typically the values are normalized by dividing by $\sqrt{2^i}$ where i is the height of the node above the leaves. Normalization does not affect the examples or algorithms here.

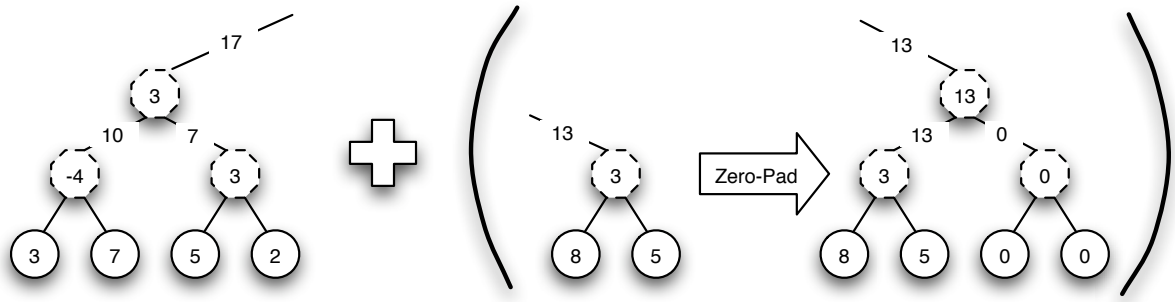


Figure 2: Given support subtrees of differing sizes, the PM technique zero-pads the smaller subtree before combining them.

ger data is the Wavelet Histogram, which run-length encodes the sorted column into (value, frequency) pairs and performs a wavelet transform on the resulting sorted frequencies [13]. For categorial attributes, the best choice of sort-order is an open question; it is likely to be dependent on the wavelet basis functions chosen (e.g. Haar, Daubechies-4, Mexican Hat, etc.)

2.2 Haar Wavelets as a Distributed UDA

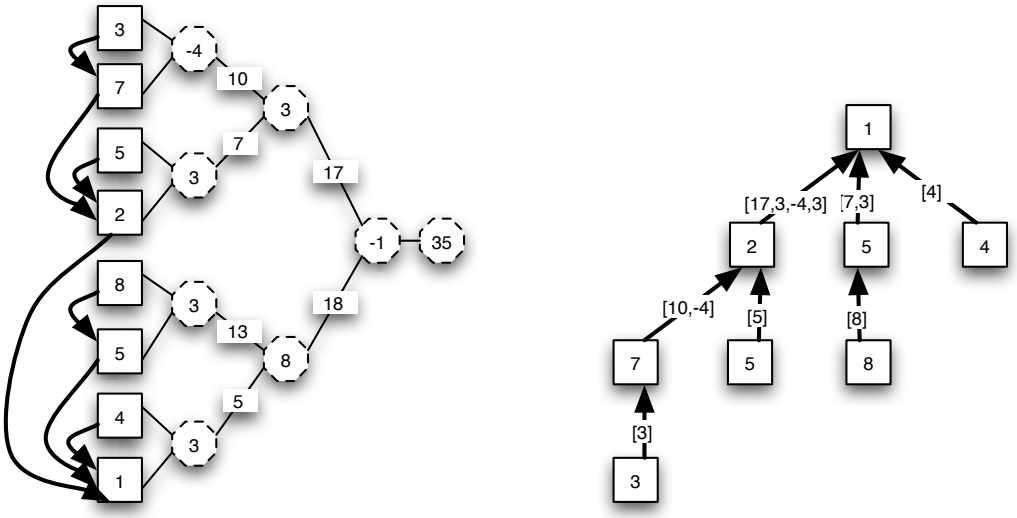
Earlier work based on the TinyDB system presented a User-Defined Aggregation (UDA) technique to compute a Haar wavelet over readings gathered in a sensor network [10]. We refer to this as the Pad-Merge or PM technique, and briefly review it here.

As in extensible databases, UDAs in TinyDB are represented by a triplet of functions: a merging function f, an initializer i, and an evaluator e. The initializer converts a scalar input value into an opaque *partial state record (PSR)*, the merging function takes two PSRs and combines them into a new PSR, and the evaluator takes a PSR and produces an output scalar value. In sensor-net query systems like TinyDB, an aggregation query is disseminated to participating sensor nodes, which call the initializer function on their local reading and then communicate PSRs up a *communication tree* of network links to the query node. When a node N receives a PSR from a child in the tree, it calls the merging function to merge the incoming PSR into N's current PSR; when N has merged in all of its children's PSRs, N sends the merged PSR to its own parent. Details of this aggregation scheme, including the dissemination of queries and construction of communication trees, can be found in the literature [12].

The PM technique uses a distributed, bottom-up scheme to construct a Haar support tree like that of Figure 1. It has a total communication cost that is linear in the number of nodes of the network (one fixed-size message per node). The PSRs in the PM technique are essentially arrays of k wavelet coefficients represented as $(position, value)$ pairs. Each PSR corresponds to a subtree of a complete Haar support tree. The main logic in the PM technique is in the merging function, which takes two arrays of wavelet coefficients (representing two Haar subtrees), generates a new set of wavelet coefficients representing the two trees connected by a new root, and keeps the top k of those coefficients as the new PSR[3]. Upon completion, the PM technique produces k large wavelet coefficients that can be used to lossily reconstruct the input data.

A Haar support tree is a balanced binary tree. But aggregation in TinyDB imposes no structure on the communication tree, and hence it does not control the order in which PSRs are merged. The merging function can be invoked on two *arbitrary* PSRs, which may represent Haar subtrees of differing heights. To handle this, the PM approach proposes a zero-padding technique to "promote" the smaller of the two input PSRs to a tree of the same height as the larger: it pads the smaller PSR with an appropriate number of zero-valued leaves until it becomes

[3]The order in which PSRs are combined recursively determines the left-to-right ordering of the leaves of the Haar tree. In our discussion here we focus on set-oriented query scenarios where this order – or, equivalently, IDs of the nodes – is insignificant. Preserving the order or node IDs can be done in a number of different ways that would complicate our discussion here unnecessarily.



[t]

Figure 3: An in-network computation of the Haar wavelet of Figure 1. The left side annotates the (logical) support tree with dark arrows representing physical message-passing between the sensor nodes. The right side of the figure shows just the (physical) communication tree, i.e., the leaf level of the left side. Each edge on the right is labeled with the wavelet coefficients sent.

a balanced binary tree of the same height as the larger PSR (Figure 2). This guarantees that the PM technique always merges two PSRs of the same size, and hence always constructs balanced binary Haar support trees.

If the PM technique never truncates any coefficients, it can reconstruct the data perfectly: the extra zeros introduced by padding can be correctly accounted for and deleted in the decoding process. However, in the practical cases where the PSR is much smaller than the number of nodes in the network, each merging step has to truncate to the top k coefficients. When zero-padding is used, the truncating can smooth the spurious zeros across the true data. In the end, the PM technique will produce a k-coefficient wavelet that is not as accurate as the one that would be produced in a centralized implementation of the Haar encoding – the PM wavelet will incorrectly bias the reconstructed data toward zero, in many cases in a significant way.

2.3 Haar-Specific Network Topologies

The PM technique introduces bias when padding Haar support subtrees of unequal size. Imagine that one could guarantee that only equal-sized subtrees were merged. Then no zero-padding would be needed, and the correct top-k wavelet coefficients would be produced as a result. In this section we explore the possibility of achieving such an invariant by controlling the sensor network topology used for aggregation in the network.

For purposes of illustration, assume for a moment that we have a *fully-connected* communication network with nodes numbered 1 through 2^l. Our goal is to construct the Haar support tree bottom-up by passing messages between nodes. By convention, we will assume that lower-numbered nodes will pass messages to higher-numbered nodes. The process begins at the leaves of the support tree: node 1 passes its value to node 2, node 3 passes its value to node 4, etc. The even-numbered recipients pass along PSRs that contain their top k difference coefficients as well as their sum: node 2 passes its PSR to node 4, node 6 passes its PSR to node 8, etc. At the end of this process, the contents of the Haar support tree would be distributed throughout the network, with the top-k coefficients and the overall sum residing at node 2^l. This communication pattern is depicted by the directed arrows in the left side of Figure 3.

Given our assumption of a fully connected sensor network graph, this distributed algorithm employs a very stylized subgraph that comes from the data structures literature: the *binomial tree* [4] (right hand side of Fig-

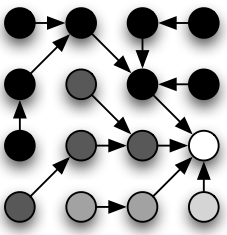


Figure 5: A binomial tree embedded in a radius-1 grid.

ure 3). In a binomial tree of 2^l nodes, the root has l children, which are binomial trees of 2^i nodes for $i \in 0, \ldots, l-1$. The depth and maximum fan-in of a binomial tree are both logarithmic in the number of nodes.

We can now relax our unrealistic requirement of full connectivity in the sensor network, and ask whether this technique is feasible in practice. This reduces to two basic questions: (1) do binomial trees naturally occur as subgraphs of practical sensornet communication graphs, and if so, then (2) can an efficient, distributed topology-selection algorithm be devised to find and maintain a binomial subtree topology in a sensor network?

It would be interesting to study this question empirically, and/or to analyze it formally for random graphs from typical distributions. Here we simply provide a bit of intuition from the canonical simplistic sensornet model of an equally-spaced 2-d grid of nodes with communication radius of 1 grid-square per node. In a 4×4 grid, it is certainly possible to find binomial trees (Figure 5). Note however that in two dimensions each node has only 8 neighbors, and the root of a binomial tree of size 2^l has l children. Hence clearly any 2-d grid topology of more than 256 nodes will not have a binomial tree embedding unless its communication radius is greater than 1. Similarly, since the corner of a grid has only 3 neighbors, there is no binomial tree rooted at a corner of our 4×4 grid of Figure 5.

3 Generalizing the Haar Example

Haar wavelets are only one of many non-trivial aggregation functions that may be of use in sensor networks. The discussion above illustrates a number of interesting, general problems that arise in computing such complex aggregates efficiently. In this section we briefly sketch a set of research problems that arise in this space.

3.1 A Static Optimization Problem

Section 2.3 raises the challenge of finding communication trees that match the Haar wavelet support tree. This is an example of a more general optimization problem in sensornet aggregation. The challenge is to take any aggregation function and map it onto the graph of radio connectivity in the network. This can be viewed as a multi-layer optimization problem: as illustrated in Figure 4: (a) a support graph must be chosen for the aggregation function, and (b) the support graph must be mapped onto a communication tree; the communication tree in turn is constrained to be a subgraph of (c) the radio connectivity graph of the sensornet. Note that depending on the aggregation function, there may be more than one satisfying support graph for step (a). Similarly, in step (b) there are multiple communication trees corresponding to a chosen support graph, more than one of which may be a subgraph of the radio connectivity.

In the case of the Haar wavelet, the mapping from support graph to communication graph was quite elegant: a balanced binary support tree became a binomial communication tree. Since the properties of binomial trees are well known, they are amenable to analysis and (hopefully) simple construction and maintenance algorithms. It is unclear whether the mappings of other support graphs into communication graphs will be as elegant. The curious reader is encouraged to play with the Daubechies-4 wavelet as a more complex example, since it has a support DAG rather than a support tree. The general mapping problem itself is of interest, as is the question of characterizing the communication graphs at the output.

As noted in the previous section, it may in some cases be impossible to find a communication graph in the network to match a particular support graph for a function. In such cases, two options are available. One is to achieve such a topology as an *overlay* network, by having some sensors *forward* PSR messages directly to other nodes without applying the merging function. This of course causes overheads that spoil the ideal linear communication cost of many aggregates. The second option is to always apply the merging function on arriving PSRs regardless of data dependencies in the support graph; logically this reshapes the support graph that gets computed. This is exactly the approach taken by the PM technique for Haar wavelets. Ideally this latter approach

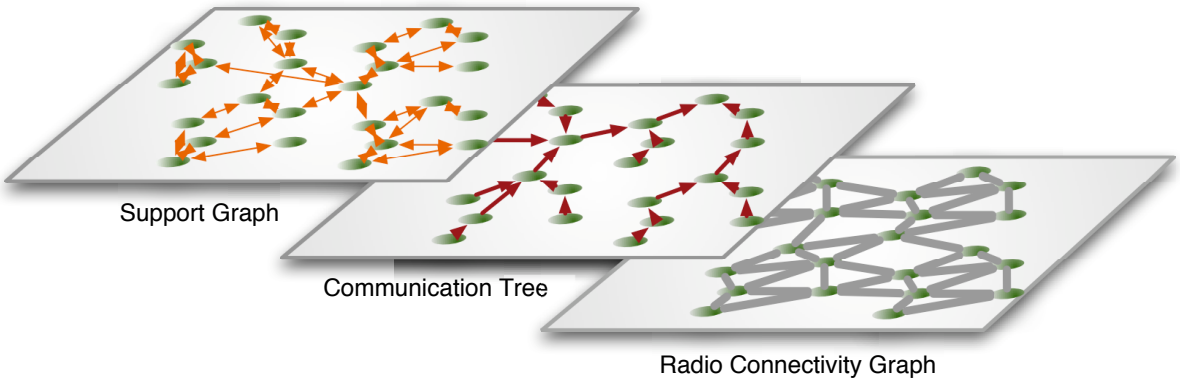


Figure 4: The general optimization problem needs to choose a Support Graph, and map it to a Communication Graph that is a subgraph of the Radio Connectivity Graph.

should include a technique to quantify the error introduced by such inappropriate merging.

The general optimization problem is as follows. Given an aggregation function, a connectivity graph, and a cost function to minimize, the challenge is to choose a min-cost communication graph in the network that is a subgraph of the connectivity graph. The communication graph must be annotated to differentiate between cases of PSR forwarding and PSR merging. The cost function is likely to be a multi-objective metric, incorporating perhaps such issues as bandwidth, latency, power consumption, and bounds on errors in the result.

3.2 Real-World Complications

This optimization problem is relatively well-defined, but not entirely realistic. Here we highlight additional challenges that are likely to arise in practice.

The first is the very real issue of packet loss in sensor networks. Loss probabilities on radio links can be estimated, and added as inputs to the optimization problem. But this leaves the question of how to deal with loss. A natural option is to implement network retries; the expected number and cost of retries can be translated in the cost metric to bandwidth, latency and power consumption. A second option is simply to tolerate loss, and estimate the loss in accuracy of the answer. A third, intriguing direction is the use of forward error correction. Naive application of error-correcting codes seems like a bad idea, since the codes are traditionally used to preserve opaque packets. Given our knowledge of application semantics, it is interesting to explore the joint design of error-correcting aggregation functions. The recent work on duplicate-insensitive distinct count sketching [3] may seen as an example of this idea. A generic challenge with any of these schemes is to minimize waking time: if a node chooses not to propagate any data (e.g., because its coefficients are below a threshhold) it should be able to power down. This is complicated by the problem of loss, since it is unclear how receivers differentiate between lost packets and unsent packets.

A second critical challenge is that of network dynamism. Experience shows that connectivity in a sensornet changes over time as a function of many factors. Given that the physical graph will change over time, a dynamic reoptimization technique is needed for the problems sketched above, and preferably one that works in a distributed fashion with minimal communication requirements.

An additional, fundamental challenge arises at the architectural level. This paper advocates algorithmic optimizations that collapse traditional boundaries between application-level logic and various parts of the network stack (e.g. topology construction, loss handling, etc.) This raises the challenge of architecting a system that allows users defining new aggregation functions to describe acceptable networking choices with a minimum of fuss. This is an extensibility interface that is not well understood. A better understanding of this interface might also provide guidance in choosing data reduction functions to compute. For example, the support graphs of various wavelet variants (Haar, Daubechies-4, etc.) are

quite different. Understanding how to *describe* these differences compactly to a system might also provide analytical insight into their relative merits in terms of mappability to communication graphs.

Finally, this discussion raises the question of what one does with multiple concurrent functions with competing desires – e.g. a query that requests the simultaneous computation of two very different aggregates.

4 Open Issues and Alternatives

This paper describes a relatively focused family of optimization challenges. In this section we briefly touch on some broader issues and alternative approaches.

An important challenge in this context is to handle changes in the data while the aggregation protocol is running. Multi-resolution schemes like wavelets can let users watch detail accumulate as coefficients are passed up in multiple rounds of communication, in the spirit of Online Aggregation [9]. However, during the multiple rounds of communication, the data itself may be changing, and it may be more beneficial to send newer, coarse-grained data rather than increasing refinements on stale data. In this vein, it might be beneficial look at spatio-temporal wavelet encoding, and consider which coefficients of the spatio-temporal wavelet to communicate at each timestep. This tradeoff emcompasses data properties and user desires, and it inherently a mix of systems, coding, and HCI issues.

The traditional database approach to aggregation has a unidirectional dataflow that results in the one-way communication trees we have discussed here. A broad class of data analysis techniques can be more efficiently computed in two communication rounds: one up a tree and the other back down. This includes multi-dimensional regression, Fast Fourier Transforms, and Bayesian belief propagation, all of which can be computed via the Junction Tree algorithm [1]. These techniques have been mapped into the sensornet domain in recent years [7, 15]. But current sensornet query engines have yet to incorporate these approaches into their architectures or languages, and the integration may require a new architecture beyond analogies to Object-Relational UDAs. It is worth noting that many of the problems suggested here are related to work being studied in the Junction Tree context [15].

Another fruitful vein of exploration is to design data reduction techniques whose merging function is fully commutative and associative. The network optimization for these aggregates is therefore unconstrained by the choice of support tree. AMS sketches [2] are one example that may be a good alternative to wavelets. Nath and Gibbons propose a scheme to additionally introduce duplicate insensitivity to aggregates in a general way [14]. Duplicate insensitivity removes the constraint of the communication graph being a tree, allowing for arbitrary "diffusion" or "gossip" of messages.

Wavelets have been proposed for sensor networks in the work of Ganesan, et al. on DIMENSIONS [5]. DIMENSIONS does not perform any distributed wavelet computation. Instead it has two main components: (a) it uses local wavelets to lossily compress archival storage of readings over time at each node in the network, and (b) it embeds a geographic quad-tree in the network to provide distributed, hierarchical spatial summarization. Each node of the quad tree receives the (wavelet-encoded) data from the nodes below, decodes it to form a 2-d array, and re-encodes the array into (threshholded) 2-d wavelet coefficients used both for lossy local storage and for communication further up the quad-tree. DIMENSIONS blends two approaches to hierarchical data reduction: local wavelets and distributed quad trees. An interesting question is whether a distributed multi-dimensional wavelet of the form described in this paper could be extended appropriately to achieve the functionality of DIMENSIONS in a unified fashion.

5 Conclusion

If sensornet query engines are to succeed, they need to either provide a wide range of useful built-in functionality, or be easily extended to incorporate new functionalities. Given the relative immaturity of the area, it is unlikely that we will anticipate many of the important features in advance. The traditional User-Defined Aggregation functionality of extensible databases should be a key feature in sensornet query systems, and optimization of UDAs over networks will be a key challenge. Perhaps the most critical aspect of the work described here is architectural challenge raised: how do users define the merging rules for complex UDAs to the system, and are there general optimization techniques to take such rules and use them to achieve good performance?

Acknowledgments

Thanks for conversation and feedback to Amol Deshpande, Christos Faloutsos, Minos Garofalakis, Phil Gibbons, Carlos Guestrin, Sam Madden, Yossi Matias, Mark Paskin, Kannan Ramchandran, and Mehul Shah. Mark Paskin devised the visualization of the layered optimization problem for his work on distributed inference [15].

References

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Trans. Info. Theory*, 46(2), 2000.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 20–29, Philadelphia, PA, 1996.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *Proc. International Conference on Data Engineering (ICDE)*, Mar. 2004.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, 2001.

[5] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Estrin, and J. Heidemann. An evaluation of multi-resolution storage for sensor networks. In *Proc. First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[6] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. M. andf S. Muthukrishnan, S. Ramaswamy, and T. Suel. AQUA: System and techniques for approximate query answering. Technical report, Bell Laboratories, Murray Hill, NJ, Feb. 1998.

[7] C. Guestrin, R. Thibaux, P. Bodik, M. A. Paskin, and S. Madden. Distributed regression: An efficient framework for modeling sensor network data. In *Proc. 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*, 2004.

[8] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis with CONTROL. *IEEE Computer*, 32(8), August 1999.

[9] J. M. Hellerstein, P. J. Haas, and H. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD*, pages 171–182, Tucson, AZ, May 1997.

[10] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Towards sophisticated sensing with queries. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN)*, 2003.

[11] D. Keim and M. Heczko. Wavelets and their applications in databases. In *Proc. International Conference on Data Engineering (ICDE)*, Heidelberg, Germany, 2001.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Symp. Operating Systems Design and Implementation (OSDI)*, 2002.

[13] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD*, pages 448–459, Seattle, Washington, 1998.

[14] S. Nath and P. B. Gibbons. Synopsis diffusion for robust aggregation in sensor networks. Technical Report IRP-TR-03-08, Intel Research, 2003.

[15] M. A. Paskin and C. E. Guestrin. A robust architecture for distributed inference in sensor networks. Technical Report IRB-TR-03-039, Intel Research, 2003. Submitted for publication.

[16] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1997.

WaveScheduling: Energy-Efficient Data Dissemination for Sensor Networks

Niki Trigoni, Yong Yao, Alan Demers, Johannes Gehrke
Cornell University
Ithaca, New York 14850
{niki,yao,ademers,johannes}.cs.cornell.edu

Rajmohan Rajaraman
Northeastern University
Boston, Massachusetts 02115
rraj@ccs.neu.edu

Abstract

Sensor networks are being increasingly deployed for diverse monitoring applications. Event data are collected at various sensors and sent to selected storage nodes for further in-network processing. Since sensor nodes have strong constraints on their energy usage, this data transfer needs to be energy-efficient to maximize network lifetime. In this paper, we propose a novel methodology for trading energy versus latency in sensor database systems. We propose a new protocol that carefully schedules message transmissions so as to avoid collisions at the MAC layer. Since all nodes adhere to the schedule, their radios can be off most of the time and they only wake up during well-defined time intervals. We show how routing protocols can be optimized to interact symbiotically with the scheduling decisions, resulting in significant energy savings at the cost of higher latency. We demonstrate the effectiveness of our approach by means of a thorough simulation study.

1 Introduction

Sensor networks consisting of small nodes with sensing, computation and communication capabilities are becoming ubiquitous. A powerful paradigm that has emerged recently views a sensor network as a distributed SensorDBMS and allows users to extract information by injecting declarative queries in a variant of SQL. In deploying a SensorDBMS one should consider important limitations of sensor nodes on computation, communication and power consumption. Energy is the most valuable resource for unattended battery-powered nodes. Since radio communication consumes most of the available power, SensorDBMSs need energy-efficient data-dissemination techniques in order to extend their lifetime.

An important communication pattern within sensor networks is the sending of sensor readings to a designated sensor node. Let us give two examples where this pattern arises. First, consider a heterogeneous sensor network with two types of sensor nodes: many small-scale *source nodes* with low-power multi-hop communication capabilities, and a few powerful *gateway nodes* connected to the Internet. In this setup, data flows from the sources to the gateway nodes. Our second example is motivated by resource savings through *in-network processing*. In-network processing algorithms coordinate data collection and processing in the network at designated nodes called *view nodes* [1, 2]. Data flows from sources to relevant view nodes for further processing. For example, in a sensor network that monitors a remote island and records the movements of different types of animals, each view node could be responsible for storing the detection records (and computing tracks) for a given type of animal.

Since power is a major resource constraint, we would like this data flow between sources and view nodes to be as power-efficient as possible; in particular, for non-time-critical applications, we would like to trade message latency versus power usage as events are routed from the sensor nodes where they originated to the respective view nodes.

In order to achieve energy-efficient data flows between sources and view nodes, we address several challenges intrinsic to ad hoc network communication: minimizing collisions at the MAC layer, managing radios in a power-efficient manner, and selecting energy-efficient routes. In this paper we consider data dissemination strategies that avoid collisions (and message retransmissions) at the cost of higher message latency. We carefully coordinate transmissions between nodes, allowing them to turn their radios off most of the time. Current generation radios consume nearly as much power when listening or receiving as when transmitting (typical idle:receive:transmit ratios are 1:1.2:1.7 [3], 1:2:2.5 [4], and 1:1.05:1.4 [5]). Thus, the ability to turn them off when not needed yields significant energy savings.

The remainder of this paper is organized as follows. Section 2 enumerates several variants of scheduling problems and discusses their complexity. Section 3 presents our scheduling algorithm and highlights its close interaction with the routing layer. A thorough experimental evaluation of the proposed algorithm and competing approaches is presented in Section 4. We discuss related work in Section 5 and draw our conclusions in Section 6.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

2 Problem space

With coordinated scheduling, a data dissemination protocol in a sensor network has two components: a *scheduling algorithm* that activates network edges so that their transmissions do not interfere with one another, and a *routing algorithm* that selects routes for individual messages. Two important performance metrics are *energy consumption* and *message latency*. In this section, we consider each of these metrics and sketch complexity results for the following optimization problems: (i) finding an optimal pair of routing and scheduling algorithms; (ii) finding an optimal routing algorithm for a given schedule; (iii) finding an optimal schedule for a given collection of routes. Full proofs of these results can be found in an extended version of the paper [6].

The underlying framework for our optimization problems is as follows. We assume the sensor nodes form a multi-hop wireless network embedded in the plane. For simplicity, we assume the node radios have identical ranges of one unit. Thus, the nodes form a *unit disk graph:* two nodes are connected by an edge iff the Euclidean distance between them is at most 1. We represent the communication workload by the rate of message generation at each node i, given by r_i, together with a probability distribution p_{ij}, giving the probability that a message generated at node i is destined for node j.

Energy minimization. In the *energy minimization problem*, we are given a communication workload among the sensor nodes and view servers, and our goal is to determine a data dissemination scheme that minimizes the energy consumed in delivering all messages within a bounded delay. In our model, we assume that the energy consumed when a network edge is activated is $(\alpha + \beta m)$, where α is a fixed *start-up cost* for turning the radio on, β is the per-message transmission and reception cost, and m is the number of messages sent during the activation.

Theorem 2.1 *For any $\alpha > 0$ and $\beta > 0$, finding an optimal routing-scheduling pair to minimize energy is NP-hard, even when there is only one view server. It is also NP-hard to determine an energy-optimal activation schedule given a fixed set of routes. The problem of finding a set of energy-optimal routes given an activation schedule can be solved in polynomial time.*

Latency minimization. In the *latency minimization problem*, we are given a communication workload and seek a data dissemination protocol that minimizes average message propagation latency. It is already known that minimizing latency in an *ad-hoc* wireless network is NP-hard even for the special case where nodes exchange messages only with their neighbors [7]. This reduction can be extended to unit disk graphs.

Theorem 2.2 *Finding a routing-scheduling pair that minimizes latency is NP-hard. It is also NP-hard to determine an optimal activation schedule given a fixed set of routes. A set of latency-optimal routes for a given activation schedule can be obtained in polynomial time.*

These results indicate that the general problem of designing an optimal data dissemination protocol, given an arbitrary sensor workload, is intractable. In this paper, we focus on one element of the design space, namely that of first developing an interference-free schedule for edge activation, and then designing delay- or energy- optimal routes given this schedule.

3 Wave Scheduling and Routing

In this section, we focus on developing a schedule for edge activations, and then designing optimal routes given this schedule. Our scheduling mechanism is defined over a simple partitioning of the network, which we first describe in Section 3.1. We then select a class of periodic schedules, presented in Section 3.2, which are aimed at avoiding collisions at the MAC layer. Finally, in Section 3.3, we present energy-based and delay-based routing protocols that optimize the relevant metric for a given schedule.

3.1 Partitioning

Our scheduling mechanism is layered on top of a protocol like GAF [8], which partitions nodes into cells and periodically elects a single leader node for each nonempty cell. Nodes determine the cell that they belong to by using distributed localization techniques [9, 10]; experiments have shown that GAF is robust to somewhat inaccurate position information [8]. The size of each cell is set so that a node anywhere in a cell can communicate directly with nodes in any of its four horizontal and vertical neighbor cells. This constrains the side of a cell to have length L at most $R/\sqrt{5}$, where R is the transmission range of a node. Since only leaders are engaged in inter-cell message routing, the remaining nodes may turn off their radios most of the time, achieving significant energy savings. The schedules that we will propose in this section exploit the GAF topology control scheme in order to achieve further energy savings. They leverage the abstraction of partitioning irregularly positioned nodes into cells organized in a rectilinear grid and focus on coordinating inter-cell communication. In what follows, we will refer to cells as *supernodes* or simply *nodes*.

For convenience of exposition, we assume here that the rectilinear grid is a square. Let N denote the number of supernodes along an edge of the grid. We identify the supernodes by their coordinates (i, j); for example $(0, 0)$ refers to the node at the southwest corner of the network. Thus, $(i+1, j)$, $(i, j+1)$, $(i-1, j)$, and $(i, j-1)$ are the east, north, west, and south neighbors, respectively, of node (i, j), for $i, j \in [0, N)$.

3.2 Wave Scheduling: Algorithms

Given a set of supernodes arranged in a rectilinear grid, we propose a class of periodic activation schedules that conserves energy by (i) avoiding interference at the MAC layer and (ii) allowing supernodes to turn off their radios whenever they are not sending or receiving messages. In these schedules, which we call *wave schedules*, every (directed) edge of the rectilinear grid is activated periodically at well-defined communication intervals, called *send-receive* intervals. For any two neighboring supernodes A and B, the edge $A \to B$ is activated in the send-receive intervals $[t + iP, t + iP + \delta]$, for every $i \geq 0$, where t is the first time the edge is activated and P is the period of the schedule. We now elaborate on the edge activation

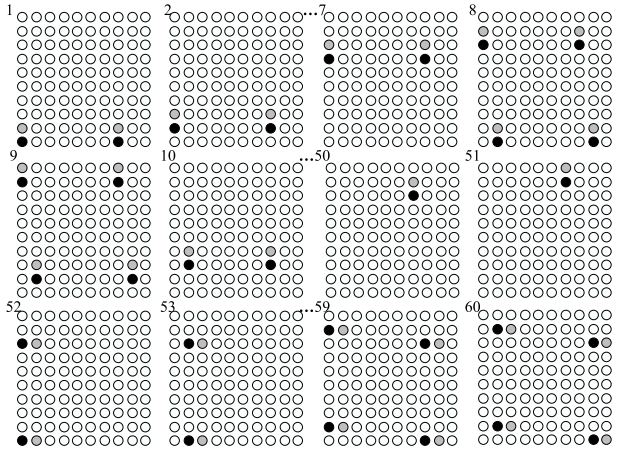


Figure 1: SimpleWave

step and then present two wave schedules: SimpleWave and PipelinedWave.

Edge activation. An edge activation $A \rightarrow B$ consists of a contention-based and a collision-free period. During the contention-based period, all nodes within the cell A turn on their radios in order to run the GAF protocol (GAF only runs locally in cell A). They check whether the leader has enough energy reserves to continue assuming the leadership role. If the leader is energy-drained, a re-election protocol selects the new leader. Messages in the queue of the old leader, as well as inter-cell routing information, are transfered to the new leader. The remaining nodes then send their sensor readings, which were generated since the previous GAF period, to the leader of the cell. Contention resolution MAC protocols work very well in avoiding intra-cell contention, since all nodes in the cell are within communication range and there are no occurences of the hidden terminal problem. This adapted version of the GAF protocol is more energy-efficient than the original GAF scheme, because it avoids interference caused by concurrent leader reelection in consecutive cells.

The collision-free period of an edge activation $A \rightarrow B$ is used in order to route messages from the leader of A to the leader of B. During that period both leaders of A and B (referred to simply as A and B) turn on their radios preparing for message transmission and reception respectively. If A has no data messages to send, it sends a special *NothingToSend* (NTS) message to node B, which allows both nodes to turn off their radios without having to wait until the end of the *send-receive* interval. As we will show in the experimental section, the use of NTS messages offers significant energy savings since it adjusts the node duty cycle to its local traffic. Since in the collision-free periods there is no interference at the wireless medium, it is not necessary to exchange RTS and CTS messages prior to sending a regular data message (or an NTS message). A data (or NTS) message is simply followed by an ACK. The first data or NTS message that A sends to B (and its ACK) can be used in order to resynchronize the clocks of the two nodes for the next activation of edge $A \rightarrow B$. If the synchronization error between two neighbor nodes at the beginning of the collision-free period is bound by e msecs, we set the receiver B to wake up e msecs earlier than scheduled according to its local clock. Synchronization issues are discussed in more detail in the end of this section.

In the remainder of the paper, by edge activation we mainly refer to the collision-free period of the edge activation used for inter-cell communication. The ratio of the collision-free period to the contention-based period depends on the traffic patterns of the application. For instance, for traffic workloads with messages following multiple hops before reaching the destination, the collision-free (inter-cell communication) period should dominate the contention-based (GAF) period.

SimpleWave. The intuition behind wave schedules is to coordinate message propagation in north, east, south and west phases. For instance, during the east phase, only edges of the form $(i, j) \rightarrow (i + 1, j)$ are activated sending messages along the east direction. Owing to interference, however, we cannot schedule all of the edges along the east direction. If Δ denotes the ratio of the interference range to the transmission range, then a sufficient condition for transmissions from two supernodes (i, j) and (i_1, j_1) to avoid interference is the following:

$$\sqrt{(i - i_1 - 1)^2 + (j - j_1 - 1)^2} \cdot L \geq \Delta \cdot R$$

In particular, if we consider two supernodes (i, j) and (i_1, j), then their transmissions do not interfere it $i - i_1 - 1 \geq \Delta R/L$. Since $i - i_1 - 1$ is an integer, we obtain that the two supernodes can transmit simultaneously if $i - i_1 \geq \lceil \Delta \cdot R/L \rceil + 1$, which we denote by g. If we adopt the IEEE 802.11 settings of $R = 250m$ and $\Delta = 550/250$, and set L to its minimum value $R/\sqrt{5}$, we obtain that $g = 6$.

In the SimpleWave schedule, we schedule together edges that are g positions apart. Figure 1 illustrates the SimpleWave schedule on a 10×10 network, with $R = 250m$, $\Delta = 550/250$, setting L to a round number of $100m$ (instead of its minimum value $R/\sqrt{5}$), yielding $g = 7$. The north phase starts at time 1 and it lasts for 51 send-receive intervals during which every north edge is activated exactly once. The following east phase starts at time 52. Notice that only two nodes of the first column ($(0, 0)$ and $(0, 7)$) are sending concurrently to the east, which are spaced apart by 7 hops. In the next interval (time 53) the pattern shifts east by one cell. Only when the wave has propagated to the eighth column (time 59) does it no longer interfere with node communication in the first two columns. Note that at time 59 it becomes possible to schedule concurrently four edges: $(7, 0) \rightarrow (8, 0)$, $(7, 7) \rightarrow (8, 7)$, $(0, 1) \rightarrow (1, 1)$ and $(0, 8) \rightarrow (1, 8)$.

There are variants of the *SimpleWave* algorithm defined above, differing by the order in which wave directions are scheduled. We refer to these as the (N, E, S, W), (N, W, S, E), (N, S, E, W), and so forth. The variants are logically equivalent, but the choice of scheduling variant affects the choice of routes, as will be explained in detail in section 3.3. The period of a *SimpleWave* depends on the size of the network. Each phase takes $(N - 1) + (g - 1) \cdot g$ send-receive intervals and the entire wave period lasts for $4 * ((N - 1) + (g - 1) \cdot g)$ intervals. This is not a desirable property, because it prevents the distributed deployment of the algorithm in a dynamic network. When a new supernode (cell) joins (or leaves) the network, it affects the

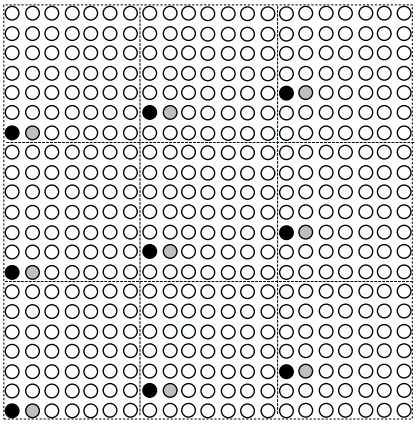


Figure 2: PipelinedWave

wave period and therefore the activation times of all the other supernodes. In addition, in order to identify the activation time of its adjacent edges a supernode should know its location within the network, as well as the size of the network. Another important downside of the *SimpleWave* algorithm is that it underutilizes the capacity of the network. For instance notice in Figure 1 that at time 1, it activates only two north going edges, whereas one could identify two additional edges that could be activated concurrently without causing any interference.

PipelinedWave. This algorithm is motivated by the need for schedules that can be deployed in a distributed and scalable manner, and that make a good use of the network capacity. Conceptually, a network can be divided in a number of fixed-size ($g \times g$) squares of g^2 supernodes each, where all squares have identical schedules. In such a network, the schedule of the incident edges of a node is determined by its relative location in the square. Since all edges within the same square interfere with one another, we can only schedule one edge at a time. In effect, we partition all the edges of the network into a collection of *maximal independent sets*, each independent set corresponding to a set of edges that can be simultaneously activated without interference. The period of the resultant schedule is $4g^2$ send-receive intervals. This means that for pipelined waves, new nodes can join the network and schedule themselves without affecting the schedules of existing nodes. If a supernode joins an existing square, it waits for at most one period in order to interact with its neighbors and locally determine its location with respect to them and therefore its local coordinates within the square. By overhearing the schedules of its immediate neighbors it determines the time at which it should schedule itself in each direction. A similar local interaction occurs when a new supernode joins the network initializing a new square. When a node leaves the network, the schedules of the remaining supernodes do not change.

Note that in the *PipelinedWave* algorithm two edges are scheduled concurrently if they have the same direction and the sender nodes (and the receiver nodes) have exactly the same local coordinates within a $g \times g$ square. This implies that the algorithm avoids all interference at the MAC layer. It schedules a maximum number of non-interfering edges at each send-receive interval thus increasing the network capacity with respect to the *SimpleWave* algorithm. It is easily deployable in a distributed manner, since local coordination suffices for scheduling a new supernode. Finally, it is scalable because the node schedules are not affected by the size of the network.

A modified version of the *PipelinedWave* algorithm does not define identical schedules for each square, but schedules shifted by g positions with respect to the schedules of the four neighbor squares. More specifically, the east wave of a square is shifted g positions (send-receive intervals) earlier than the east wave of the west neighbor square, the north wave is shifted g positions earlier than the north wave of the south neighbor square etc. A snapshot of the modified *PipelinedWave* algorithm (during the east phase) is shown in Figure 2. The east phase in a given (dotted) square proceeds by shifting one edge to the right and moving to the row below when the entire row of the square is traversed. Notice that by the time an entire row is traversed in a given square, the respective row of the right neighbor square just starts being traversed. The new pipelined algorithm decreases the latency of message delivery at the square boundaries; this will become evident when we describe *delay-based* routing in Section 3.3. The south, west and north phases are scheduled in a similar manner. This improved *PipelinedWave* is the schedule evaluated in our experiments in Section 4.

Another tunable parameter in *PipelinedWave* is the number of send-receive intervals for each direction (phase) before the wave switches to another direction. Our experiments show that this parameter, referred to as *step*, has no noticeable impact on the performance of the wave schedule [6]. In Section 4, we evaluate the variant of *PipelinedWave* with *step=1*.

Synchronization. We briefly discuss two synchronization requirements imposed by wave schedules: i) neighbor nodes must have the same notion of time regarding their communication slot and ii) nodes in the *close* neighborhood must be well synchronized so that only edges at least g positions away are scheduled simultaneously. Acknowledging that perfect time synchronization is hard to achieve, we relax the initial requirements and propose a *fault-tolerant* version of wave schedules. If the drift between two neighbor clocks does not exceed ϵ, nodes that are g positions away from each other are synchronized within $g\epsilon$. In every edge activation, we schedule the receiver to turn on the radio ϵ time units earlier than the scheduled time according to its local clock. In order to ensure that there is going to be no interference due to the clock errors, we can increase the distance between two non-interfering edge activations (e.g. from 7 to 8). Notice that although a perfectly aligned wave schedule implies global synchronization, a reasonable implementation of waves is achievable by ensuring that nodes are well-synchronized with neighbors within interference range.

Recently proposed synchronization protocols for sensor networks (e.g., RBS [11] and TPSN [12]) provide tight synchronization bounds (e.g., $0.02ms$ for neighbor nodes [12]) and exhibit good multi-hop behavior. Their performance however is bound to decay for very large networks (an open problem that we discuss in Section 4); in this case we assume that a few GPS-equipped nodes will undertake the synchronization task for their local regions.

3.3 Routing

The proposed wave schedules are TDMA-based MAC protocols that assign periodic transmission slots to inter-cell communication. Wave schedules are general-purpose energy-efficient MAC protocols that can potentially be combined with arbitrary routing protocols. In this section we consider two important metrics for evaluating the efficiency of a routing algorithm, namely *node energy consumption* and *message propagation latency*. Note that energy-optimal routes do not depend on the underlying wave schedule, whereas latency-optimal routes are intrinsically coupled with it.

Energy-based routing. As noted in Section 2, minimum energy routing is achieved by routing along shortest hop paths. We adopt a simple flooding approach that evaluates minimum-hop paths from all nodes in the network to a given view node. Flooding initiated at a view node results in the construction of a tree connecting all supernodes to the root (view) node, as described in [13]. Since we consider more than one view, the minimum-hop routes form a forest of trees built on top of the grid overlay.

Each node maintains an in-memory routing table of size proportional to the number of view servers. For each view server, the routing table includes a 2-bit entry giving the direction of the next hop towards the view. This simple approach works even in the presence of "holes" (empty cells), as is shown by Madden et al. [13]. Dynamic node failures (which manifest themselves as the appearance of new holes) can be dealt with by a local flooding phase to repair affected routes, as in AODV, or by introduction of a greedy face-routing mode as in GPSR [14, 15]. Alternatively, a node that fails to deliver a message may store it in memory until the next flooding phase that reconstructs the tree.

Delay-based routing. We propose a *delay-based* routing algorithm that, given a certain wave schedule, minimizes message latency between a pair of source and view nodes. Each node C maintains a routing table, that contains for each view V and each neighbor N a triple $\langle V, N, d\rangle$, where d is the latency of the minimum-latency path from C to V among all paths with the next-hop being N that C is presently aware of. On updating a routing entry, node C also sends the information $\langle V, N, d\rangle$ to its neighbors. On the receipt of such a message, neighbor N^* of C does the following: i) it evaluates the time dt that a message sent over $N^* \rightarrow C$ remains at C before being forwarded with the next wave via $C \rightarrow N$ towards view V; ii) if an entry $\langle V, C, d'\rangle$ with $d' < d + dt$ exists in the routing table of N^*, then the routing message is dropped - otherwise, the routing entry is replaced by $\langle V, C, d + dt\rangle$.

When the above distributed algorithm converges, every node has determined the minimum-latency paths to each view. Routing messages can be piggy-backed on regular or NothingToSend messages as in the case of energy-based routes. Local repairs can be performed as in the case of energy-based routing, but by considering latency as the primary metric for evaluating the goodness of a route.

4 Experimental Evaluation

We implemented a prototype of wave scheduling in the NS-2 Network Simulator [16] and compared its performance with other approaches. In Section 4.1, we test the behaviour of wave schedules under different routing metrics, as well as varying the number of views and empty cells. Section 4.2 presents the performance of two competing tree-based scheduling approaches and Section 4.3 shows the behavior of IEEE 802.11 with various duty cycles. A comparison of wave schedules with the other approaches is presented in section 4.4.

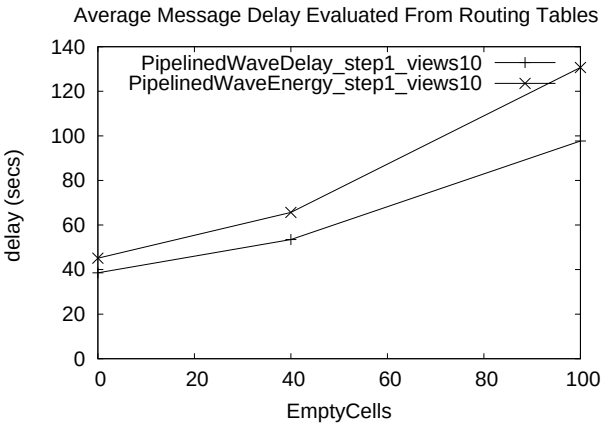


Figure 3: Delay vs energy routing

4.1 Wave Scheduling

We simulate a network of 20 by 20 grid cells of size $100m^2$ each. The ratio of interference to communication range is 550/250 and the ratios between radio idle, receive and transmit power are 1:1.2:1.6. Every edge activation between two consecutive cells lasts for 200ms. In the wave schedules, all routing happens at the level of the grid overlay network. A node can send about 10 packets during an edge activation given a link bandwidth of 20kbps. The receiver wakes up 30ms before the sender to avoid message loss when clocks are subject to small drifts.

The size of a square in a pipelined wave is set to 8 by 8 grid cells (instead of 7 by 7) in order to avoid interference as a result of small synchronization errors. Experiments run for 1000 seconds and the traffic workload varies from 0 to 2500 messages. The time that a message is generated is selected at random, uniformly over the simulation period. The source location of a message is randomly selected to be any of the non-empty cells, and the destination to be any of the views. Cells containing views and empty cells are randomly distributed in the network.

Energy- vs. delay-based routing. We first compare the behavior of the PipelinedWave schedule under two wave routing metrics: minimum hop-count and minimum-delay. Recall from Section 3.2 that due to the scheduling of the waves, the path with minimum delay is not necessarily the path of minimum hop count. Figure 3 shows the average path delay under light load for the two metrics, i.e. it shows the time between generation of a message at its source and delivery of the message at its destination. This delay is computed by deriving information from the routing tables of the nodes. It coincides with the real message propagation delay when the traffic is low and nodes can completely drain their buffer during an edge activation. The minimum-energy routing metric defines paths

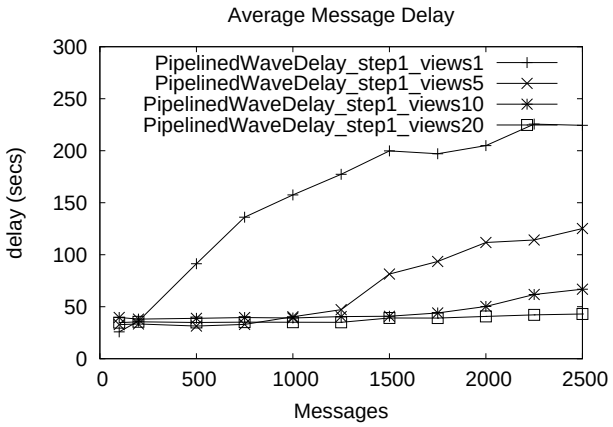


Figure 4: Effect of views on delay

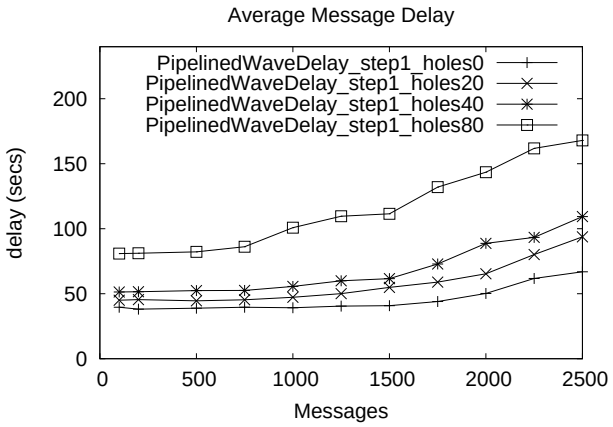


Figure 6: Effect of holes on delay

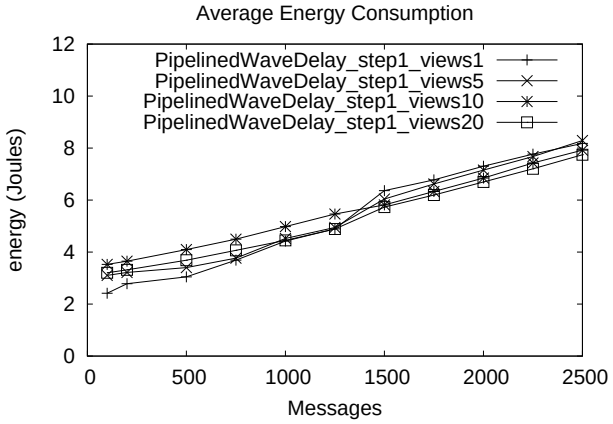


Figure 5: Effect of views on energy

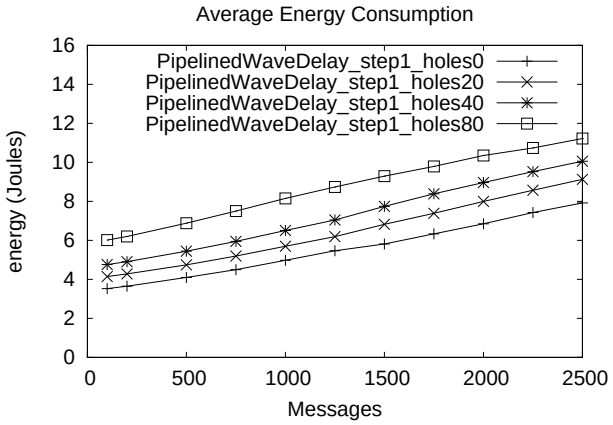


Figure 7: Effect of holes on energy

with higher delay than the minimum-delay metric and the gap increases as we increase the number of holes from 0 to 100 (25% of all cells). For 100 holes (or empty cells), the minimum-energy metric yields paths that are 30% slower than the minimum-delay metric. The energy overhead of the minimum-delay metric was observed to be negligible. In the remainder of the section, we use minimum-delay as the default routing metric.

Scalability with the number of views. Our second experiment shows the scalability of our scheme with respect to the number of view nodes. Figure 4 shows the average observed message delay, which captures queueing delay due to traffic. We set the number of empty cells to be 0. With more view nodes, the load is better balanced across the network, the average message propagation delay is smaller and the overall capacity of the network increases. With more than 200 messages for a single view the network is overloaded, and the queues in the network start to grow, and they would continue to grow without bounds if we would not have limited the length of the experiment to 1000 seconds. Figure 5 shows that the energy usage of the wave does not increase with the number of views, for a given number of messages. This confirms the nice behavior of wave routing which makes it exceptionally suitable for sensor networks with multiple *gateway* (or *view*) nodes.

Effect of empty cells. We also examine the impact of empty cells on the performance of wave schedules. The number of views is 10 and a randomly selected set of 0 to 80 cells are set to be holes. Figure 6 shows that the message latency increases with the number of holes: messages wait longer in order to make a turn to bypass a hole. The capacity of the network is only 500 messages for 20% (80) holes (the message delay increases considerably after that point), whereas it rises to more than 1500 for networks without holes. Interestingly, the average energy consumption per non-empty cell (per node) increases with the number of empty cells, as shown in Figure 7. Although fewer messages are delivered per time unit, these messages follow longer paths. Thus every node ends up routing more messages and spending more energy.

4.2 Tree Scheduling

We compare wave scheduling with an existing tree-based scheduling and routing scheme [13]. Trees are generated using a flooding mechanism initiated at each view node. Every node selects as its parent the neighbor on the shortest path to the root (view). It is therefore expected that the paths used in tree schedules are shorter than paths used in waves, since the latter are built on top of the grid overlay. Routing in a tree is trivial: each non-view node forwards every message it receives to its parent. In a tree-

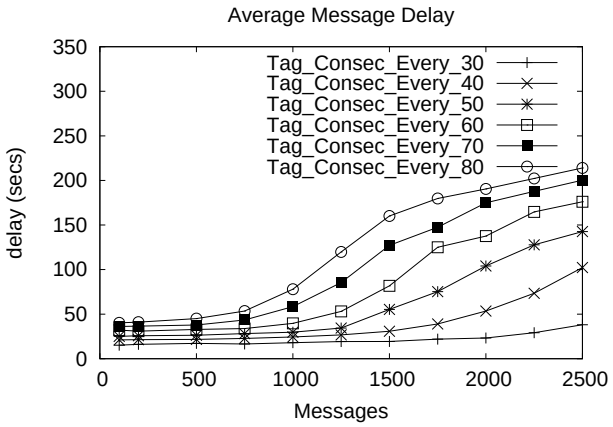


Figure 8: Delay: consecutive trees

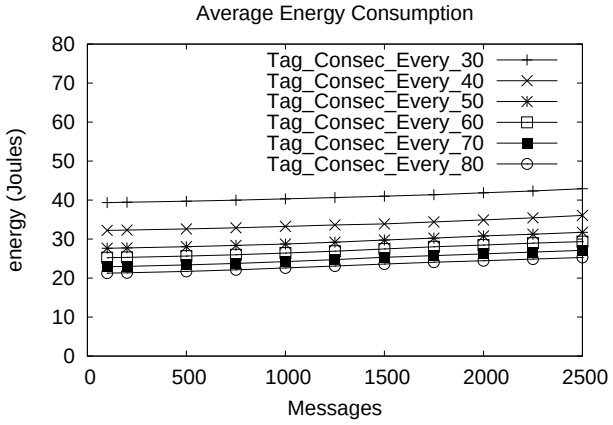


Figure 9: Energy: consecutive trees

based schedule, we activate edges in reverse order of their distance from the root, enabling a message to propagate from any leaf of the tree to the view node in a single tree activation period. Every tree edge is activated for 200ms, as in the case of the wave.

To generalize tree scheduling to handle multiple views, we construct a collection of spanning trees, one tree rooted at each view server. An edge activation schedule can then be derived in several ways. At one extreme is a *conservative* schedule, which is simply a concatenation of schedules for the individual trees. The simplest conservative schedule is to activate tree rooted at view $i+1$ immediately after all edges of tree rooted at view i have been activated. In this simple conservative schedule, latency grows linearly with the number of views. In our experiments we study energy-efficient variants of this simple schedule: We define a period p of repeating the activation of every tree. If we have m views, the first tree is activated at times $\{0, p, \ldots\}$, the second at $\{p/m, p+p/m, \ldots\}$, and so on. We assume that the interval p/m is long enough to activate all edges of a single tree, so that consecutive activations do not overlap. In Figures 8 and 9, these schedules are referred to as $Tag\_Consec\_Every\_p$, where p is the period between two activations of the same tree.

At the other extreme, we consider *aggressive* schedules that activate all trees in parallel. In the simplest aggressive schedule, which is called Tag_Parall, consecutive activations of the same tree follow one another immediately after completion. In order to study power-saving variants of the aggressive schedules, we consider periodic activations of the same tree. In our experiments, we use the name $Tag\_Parall\_Every\_p$ to refer to aggressive schedules in which all trees are activated concurrently every p seconds (Figures 10 and 11).

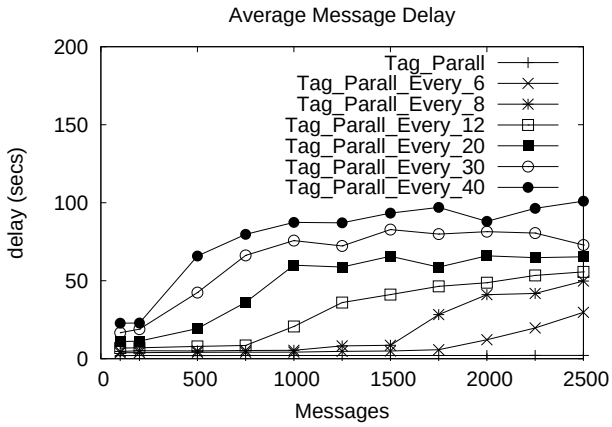


Figure 10: Delay: parallel trees

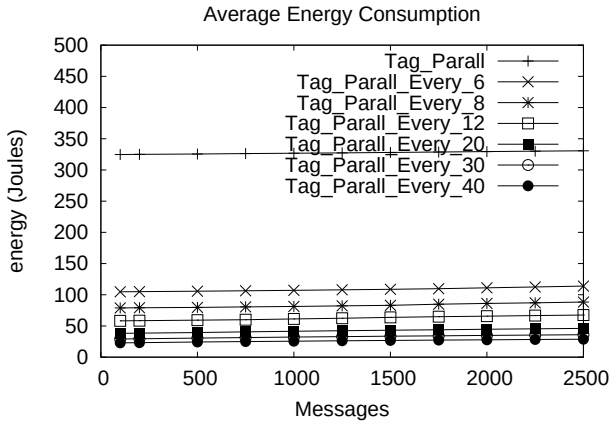


Figure 11: Energy: parallel trees

In both consecutive and parallel schedules, we observe a graceful tradeoff between energy and delay. As the activation period increases, the energy decreases at the expense of higher message latency and smaller network capacity. Applications aiming at energy preservation should take into consideration the traffic load in order to determine an energy-efficient tree schedule. For instance, the most energy-efficient consecutive schedule that achieves a capacity of 1000 messages has period 60 seconds (Figure 8). Likewise, the most energy-efficient parallel schedule that achieves a capacity of 1000 messages is activated approximately every 12 seconds (Figure 10). Beyond 1000 messages (per 1000 seconds), the delay for these two schedules starts increasing and it would increase without bounds had we continued to generate messages with the same rate for longer periods.

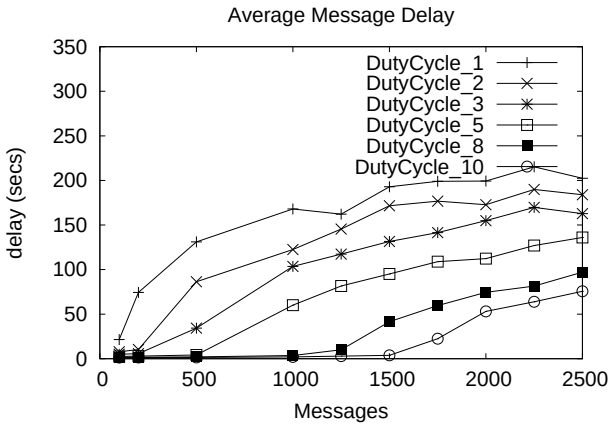


Figure 12: Delay: 802.11

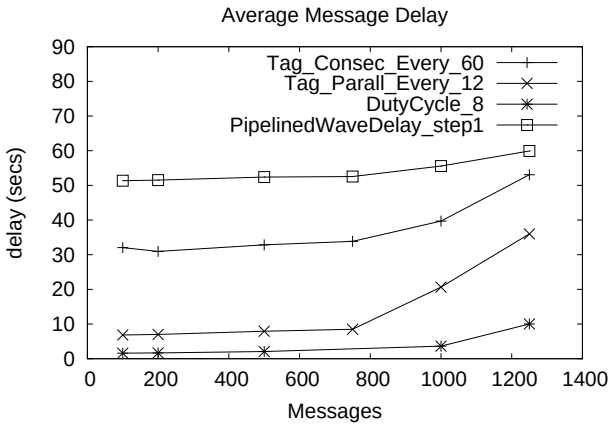


Figure 14: Comparing schemes

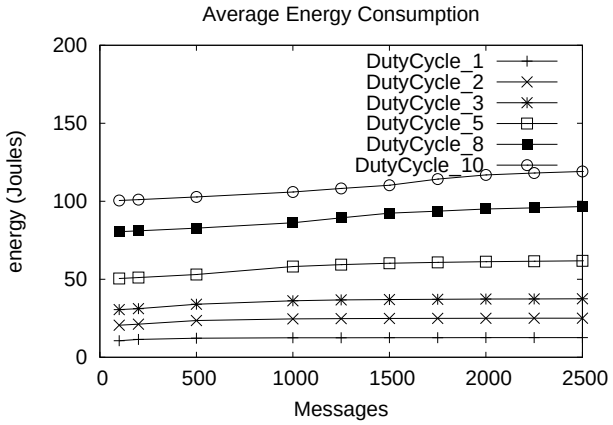


Figure 13: Energy: 802.11

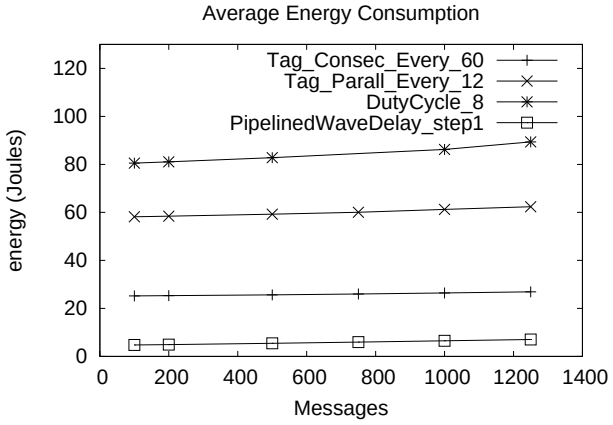


Figure 15: Comparing schemes

4.3 IEEE 802.11 with Different Duty Cycles

Besides tree scheduling, in which edges are activated in reverse order of their distance to the root, we also study power-conserving variants of the IEEE 802.11 protocol. We vary the duty cycle of the protocol, by turning off the radio regularly and allowing communication only 1 to 10% of the time. The performance of the resulting schemes, named $Duty\_Cycle\_x$, is shown in Figures 12 and 13. Routing is performed as in tree-scheduling, i.e. messages follow the shortest paths to the views. Notice that for a load of 1000 messages we can only select duty cycles greater than 8%, otherwise the traffic exceeds network capacity and the queues increase without bound. The reader can see trends in energy and delay similar to those observed in the tree-scheduling schemes. As the duty cycle decreaases, the average message delay decreases significantly at the expense of higher energy usage.

4.4 Comparison with Other Schemes

In order to compare different protocols we first select a traffic load and then consider only protocols that can serve this load without exceeding capacity (the point at which average delay begins to increase). We compare the most energy-efficient versions of different protocols (with 10 views and 10% empty cells): for 1000 messages, we select the variants $Tag\_Consec\_Every\_60$, $Tag\_Parall\_Every\_12$, $Duty\_Cycle\_8$ and the pipelined wave with step 1. From the previous graphs, the reader can see that these are the variants of different protocols that accomodate the given traffic with the least energy consumption.

Figure 14 shows that the wave protocol has the longest delay, followed by the consecutive tree schedule, the parallel tree schedule and the 802.11 (with duty cycle 8%). The reverse pattern is observed with respect to node energy consumption in Figure 15. The wave protocol is at one extreme, offering the most energy savings (better by an order of magnitude than any other scheme) at the cost of higher delay. The 802.11 protocol with duty cycle 8% is at the other extreme offering very small message delays at the cost of higher energy. The energy-delay tradeoff of the two tree scheduling algorithms is also worth observing: activating trees consecutively (as opposed to concurrently) saves energy because it avoids interference among different trees, but it incurs higher message latencies.

5 Related Work

The advent of sensor network technology has recently attracted a lot of attention to MAC and routing protocols

that are specifically tailored for energy-constrained ad-hoc wireless systems.

MAC protocols: Medium access protocols are divided into two main categories, contention-based and schedule-based protocols, depending on whether they *resolve* or completely *avoid* collisions at the wireless medium. IEEE 802.11 [17] is the most widely used contention-based protocol; although nodes can periodically switch to a power saving mode, in the active periods they suffer from interference and overhearing. The PAMAS MAC-level protocol turns radios off when nodes are not communicating [18], but it requires a second channel for RTS-CTS messages. PicoNet also allows nodes to turn off their radios [19]; a node wishing to communicate must stay awake listening for a broadcast message announcing its neighbor's reactivation. In S-MAC [20, 21], nodes are locally synchronized to follow a periodic listen and sleep scheme. S-MAC does not explicitly avoid contention for the medium, but reduces the period of overhearing by sending long DATA packets annotated with their lengths. Sift [22] is a randomized C-SMA protocol that aims at reducing latency, rather than energy, in case of spatially-correlated contention.

Schedule-based MAC protocols conserve energy by avoiding message retransmissions or idle listening [23, 24, 25]. NAMA [24] and TRAMA [25] avoid all collisions at the MAC layer by announcing the schedules of nodes in the 2-hop neighborhood and electing nodes to transmit in a given time slot. Our waves avoid schedule propagation overhead, at the expense of having fixed slots for every edge activation. Fixed assignment of communication slots affects message latency, but not the energy consumption at the nodes. TRAMA does not consider interference due to ACK messages, since it assumes that nodes that are three hops away can schedule transmissions cuncurrently.

GAF (Geographical Adaptive Fidelity) [8, 26] is a topology control scheme that conserves energy by identifying nodes that are equivalent from a routing perspective (belong to the same cell) and then turning off unnecessary nodes. The proposed wave algorithms are tightly integrated with the GAF protocol. Unlike S-MAC (a contention-based scheme) and TRAMA (a schedule-based scheme), under low traffic, the propagation delay of messages from a source to a destination over a multi-hop path is almost *constant*. It depends only on the topology of the network, i.e. which cells are empty, which does not change very rapidly. This desirable property stems from the fact that wave schedules coordinate radio usage across the sensor network.

Routing algorithms: Several routing protocols for ad-hoc networks have been proposed in the literature [27]. There has also been a plethora of work on energy-aware routing [18, 28, 29] but without considering the interplay of routing and scheduling. The TinyDB Project at Berkeley investigates tree-based routing and scheduling techniques for sensor networks [13, 30]. Tree-based routing is tightly combined with node scheduling; all nodes in the same level of the tree are scheduled to send messages to their parents concurrently at a time interval that depends on their distance from the root. Tree-based routing and scheduling is a representative example of the tight coupling between the MAC and routing layers in sensor networks. In this paper we have shown a different kind of interaction, namely how given a certain schedule of edge activations, we can identify routes that yield minimum message delays.

An energy-efficient aggregation tree using data-centric reinforcement strategies is proposed in [31]. A two-tier approach for data dissemination to multiple mobile sinks is discussed in [32]. Pearlman et al. [33] propose an energy dependent participation scheme, where a node periodically re-evaluates its participation in the network based on the residual energy in its battery. GEAR [29] uses energy-aware neighbor selection to route a packet towards a target region and restricted flooding to disseminate the packet inside the destination region; it addresses the problem of energy conservation from a routing perspective without considering the interplay of routing and node scheduling.

6 Conclusions and Future Work

In this paper, we have presented a class of algorithms that allow us to trade energy versus delay for data dissemination in sensor networks. Our approach is based on carefully *scheduling* the sensor nodes so that each node can stay idle most of the time, turning on its radio only at scheduled intervals during which it can receive or send a message. Our experiments show that the proposed wave scheduling algorithm results in significant energy savings at the expense of increased message latency.

In the future, we plan to study irregular wave schedules, in which we relax the current assumption that every directed edge in the network is scheduled regularly once per period, and thus has the same capacity. In practice, incoming edges to view nodes are expected to be more heavily loaded than edges at the border of the network. We believe that better network utilization can be achieved by considering a more general class of wave schedules in which different edges are activated with different rates. For instance, the network can be divided into highways (frequently-activated edges) and driveways (low-capacity edges). It would be interesting to study the tradeoff between energy and delay in such an irregular model.

Another interesting direction is to investigate the problem of time synchronization for wave schedules. Existing approaches, like RBS [11] and TPSN [12], provide tight synchronization bounds and exhibit good multi-hop behavior – with high probability, the error is less than linear in the number of hops. Using a tree-based approach, they aim at providing a global timescale exceeding the more relaxed requirements of wave schedules. Their performance is therefore bound to decay for very large networks. We intend to investigate highly distributed and scalable algorithms that are specifically tailored to achieve good time synchronization among nodes within interference range, instead of achieving global synchronization.

References

[1] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "Ght: A geographic hash table for data-centric storage," in *WSNA*, 2002.

[2] A. Ghose, J. Grossklags, and J. Chuang, "Resilient data-centric storage in wireless ad-hoc sensor networks," in *MDM*, 2003, pp. 45–62.

[3] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, "Span: A energy-efficient coordination algorithm

for topology maintenance in ad hoc wireless networks," *ACM Wireless Networks*, vol. 8, no. 5, September 2002.

[4] O. Kasten, "Energy consumption," Tech. Rep., ETH-Zurich, 2001.

[5] M. Stemm and R. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices," *IEICE Transactions on Communications*, vol. E80-B, pp. 1125–1131, 1997.

[6] N. Trigoni, Y. Yao, A. Demers, J. Gehrke and R. Rajaraman, "WaveScheduling: Energy-Efficient Data Dissemination for Sensor Networks," 2004 cougar.cs.cornell.edu.

[7] A. Sen and M. Huson, "A new model for scheduling packet radio networks," in *INFOCOM*, 1996, pp. 1116–1124.

[8] Y. Xu, J. Heidemann, and D Estrin, "Geography-informed energy conservation for ad hoc routing," in *MOBICOM*, 2001, pp. 70–84.

[9] P Bahl and V.N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *INFOCOM (2)*, 2000, pp. 775–784.

[10] N. Bulusu, J. Heidemann, and D. Estrin, "Gps-less low cost outdoor localization for very small devices," 2000.

[11] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review, SI: Physical Interface*, vol. 36, pp. 147–163, 2002.

[12] S. Ganeriwal, R. Kumar, and M.B. Srivastava, "Timing-sync protocol for sensor networks," in *SENSYS*, 2003, pp. 138–149.

[13] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," in *OSDI*, 2002.

[14] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, no. 6, pp. 609–616, 2001.

[15] B. Karp and H.T. Kung, "GPSR: greedy perimeter stateless routing for wireless networks," in *MOBICOM*, 2000, pp. 243–254.

[16] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[17] IEEE Computer Society, "Wireless LAN medium access control (mac) and physical layer specification," IEEE Std 802.11, 1999.

[18] S. Singh, M. Woo, and C.S. Raghavendra, "Power-aware routing in mobile ad hoc networks," ACM SIGMOBILE, 1998, pp. 181–190, ACM Press.

[19] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, and D. Leask, "Piconet: Embedded Mobile Networking," *IEEE Personal Communications*, vol. 4, no. 5, pp. 8–15, Oct. 1997.

[20] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *INFOCOM*, 2002, pp. 1567–1576.

[21] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated, adaptive sleeping for wireless sensor networks," Tech. Rep. ISI-TR-567, USC/Information Sciences Institute, January 2003.

[22] K. Jamieson, H. Balakrishnan, and Y.C. Tay, "Sift: A mac protocol for event-driven wireless sensor networks," Tech. Rep., MIT, May 2003.

[23] G. J. Pottie and W. J. Kaiser, "Embedding the Internet: wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–51, May 2000.

[24] L. Bao and J.J. Garcia-Luna-Aceves, "A new approach to channel access scheduling for ad hoc networks," in *MOBICOM*, 2001, pp. 210–221.

[25] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves, "Energy-efficient collision-free medium access control for wireless sensor networks," in *SENSYS*, 2003, pp. 181–192.

[26] Y. Xu, S. Bien, Y. Mori, J. Heidemann, and D. Estrin, "Topology control protocols to conserve energy inwireless ad hoc networks," Tech. Rep. 6, University of California, Los Angeles, Center for Embedded Networked Computing, January 2003, submitted for publication.

[27] C.E. Perkins, *Ad hoc networking*, Addison-Wesley Longman Publishing Co., Inc., 2001.

[28] J.-H. Chang and L. Tassiulas, "Energy conserving routing in wireless ad-hoc networks," in *INFOCOM*, 2000, pp. 22–31.

[29] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks," Tech. Rep. UCLA/CSD-TR-01-0023, University of Southern California, May 2001.

[30] J.M. Hellerstein, W. Hong, S. Madden, and K. Stanek, "Beyond average: Towards sophisticated sensing with queries," in *IPSN*, 2003.

[31] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," ACM SIGMOBILE, 2000, pp. 56–67, ACM Press.

[32] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *MOBICOM*, 2002.

[33] M.R. Pearlman, J. Deng, B. Liang, and Z.J. Haas, "Elective participation in ad hoc networks based on energy consumption," in *IEEE GLOBECOM*, 2002, pp. 17–21.

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 for modeling, emulation, and analysis of data of wireless sensor networks. This project is motivated by the unique need of 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 of sensor networks 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, are scattered geographically, and may be mobile. In order to facilitate studying 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.

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 for collecting sensory data. In TinyDB, the data collection process is the execution of declarative, SQL-like queries, which eases application development and allows for optimization for performance. However, if we would like to answer some important questions about the query processor for the application, we find it is difficult or infeasible to obtain the answers through a simple field study. Specifically, some of the 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 in order to observe important phenomena such as trends in temperature, humidity, and frog croaks around the frog pond?

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

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

(3) If we change the type of sensor nodes (e.g., CPU, radio channel, sensing units), the routing scheme of the sensor network, or the data collection queries, what will be the new 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 estimate power consumption in various scenarios realistically 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 the 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 initial steps 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 some 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 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 previous studies with a goal of answering 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 limit. However, it 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.

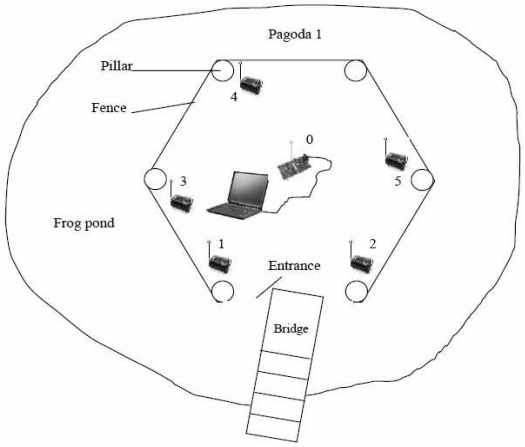


Figure 1: Deployment of Group 1 Motes

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 of Group 1 were similar to one another at a point of time. We picked two motes that differed most in the readings, Motes 1 and 5, to show in Figure 3. These readings captured frog croaks mainly. They indicate that frogs croaked most actively in the early morning and were most quiet during 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.

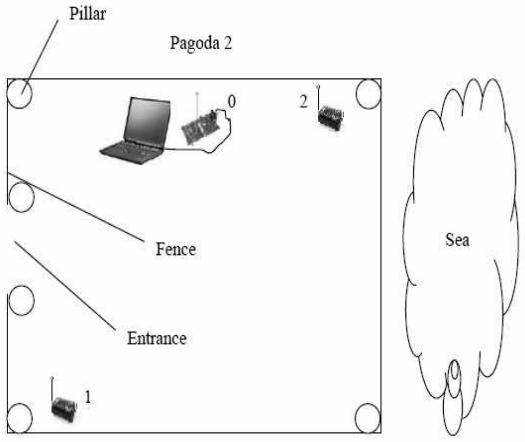


Figure 2: Deployment of Group 2 Motes

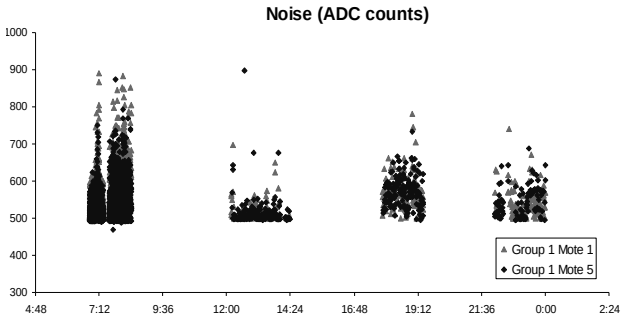


Figure 3: Group 1 Noise Readings

The humidity readings of Group 2 remained at the level of around 90% most of the time (Figure 4). There were some readings of abnormally high humidity (larger than 130%) of Mote 1 at the beginning of the morning period. These abnormal readings were because some rain drops splashed onto Mote 1 by accident when we took it out of a box and deployed it. The water made the humidity sensor of Mote 1 malfunction and to return abnormally high readings. This kind of physical problems 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 a distance of 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.

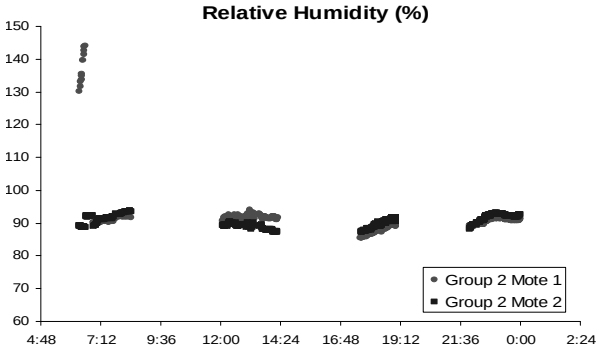


Figure 4: Group 2 Humidity Readings

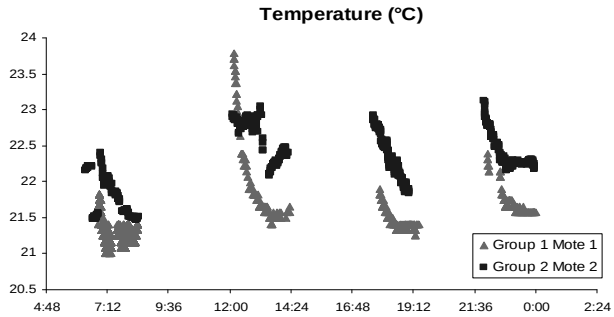


Figure 5: Temperature Readings of Two Groups

2.3 Discussion

From our data analysis, we suggest that the application just 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 have already 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 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 hard-ware-specific calibration of the readings, and the function plot() plots data points and curves together with analytical summaries following user-defined criteria.

(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 kinds 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 sometimes are 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 will be 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, if one sensor node detects a sudden increase of noise level, it might be a real event as well as a system error. But if multiple nearby nodes report the same event, the probability of a system error is much lower than that 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 of 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 by 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 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 power efficiency of sensor databases systematically. The main reason is that there are many intertwined factors that affect power consumption in a sensor database system, for instance, sensor node computation, wireless transmission, and various query processing techniques. Therefore, we propose to represent these factors in a general model for studying 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 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 main factors in wireless communication that affect power consumption in a sensor network. It adapts the quantitative approach provided by the sensor node model to a network environment. The sensor database model formalizes 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 estimate power consumption of query workloads realistically. 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.

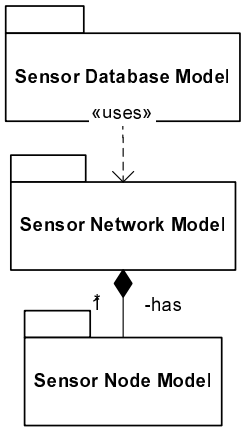


Figure 6: Model Hierarchy

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 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 occurred during this 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. Obviously, different routing schemes have different power consumption characteristics. Our sensor network model aims to provide insights for designing power-efficient routing schemes.

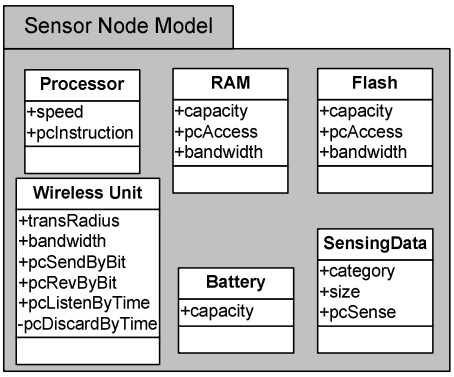


Figure 7: Sensor Node Package

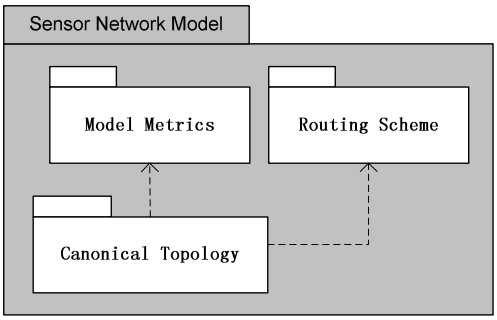


Figure 8: Sensor Network Package

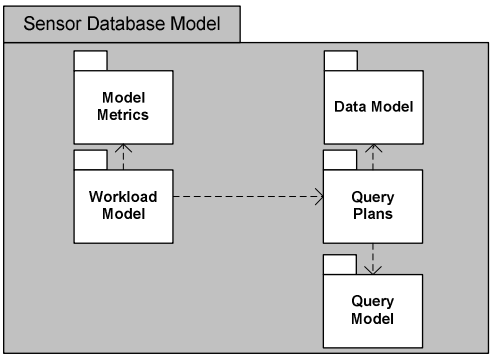


Figure 9: Sensor Database Package

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 to be 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 have developed algorithms for estimation of sensor network lifetime in terms of power consumption in the static (the routing tree does not change as long as the network topology does not change) and dynamic (the routing tree changes dynamically) deployment respectively. 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 in the following steps:

(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 there are no nodes run out of battery, there will still be 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 have 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 an area of a circle 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.

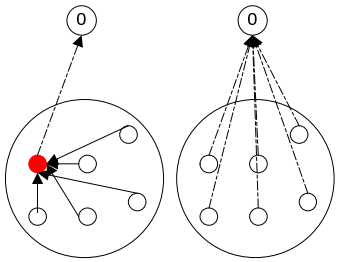


Figure 10: LEACH (left) versus Flooding (right)

Figure 11 shows the predicted average node lifetime in a network of *N* (ranging from 6 to 24) nodes resulted from our model. Our model predicts that LEACH results in 5-times improvement on power efficiency over flooding whereas in the original LEACH paper this factor was 8. One major reason for this difference is that we considered power consumption of database workloads as well as individual sensor nodes in addition to networking.

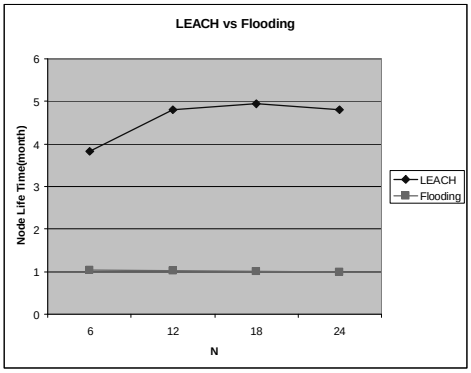


Figure 11: Predicted Average Node Lifetime

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 fairly realistically by using real-world factors such as sensor node hardware configuration, representative routing schemes, and typical queries in monitoring applications. In order 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 node neighborhood in the sensor network model by considering synchronization characteristics of transmission. A neighborhood of a node is a basic topology element in a multi-hop networking environment and transmission between nodes can be synchronous or asynchronous. We have modeled transmission to be synchronous as commonly assumed by existing work. In order to achieve 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. Using the reduction factor only 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, sometimes are 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 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 data 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 one other through the virtual channel, which is implemented on top of UDP (User Datagram Protocol) on a LAN (Local Area Network) and simulates a real radio channel with bit errors and delays.

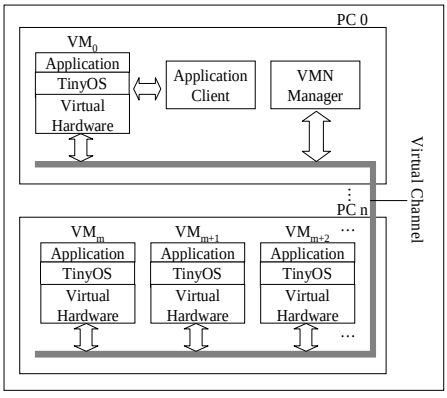


Figure 12: Architecture of a VMN

Each VM (Figure 13) emulates a MICA2 mote running TinyOS. We partition a VM into the upper layer

and the lower layer. 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 is to identify the components that are pertinent to program execution and then to put these components into the upper layer. Consequently, it is solely the task of the upper layer to emulate the environment such that the real code of a query processing application for a real sensor mote runs on a VM as if it runs on the real mote.

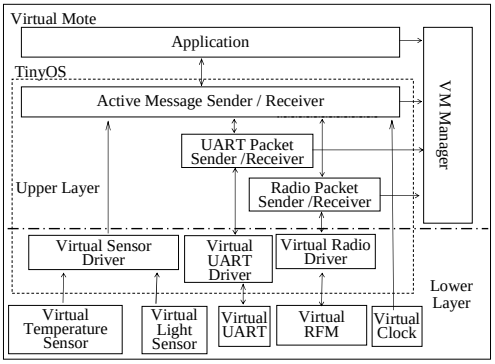


Figure 13: Architecture of a VM

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 (number of error bits received by the receiver) / (number of total 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 will wait 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 the 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).

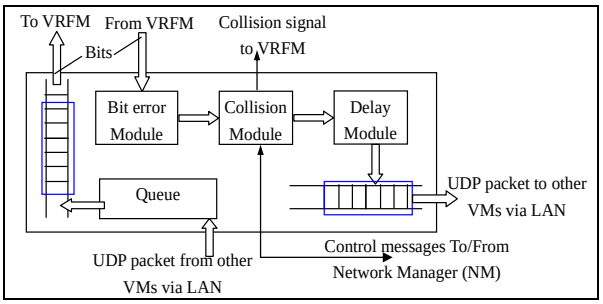


Figure 14. Virtual Channel

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 the 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 2 seconds. The 2 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.

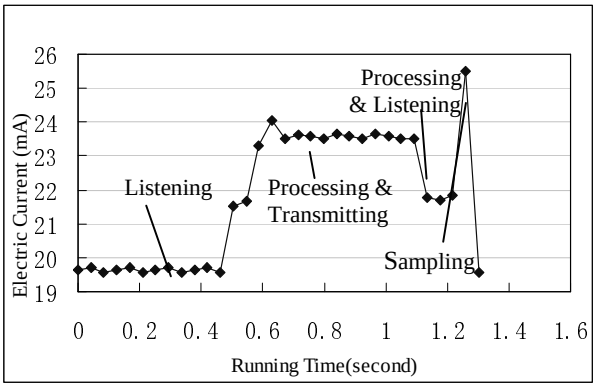


Figure 15: Measured Power Consumption of a MICA2 Mote

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%.

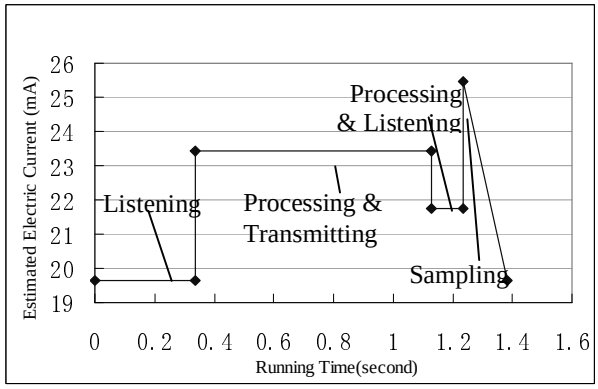


Figure 16: Estimated Power Consumption of a VM

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 with our modeling and emulation work, (2) conducting extensive, more 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 Grant 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.

A Framework for Extending the Synergy between MAC Layer and Query Optimization in Sensor Networks*

Vladimir I. Zadorozhny
Info Science & Tele Dept.
University of Pittsburgh
Pittsburgh, PA 15260
vladimir@sis.pitt.edu

Panos K. Chrysanthis
Computer Science Dept.
University of Pittsburgh
Pittsburgh, PA 15260
panos@cs.pitt.edu

Prashant Krishnamurthy
Info Science & Tele Dept.
University of Pittsburgh
Pittsburgh, PA 15260
prashant@sis.pitt.edu

Abstract

Queries in sensor networks are expected to produce results in a timely manner and for long periods, as needed. This implies that sensor queries need to be optimized with respect to both response time and energy consumption. With these requirements in mind, we develop novel cross-layer optimization techniques that utilize information about how the medium access control (MAC) layer operates while processing queries in large scale sensor network environments. The central framework of our approach is a Data Transmission Algebra that uniformly captures the structure of data transmissions along with their constraints and requirements. Our framework enables both qualitative analysis and quantitative cost-based optimization of sensor queries. We illustrate the effectiveness of our framework by developing a collision-aware scheduler and evaluating it experimentally.

1 Introduction

We are rapidly moving towards a world that is networked to an unprecedented scale where every device and appliance will have computing and communications capabilities and smart sensor networks will be deployed widely. A large part of the information infrastructure is evolving towards *large-scale wireless* sensor networks, e.g., information tracking systems such as airport security infrastructure, monitoring of children in metropolitan areas, product transition in warehouse networks, fine-grained weather measurements, etc. All of these tasks require efficient mechanisms for querying the sensor data and getting the result of the query in a timely manner. Typical sensor query execution maps into a tree-like data delivery pattern where a responding sensor node sends its data to a neighbor node which transmits it further to the next node towards the requesting node (the root). The data combined from all relevant sensors may be quite large and will require very high data transmission rates to satisfy time constraints. Meanwhile, limitations on sensor node resources like battery power imply that excessive transmissions in response to sensor queries can lead to premature network death.

Several techniques have been proposed to alleviate the problem of limited power at the network level such as energy-efficient routing, clustering and transmission scheduling [12, 25, 11, 6]. Sensor database research has also looked into sensor query processing strategies to minimize the query response time and reduce energy consumption that include sampling [16], prediction [10], approximation [5], and in-network query processing (or aggregation) [2, 15, 21]. With the same goal in mind, our research makes an effort to fuse the techniques and methods currently used in the two different areas of databases and networking. We believe that there is a natural convergence towards combining sensor query processing and lower layer network protocols that can systematically be explored in order to enable efficient operation of sensor networks.

In this paper we introduce an integrated approach to sensor query processing that utilizes performance and functional trade-offs between the query processing schemes, and the medium access control (MAC) layer. An examination of the reasons that affect both energy consumption and response time reveals that (a) data transmission *collisions* represent a major source of energy waste in wireless communication; (b) unnec-

*This research was partially supported by NSF awards IIS0219909, ANI-0123705, and a University of Pittsburgh CRDF award.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

essary amounts of *active time* for the sensors, due to lack of synchronization among data transmissions, is another major source of wasted energy in sensor networks; and (c) multi-rate data transmissions can have a considerable impact on the energy versus time tradeoff.

We propose a Data Transmission Algebra (DTA) that can capture the information about how the MAC layer operates while processing sensor queries. That is, the DTA can uniformly capture the structure of data transmissions, their constraints and their requirements. Our framework enables both qualitative analysis and quantitative cost-based optimization of sensor queries. Further, it allows the automatic generation and evaluation of alternative routing trees for a given set of queries and network configurations.

Using our framework, we have been able to develop novel *cross-layer optimization techniques*. An example of such an optimization discussed in this paper is *collision-aware query scheduling* that minimizes simultaneous transmissions that interfere with each other. As opposed to other schemes which assume that the MAC layer handles collisions in an appropriate manner, our collision-aware query scheduling reduce the amount of retransmissions and thus saves energy by explicitly considering data transmission collisions.

In realizing the DTA within an efficient query processor and optimizer, we are implementing a novel structure, *a pervasive catalog* that maintains highly available and accurate query statistics and other relevant network run-time information (i.e., meta-data). Such information includes current network topology, processing and transmission delays, collision domains, data rates, and current distribution of already aggregated and materialized data. We evaluate the effectiveness of our framework and the efficiency of the optimization algorithms experimentally.

In Section 2, we set the stage for our framework and overview closely related work. In Section 3, we introduce DTA and its application to cost-based query scheduling. In Section 4, we discuss the challenges in building a pervasive catalog infrastructure. We present the results of our experimental evaluation in Section 5 and discuss the applicability of our approach in Section 6.

2 Background and Related Work

Packet collisions are a major source of energy waste in wireless local communications [14]. Collisions occur when two or more nodes transmit at the same time in an area where both transmissions will have sufficient signal strength at the receiver node. When a collision occurs packets are corrupted and discarded unless there is some sort of capture [18].

Figure 1 elaborates on the concept of the *Collision Domain (CD)* in typical wireless systems such as IEEE 802.11. Assume that a sensor $n1$ wishes to initiate

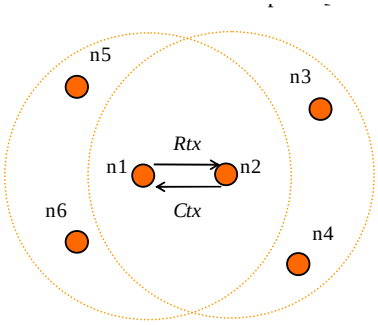


Figure 1: Collision domain of two communicating nodes

transmission to sensor $n2$. Initially, $n1$ sends a request for transmission (Rtx) (called request to send or RTS in 802.11) to $n2$. All other nodes in its transmission range ($n5$ and $n6$ in Figure 1) become aware of the request and remain silent until $n1$ ends the transmission to $n2$. The period of silence is based on virtual carrier sensing where information in the Rtx is used to determine how long they should back off. Note that sensors $n3$ and $n4$ do not sense the Rtx and could potentially transmit at the same time either to $n2$ or to each other resulting in collisions. To prevent this from happening, sensor $n2$ replies to $n1$ with a confirmation (Ctx) (called clear-to-send or CTS in 802.11). This time, the nodes in the transmission range of $n2$ ($n3$ and $n4$ in Figure 1) hear the Ctx and do not transmit until the end of the transmission from $n1$ to $n2$. In this scenario, the nodes $n3$, $n4$, $n5$, and $n6$ belong to the same collision domain. In general, any two communicating nodes ni and nj specify a *collision domain* CD(ni,nj) that can be defined as the union of transmission ranges of ni and nj.

Another way of eliminating collisions is to create an orthogonal transmission mechanism whereby a central authority, such as a base station allocates specific time slots for nodes to transmit based on reservation or polling [19] that will be similar to time division multiple access (TDMA). This however requires a centralized synchronization mechanism that could be fairly complex to implement, consume significant overhead for signaling and be difficult to implement in a multi-hop scenario. Although collisions, overhearing, and idle listening are major sources of energy waste in wireless multi-hop network, control traffic overhead is a significant factor in the energy consumption that should also be taken into account [24]. This can be achieved by efficient methods of wireless meta-data management [7, 26].

An important open research direction related to our work is developing intelligent cost-based strategies for switching nodes to sleep mode to minimize energy consumption [27, 22, 24, 4]. In [28] the authors proposed a cross-layer design for power management. The term "cross layer" here refers to a power manage-

ment layer utilizing knowledge about route setup and packet forwarding. In-network aggregation has also been proposed to save energy by reducing the amount of communications at the expense of extra computation [15, 23]. TAG [15] and Cougar [23] generate query routing trees in a way similar to what we consider in this paper. TiNA [21] is a middleware layer sitting on top of either TAG or Cougar. TiNA employs query semantics (and in particular, Quality of Data) and can reduce energy consumption significantly, by eliminating redundant data transmissions. However, none of these schemes considered data transmission collisions to reduce the amount of retransmissions and thus save energy. All of these schemes assume that the MAC layer handles collisions. Unlike TAG, Cougar, and techniques similar to TiNA or GaNC [3], our approach employs query and network metadata to generate query plans and routing trees that avoid collisions and maximize sleep time, while balancing response time and energy consumption.

3 Query Scheduling using DTA

We develop an algebraic framework that allows a sensor query optimizer to arrange concurrent data transmissions in the query tree so as to avoid collisions. The idea is that the query optimizer generates a schedule for data transmissions that is disseminated to each node in the query evaluation tree. As opposed to TDMA-like policies, the schedule is a suggested strategy that avoids collisions but it is up to individual node to decide how to behave within a set of constraint intervals specified by the schedule. In the event that a node cannot follow the schedule to avoid collisions, collisions are handled by the MAC layer. Thus, instead of delegating the collision resolution solely to the MAC layer, our framework utilizes query semantics to coordinate transmissions between sensor nodes.

3.1 Data Transmission Algebra

We define a Data Transmission Algebra (DTA) that efficiently enables such query scheduling. The DTA consists of a set of operations that take transmissions between wireless sensor nodes as input and produce a schedule of transmissions as the result. We call a one-hop transmission from sensor node ni to node nj an *elementary transmission* (denoted $ni \sim nj$). We also use a special symbol, *null*, that denotes a *completed* (or *empty*) transmission. Each transmission $nj \sim nj$, which is not empty is associated with a collision domain CD(ni, nj) as defined in Section 2. A transmission schedule is either an elementary transmission, or a composition of elementary transmissions using operations of the DTA as described below. The DTA includes three basic operations that combine two transmission schedules A and B:

1. ***order(A,B)*** $\equiv o(A, B)$. This is a strict order operation, that is, schedule A must be executed before B.

2. ***any(A,B)*** $\equiv a(A, B)$. This is an overlap operation that allows schedules A and B to be executed concurrently.

3. ***choice(A,B)*** $\equiv c(A, B)$. This is a non-strict order operation that either schedules A before B, or puts B before A. Thus, $c(A, B) \equiv (o(A, B) \vee o(B, A))$.

As an example of DTA operations consider the query tree in Figure 2 which was generated for some query Q. This shows an *initial* DTA specification that reflects the basic constraints of the query tree. The circles represent the ranges of the sensor nodes. For the purposes of this example, we assume that the transmission power is constant and the nodes are stationary. The initial specification consists of a set of strict order and overlap operations. For instance, operation $O1$ specifies that transmission $n2 \sim n1$ occurs after $n4 \sim n2$ is completed. This constraint reflects the query tree topology. Operation $A1$ specifies that $n4 \sim n2$ can be executed concurrently with $n6 \sim n3$, since neither $n3$ nor $n6$ belong to CD($n4$,$n2$), and neither $n4$ nor $n2$ are in CD($n6$,$n3$).

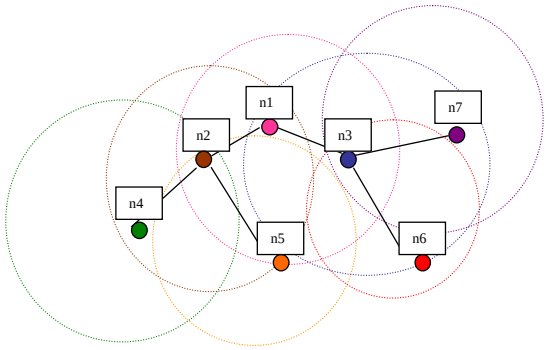


Figure 2: Query tree and initial DTA Specification

Each operation of the initial specifications defines a simple transmission schedule consisting of two elementary transmissions. The DTA introduces a set of transformation rules that can be used to generate more complex schedules from the initial specification. Figure 3 shows examples of DTA transformation rules $R1$-$R6$, and illustrates how these rules apply towards generating more complex schedules $A9$, $A10$ and $A11$ from the initial specification in Figure 2. $A9$ schedules

Example DTA transformation rules:

R1: o(A,B) ≠ o(B,A)
R2: a(A,B) = a(B,A)
R3: c(A,B) = c(A,B)
R4: a(A,B) & a(A,C) = a(A, c (B,C))
R5: c(A, c(B,C)) & o(A,B) = c(o(A,B), C)
R6: c(c(B,C), A) & o(B,A) & o(C,A) = o(c(B,C), A)

Example of DTA transformations:

A1,A2,R4 imply:
A9: a(n4-n2, c(n6~n3, n7~n3));
A3, A9, R4 imply:
A10: a(n4-n2, c(c(n6~n3, n7~n3), n3~n1));
A10,O3,O4,R6 imply:
A11: a(n4-n2, o(c(n6~n3, n7~n3), n3~n1));

Figure 3: Example of DTA transformations

schedule	cost
ni~nj	Tp(ni)+Ttx(ni~nj)+Tp(nj)
o(A,B)	cost(A)+cost(B)
a(A,B)	max(cost(A),cost(B))
c(A,B)	cost(A)+cost(B) – Tf

Figure 4: Estimating costs of schedules

three elementary transmissions, while each of $A10$ and $A11$ schedules four elementary transmissions.

None of the simple or complex transmission schedules considered so far include all elementary transmissions of the query tree, so we call them *partial schedules*. Our goal is to generate DTA expressions for *complete schedules*. A complete schedule includes all elementary transmissions of the query tree. Below we introduce a cost model for optimizing data transmissions in order to generate complete and efficient schedules.

Figure 4 shows simple cost estimation expressions for each of the DTA expressions. In this case, the cost corresponds to the execution time associated with a particular schedule. For clarity of presentation we ignore energy consumption at this point. For example, the execution time of elementary transmission $ni \sim nj$ consists of local processing times Tp at nodes ni and nj plus the time Ttx required for transmitting data from ni to nj.

The execution time of strict order of schedules A and B is the sum of execution times of A and B. For overlapping schedules A and B, the execution time would be the maximum of the execution times of A and B. Finally, the execution time of the choice between A and B is the same as the execution time of the strict order minus a predefined time factor Tf. Tf indicates that in general, the optimizer prefers the choice operation over strict order, since the latter restricts flexibility of the optimizer in query scheduling. We ignore propagation times as they are negligible in this case.

3.2 Scalable DTA Scheduling

Basic DTA scheduling may be expensive due to its combinatorial nature. The number of alternative schedules grows at least exponentially with the number of sensor nodes and elementary transmissions participating in a query. In order to decrease this complexity, we developed heuristic-based pruning methods that eliminate suboptimal alternatives. We also explored randomized algorithms to cope with the expected complexity of queries in large scale sensor networks. Randomized algorithms [13] are scalable techniques to solve complex combinatorial optimization problems that search for a solution in a large space of all possible solutions. Each solution is associated with application-specific costs. Randomized algorithms will search for a solution with the minimal cost by performing random walks in the solution space via a series of valid moves. In our case possible solutions are DTA schedules.

M1.	Choice commutativity	c(X,Y) ↔ c(Y,X)
M2.	Overlap commutativity	a(X,Y) ↔ a(Y,X)
M3.	Choice associativity	c(c(X,Y),Z) ↔ c(X,c(Y,Z))
M4.	Overlap associativity	a(a(X,Y),Z) ↔ a(X,a(Y,Z))
M5.	Order associativity	o(o(X,Y),Z) ↔ o(X,o(Y,Z))
M6.	A/C exchange	a(X,c(Y,Z)) → c(a(X,Y),Z)
M7.	Left A/O exchange	a(X,o(Y,Z)) → o(a(X,Y),Z)
M8.	Right A/O exchange	a(X,o(Z,Y)) → o(Z, a(X,Y)
M9.	C/A exchange	c(a(X,Y),Z) → a(X,c(Y,Z)), provided ***any(X,Z)*** holds
M10.	Left O/A exchange	o(a(X,Y),Z) → a(X,o(Y,Z)), provided ***any(X,Z)*** holds
M11.	Right O/A Exchange	o(Z, a(X,Y) → a(X,o(Z,Y)), provided ***any(X,Z)*** holds

Figure 5: Valid moves between DTA Schedules

Figure 5 represents valid moves between DTA schedules. Here *any(S1,S2)* is relation between two DTA schedules $S1$ and $S2$ defined recursively as follows:

any(X,Y) if $a(X,Y)$ or $a(Y,X)$.
any(X,a(Y,Z)) if *any(X,Y)* and *any(X,Z)*.
any(X,c(Y,Z)) if *any(X,Y)* and *any(X,Z)*.
any(X,o(Y,Z)) if *any(X,Y)* and *any(X,Z)*.

Different randomized algorithms employ different moving strategies and stopping conditions. Some of the most well-known randomized optimization algorithms are Iterative Improvement (II), Simulated Annealing and Two-Phase Optimization [13]. We explore performance of each of them for the purpose of scalable DTA scheduling. In Figure 6, we illustrate how DTA scheduling can utilize II algorithm.

```
Procedure II () {
minS = Sser ;
  while (not stopping_condition) do {
    S = random DTA schedule
    while (local_minimum(S)) do {
      S' = random DTA schedule
           in neighbors(S)
      if cost(S') < cost(S) then S=S'
    }
    if cost(S) < cost(minS) then minS=S
  }
return(minS)
}
```

Explanation of variables and parameters:

- ***minS*** – current DTA schedule with minimal cost;
- ***Sser*** – random serial DTA schedule;
- ***S*** – random initial DTA schedule;
- ***neighbors(S)*** – a set of schedules that can be generated from S via one valid move;
- ***stopping_condition*** – number of considered initial schedules;
- ***local_minimum(S)*** – a number of neighbors of S to be tested, of which none has lower cost than S. If the test is successful, S is considered to be a local minimum

Figure 6: II Algorithm for DTA Scheduling

3.3 Impact of multi-rate transmissions

Multi-rate transmission is supported in the new generation of standards for wireless local communications (such as 802.11a/b/g) as well as in evolving future technologies. Under these standards, it is possible for nodes to transmit at different data rates depending on signal quality. Usually, signal quality degrades with distance (although this is not the only reason) [18]. The path loss (that is dependent on the environment and frequency), the modulation scheme, the transmission power and the receiver sensitivity influence the data rates that can be provided for a given quality (bit error rate or packet error rate). For instance, consider phase shift keying (PSK) based modulation schemes. In the case of PSK, the number of bits/symbol will affect the bit error rate. Consider binary PSK (BPSK), quaternary PSK (QPSK), 8-PSK and 16-PSK that transmit 1, 2, 3 and 4 bits per symbol respectively. The energy per bit to the noise power spectral density ratios required by these modulation schemes to achieve a bit error rate of 10^{-5} are respectively 10, 10, 13.5 and 18 dB [20]. Note also that compared to BPSK, QPSK, 8-PSK and 16-PSK can transmit data at 2, 3 and 4 times higher rates in the same bandwidth. For actual products based on 802.11, similar properties apply. Assuming a constant standard transmission power, an 802.11 based node may be able to transmit data at 11 Mbps to another node that is 90 ft away, but only at 5.5 Mbps to another node that is 150 ft away[1] or 2 Mbps to a node that is 210 ft away using 802.11b technology. If the transmission power is increased or the environment is open space, the range of transmission at 11 Mbps could be increased. In outdoor areas, the distances up to which certain data rates can be achieved will be different. For example, a data rate of 11 Mbps can be achieved if the nodes are separated by 200m, 5.5 Mbps if the separation is between 200 and 300 m, and 2 Mbps if it is between 300 and 600m[2].

Alternatively, by reducing the transmission power, the range can be reduced while keeping the data rate at say 2 Mbps. Reducing the range also implies that the collision domain is shrunk allowing the possibility of concurrent transmissions between different sensor nodes. This brings up interesting opportunities for creating minimal cost query schedules. Our query optimizer estimates the transmission power, data rates, and order of transmission of sensor nodes that minimizes costs in multiple ways. We discuss such scenarios next.

Certain sensor nodes may be low on battery power and if this information is known, it would be advantageous to reduce their transmission power and range to prolong the network life. There may be sensor nodes that have sufficient energy and could increase their transmission power for a certain period of time to bypass some hops and directly reach the node that initiated the query. In this case the DTA will utilize a cost model that takes into account both response time and energy consumption while trading certain degree of concurrency (i.e., number of operations/transmissions that can overlap in the initial specification) for increasing the speed of some transmissions.

Figure 7 illustrates this idea with two simple transmission scenarios. In scenario **(a)**, transmissions $n4 \sim n2$ and $n5 \sim n3$ can occur concurrently, which is reflected by the overlap operations $A1$ in the corresponding initial DTA specification. By increasing transmission power of sensor $n4$ (scenario **(b)**), the opportunity of transmitting $n4 \sim n2$ and $n5 \sim n3$ concurrently disappears, which results in a more restricted DTA specification. However, the gain in $n4 \sim n2$ transmission speed, as well as a possibility for $n4$ to transmit directly to $n1$ can overcome the lack of concurrency in scenario **(b)** under certain circumstances. Apparently, in this case $n4$ would spend more energy to complete its transmission. We are extending the DTA cost model to capture the tradeoffs between transmission speed, transmission power and degrees of concurrency in sensor query processing. Assuming general modulation schemes and suitable ranges of transmit powers we plan to compare the results with measurements with real products like 802.11 and Bluetooth.

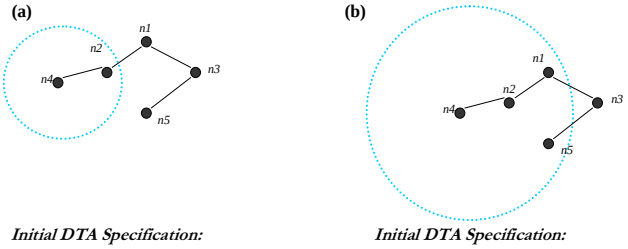


Figure 7: Explanation of the tradeoff between power, speed and concurrency

[1] These numbers are based on measurements in indoor areas by Atheros [1].

[2] These numbers are based on product information by Firetide [9].

4 Pervasive Catalog for DTA Query Scheduling

In order to support DTA query scheduling the optimizer should rely upon highly available and accurate query statistics and other relevant network metadata including current network topology, processing and transmission delays, collision domains and current distribution of pre-aggregated and materialized data. Such query statistics and network meta-data should be stored in a highly available distributed repository with varying freshness, precision and availability requirements. Design and implementation of such a repository together with an appropriate signaling system is a considerable challenge. In this section we report our on-going research on designing a *pervasive catalog system (PCat)* that implements such a meta-data repository.

We are considering three basic catalog implementation alternatives: (1) *centralized scheme*, where all the statistics metadata is maintained in a central node accessible through a base station (2) *distributed scheme*, where each node maintains its own metadata statistics, and (3) *hybrid scheme*, where some sensor nodes maintain their own statistics and host statistics about other nodes and sub-networks.

Centralized Scheme. In a centralized scheme, the root node is a base station (BS) with a large broadcast area and unlimited power supply since it is presumably a fixed node and located in an opportunistic location. The BS maintains the statistics on processing and transmission delays, the network topology, and collision domains. The synchronization of the participating nodes can be easily achieved, since every node listens to the same BS. The BS performs query scheduling using DTA and broadcasts the resulting schedule to every node in the network. For this purpose, out-of band signaling or periodic beacons can be employed. Note that sensor nodes need to only receive this information, but need not transmit information directly to the BS as this may require large transmit powers and incur large energy consumption.

Distributed Scheme. In this scheme, each wireless node maintains statistics meta-data about itself. We consider only local sensor processing times (Tp), and the transmission time to a parent node (Ttx). A query can be submitted at a root node of a routing tree and then it can propagate down the tree to every node. After receiving a query, each child node in the lowest level provides its statistics, i.e., processing and transmission times (delays) to their parent (Figure 8 - top). Then, the parent node performs query scheduling for each child node using the DTA in order to minimize collisions and the active time for the parent's receiver. The parent node returns this schedule to its children (Figure 8 - bottom). After scheduling its children, the parent node estimates and sends its own processing and transmission delay information to an upper level parent node. Then the same process propagates up the routing tree until it reaches the root node. The above process can vary depending on actual query and network statistics. For example, the transmission time of the latest node can be fixed and transmissions for the remaining nodes should be scheduled ahead of the latest node.

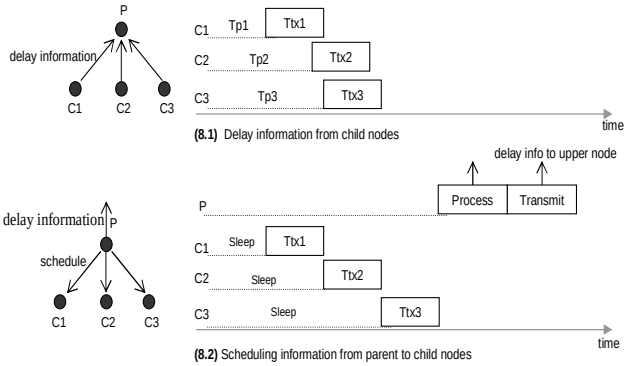


Figure 8: Distributed Query Scheduling

Hybrid Scheme. Under the hybrid scheme, every node in the sensor network is associated with its own statistics metadata, and some of the nodes can additionally host statistics meta-data (perhaps more summarized) about a subnet of devices in their local meta-data repository. Hybrid PCat implements adaptive distribution granularity that minimizes control and meta-data traffic, as well as energy consumption while providing certain level of meta-data accuracy and freshness. It can be tuned for either maximum lookup or update performance and levels in between. In this way PCat is implementing different tradeoffs between data availability, freshness and precision, ranging from purely distributed schemes to a purely centralized scheme.

5 Experiments and Analysis

In this section, we discuss the first results of the evaluation of our framework. First, we show the potential performance gains of DTA schedules. These are generated by a basic DTA scheduler that enumerates all possible schedules exhaustively. Second, given that such a DTA scheduler does not scale, we evaluated the performance of an Iterative Improvement (II) algorithm for DTA scheduling that is capable of handling large query trees. Finally, we compared DTA scheduling with 802.11 MAC in order to put our results in a better perspective.

5.1 Behavior of the DTA schedules.

In order to evaluate our approach, we implemented a basic DTA scheduler in Arity Prolog. Here, we report

on the behavior of the DTA scheduler for a medium complexity query tree involving ten sensor nodes with overlapping collision domains. Processing and transmission costs were generated randomly using Gaussian distributions.

The basic DTA scheduler generated schedules stage by stage starting from initial schedules with two elementary transmissions (stage 1). Stage 2, 3 and 4 represent schedules with 3, 4 and 5 scheduled transmissions. Stage 5 includes complete schedules covering all elementary transmissions of the query tree.

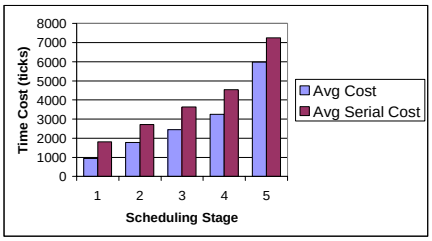


Figure 9: Comparison of DTA scheduling with serial scheduling

Figure 9 shows the average query execution time for different scheduling stages. We compare the DTA scheduling with a *serial scheduling strategy* that performs elementary transmissions sequentially. For each scheduling stage we report the average execution time of all its schedules. We observe that at each scheduling stage, the approach that uses DTA considerably outperforms serial scheduling.

Figure 10 reports on the average benefit that each scheduling stages gains from concurrent transmissions. Intuitively, the benefit is part of the time cost that the DTA scheduler is able to "hide" scheduling some transmissions concurrently. The benefit is defined recursively for each of DTA operations. The benefit of a(X,Y) is equal to minimum of costs cost(X) and cost(Y). For the rest of the DTA operations the benefit is equal to zero. Thus, any serial schedule has a zero benefit.

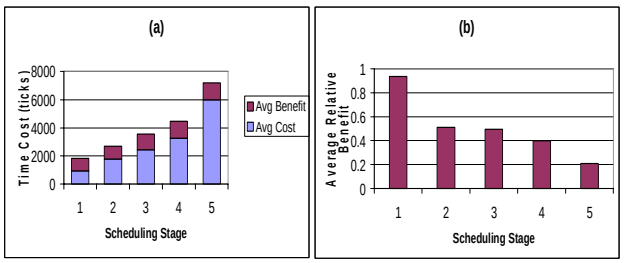


Figure 10: Time cost (a) and relative benefit (b) of DTA scheduling

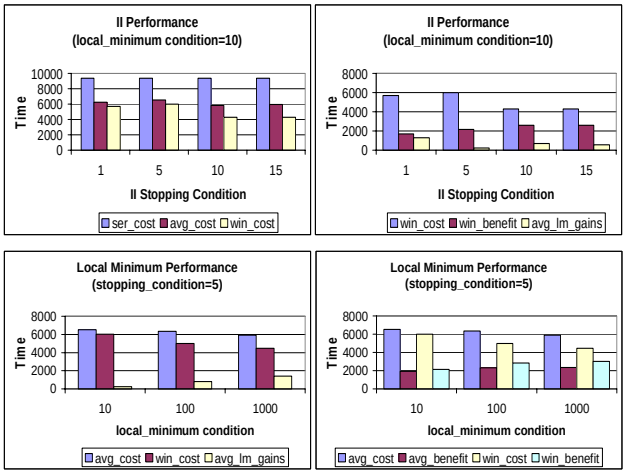


Figure 11: Performance of II-based DTA Scheduler

Figure 10(a) compares values of average time cost and average benefit for each scheduling stage. With the increase of the number of transmissions the benefit grows, but not as fast as the time cost. Figure 10(b) plots the average relative benefit as a percentage of the overall average time cost per scheduling stage. We observe that for simple initial concurrent schedules the benefit is almost equal to the time cost. This is an expected behavior. Elementary transmissions have comparable time costs. By scheduling them concurrently, DTA hides on average one half of the time cost of their serial execution. However, for complete schedules (stage 5) the average relative benefit is as low as 0.2, which means that only 20% of the total serial cost has been hidden. This is also an expected behavior, since complete schedules are composed of non-elementary transmissions (sub-schedules) with higher variance in their time cost. Thus, it is more challenging for the DTA scheduler to hide time costs of non-elementary sub-schedules.

5.2 Evaluation of the II-based DTA Scheduler

Figure 11 shows some of our experiments that evaluated the performance of the Iterative Improvement (II) algorithm for DTA scheduling. It reports average time cost and benefit of all considered schedules (*avg_cost* and *avg_benefit*) and time cost and benefit of the winner schedule chosen by II algorithm (*win_cost* and *win_benefit*). In addition to costs and benefits of the schedules, we also report a value of average gain received from the local minimum phase of the algorithm (*avg_lm_gains*). The local minimum gain occurs when II algorithm improves a random initial schedule via given number of random moves. This number should be no greater than the local minimum condition.

The upper left graph in Figure 11 illustrates a consistent improvement of II performance as we increase the values of the stopping condition with fixed local minimum condition of 10. We also provide a time cost

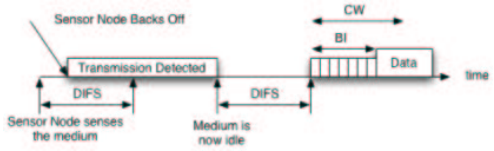


Figure 12: Back-off in 802.11 MAC

of a serial schedule (*ser_cost*) as a reference point and a worst case scenario.

The upper right graph also reports on benefit and local minimum gains of the winner schedule. While we observe steady increase of the benefit value, the local minimum gain behaves quite sporadically. This is an expected behavior, since for each value of II stopping condition we set the same local minimum condition. Thus, in general we should expect a random value of *avg_lm_gains*.

In order to explore the performance of the local mimimum phase we plot the cost, benefits and local minimum gains for different values of the local minimum conditions (lower two graphs of the Figure 11). We observe that the performance of the II algorithm consistently improves as we increase the values of the the local minimum conditions.

In summary, our experiments showed that II algorithm scales well for large query trees and demonstrates reasonable performance with proper parameter settings.

5.3 Comparison of DTA scheduling with 802.11 MAC

We note that 802.11-like transmissions may be faster than simple serial schedules considered above under lightly loaded conditions, but would still be slower than DTA. For example, in Figure 2, let us assume that the MAC layer independently operates and the query optimizer creates no schedule. For this topology, there could be concurrent transmissions $n4 \sim n2$ or $n5 \sim n2$ and $n6 \sim n3$ or $n7 \sim n3$. However, there is no guarantee which of these will occur first. Consider the contention between $n4 \sim n2$ and $n5 \sim n2$. Suppose the medium is idle and both $n4$ and $n5$ sense it as idle at the same time upon receiving the query. They will both wait for a time called distributed inter-frame space (DIFS) and transmit the packet simultaneously, resulting in a collision. If they sense the channel at slightly different times, one of the nodes will transmit first resulting in the second node backing off as shown in Figure 12.

Suppose node $n4$ was able to transmit first. Node $n5$ will back-off and wait till node $n4$ completes its transmission. After node $n4$ completes its transmission, $n5$ will wait for an additional time equal to DIFS and anywhere between 1 and 7 slots each of duration 20 μs before it attempts transmission. The number of slots (called the back-off interval - BI) will be selected randomly in a window (called the contention window - CW). In case there is a collision, the CW is doubled. This doubling occurs each time there is a collision (resulting in up to an increase of 1024 times). If there are several sensor nodes in the same collision domain, that need to transmit data, the process would result in some collisions and considerable additional waiting time. A similar scenario happens between nodes $n6$ and $n7$. The number of collisions would also depend upon network topology and the type of queries (how large the traffic will be at given points in the sensor network).

We believe DTA scheduling would reduce collisions and improve the energy savings. Collisions result in completely wasted energy. In addition, during the backoff slots, sensor nodes will be continuously monitoring the medium resulting in wasted energy consumption. We have also ignored the acknowledgment process at the MAC layer in this preliminary analysis. Currently we are implementing simulations in OPNET Modeler [17] to test the degree of time and energy savings that DTA would provide over regular 802.11-like transmissions

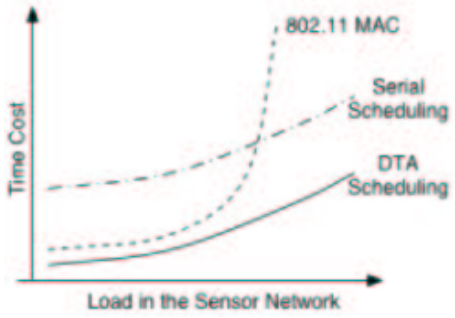


Figure 13: Time Costs with different scheduling schemes

Figure 13 represents the expected relationship between 802.11-MAC, serial and DTA-based transmissions. As discussed above 802.11-like transmissions may be faster than simple serial schedules considered above under lightly loaded conditions, but would still be slower than DTA. For higher network loads and more complex sensor queries the performance of 802.11-MAC considerably degrades comparing to serial and DTA scheduling. DTA will always outperform serial scheduling. Our preliminary simulation results support this assumption. Currently we are undertaking a comprehensive study of different query scheduling options.

6 Discussion on DTA Applicability

In this paper, we use the IEEE 802.11 standard as the basis for the medium access control mechanism as we

are considering large scale sensor networks that may need to transmit large amounts of data over fairly long distances. For lower data rates (on the order of a few kbps) and smaller ranges, a more suitable mechanism is the newly proposed IEEE 802.15.4 standard [29] for low-rate wireless personal area networks. We note that this mechanism also employs CSMA/CA for medium access although the details are different.

In explaining the DTA and in the simulations, we use a circular coverage area for each node. In reality, the radio propagation conditions determine the shape of the coverage area and this will be irregular. We do note that circular coverage areas are commonly used as approximations and also for mathematical tractability. They do provide us with insights as to how a proposed mechanism may perform. Moreover, the DTA does not depend on the shape of the collision domains, but rather on the knowledge of what transmissions from what nodes are likely to collide. For this, it is sufficient if the interference characteristics of sensor nodes are known a priori. In a fixed topology with a small number of nodes, it is easy to determine such characteristics and obtain knowledge of the collision domains. In a dense network, this could be a problem. While we do not address this problem, there have been research attempts to provide location information of sensor nodes. For routing purposes, nodes need to determine what their neighbors are and the number of hops required to reach a sensor node can provide us with equivalent information.

Finally, it is worth pointing out that our framework is not limited to tree-like data patterns, but is also capable of capturing broader data dissemination paradigms such as *wave scheduling* [8].

7 Conclusions

We introduced a novel algebraic framework for specifying and analyzing data transmissions along with constraints imposed by a query in wireless sensor networks. Our framework enables flexible cross-layer query optimization techniques that utilize information about the MAC layer. The query optimization results in reduction in energy consumption, which increases the lifetime and effectiveness of the network, to produce the expected Quality of Data in a timely manner. We also introduced the necessary infrastructure, a pervasive catalog that provides our framework with highly available and accurate query statistics and relevant network meta-data.

Currently we are undertaking a comprehensive experimental and theoretical study of our framework. It includes the implementation and testing of our framework in simulated and real-world settings, as well as exploring its completeness and complexity characteristics.

Acknowledgments

We would like to thank our students Mohamed Sharaf, Divyasheel Sharma and Chih-kuang Lin as well as the anonymous reviewers for their thoughtful comments. Special thanks to Alexandros Labrinidis whose constructive and insightful suggestions significantly improved the DTA framework.

References

[1] Atheros Communications. *Whitepaper: 802.11 Wireless LAN Performance.* (available at http://atheros.com/), April 2003.

[2] P. Bonnet, J. Gehrke and P. Seshadri. Towards Sensor Database Systems. *Proc. of MDM Conf.*, 2001

[3] J. Beaver, M. A. Sharaf, A. Labrinidis, and P. K. Chrysanthis. Location-Aware Routing for Data Aggregation for Sensor Networks. *Proc. of Geo Sensor Networks Workshop*, 2003

[4] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. SPAN: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. Proc. of ACM MobiCom Conf., 2001

[5] J. Considine, F. Li, G. Kollios and J. Byers. Approximate Aggregation Techniques for Sensor Databases. *Proc. of IEEE ICDE Conf.*, 2004

[6] U. Cetintemel, A. Flinders, Y. Sun. Power-Efficient Data Dissemination in Wireless Sensor Networks. *Proc. of ACM MobiDE Workshop*, 2003

[7] P. K. Chrysanthis and V. Zadorozhny. From Location Databases to Pervasive Catalog. *Proc. of MDDS Workshop*, 2002

[8] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni and Y. Yao. Energy-Efficient Data Management for Sensor Networks: A Work-In-Progress Report. *Proc. of 2nd IEEE Upstate New York Workshop on Sensor Networks*, 2003.

[9] Firetide Inc. *Specifications of the HotPoint 1000S Wireless Mesh Router, Datasheet.* (available at: http://www.firetide.com/images/User_FilesImages/documents/HP1000S_DS_a104.pdf)

[10] S. Goel and T. Imielinski. Prediction-based monitoring in sensor networks: Taking lessons from MPEG. *Computer Comm. Review*, 31(5), 2001.

[11] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. *Proc. of HICSS Conf.*, 2000

[12] J. Heidemann, F. Silva, C. Intanagonwiwat, R.Govindan, D. Estrin and D. Ganesan. Building efficient wireless sensor networks with low-level naming. Proc. of ACM SOSP, 2001

[13] Y. E. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. *Proc. of ACM SIGMOD Conf.*, 1990

[14] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen. A Survey of Energy Efficient Network Protocols for Wireless Networks. *Wireless Networks*, 7(4), 2001

[15] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. TAG: A tiny aggregation service for ad hoc sensor networks. *Proc. of OSDI*, 2002

[16] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks. *Proc. of ACM SIGMOD Conf.*, 2003

[17] www.opnet.com

[18] K. Pahlavan and A. Levesque. *Wireless Information Networks*. John Wiley and Sons, 1995

[19] K. Pahlavan and P. Krishnamurthy. *Principles of Wireless Networks: A Unified Approach*. Prentice Hall, 2002

[20] J. Proakis. *Digital Communications*. McGraw Hill, 2001

[21] M. A. Sharaf, J. Beaver, A. Labrinidis, and P. K. Chrysanthis. TiNA: A Scheme for Temporal Coherency-Aware in-Network Aggregation. *Proc. of ACM MobiDE Workshop*, 2003

[22] C. Schurgers, V. Tsiatsis and M. Srivastava. STEM: Topology Management for Energy Efficient Sensor Network. *Prov. of IEEE Aerospace Conf.*, 2002

[23] Y.Yao and J.E. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3), 2002

[24] W. Ye, J. Heidemann and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. *Proc. of IEEE INFOCOM*, 2002

[25] M. Younis, M. Youssef and K. Arisha. Energy-aware routing in cluster-based sensor networks. *Proc. of MASCOTS*, 2002

[26] V. Zadorozhny and P. K. Chrysanthis. Location-Based Computing. In *Telegeoinformatics: Location-Based Computing and Services*, Taylor and Francis Books, 2003

[27] R. Zheng, J. Hou and L. Sha. Asynchronous Wakeup for Ad Hoc Networks: Theory and Protocol Design. *Proc. of ACM MobiHoc*, 2003

[28] R. Zheng and R. Kravets. On-demand Power Management for Ad-Hoc Networks. *Proc. of IEEE INFOCOM Conf.*, 2003

[29] IEEE Std 802.15.4. *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, October 2003

Region Streams: Functional Macroprogramming for Sensor Networks

Ryan Newton Matt Welsh

MIT & Harvard
Cambridge, MA
U.S.A.
newton@mit.edu & mdw@eecs.harvard.edu

Abstract

Sensor networks present a number of novel programming challenges for application developers. Their inherent limitations of computational power, communication bandwidth, and energy demand new approaches to programming that shield the developer from low-level details of resource management, concurrency, and in-network processing. We argue that sensor networks should be programmed at the global level, allowing the compiler to automatically generate nodal behaviors from a high-level specification of the network's global behavior.

This paper presents the design of a functional macroprogramming language for sensor networks, called Regiment. The essential data model in Regiment is based on region streams, which represent spatially distributed, time-varying collections of node state. A region stream might represent the set of sensor values across all nodes in an area or the aggregation of sensor values within that area. Regiment is a purely functional language, which gives the compiler considerable leeway in terms of realizing region stream operations across sensor nodes and exploiting redundancy within the network.

We describe the initial design and implementation of Regiment, including a compiler that transforms a macroprogram into an efficient nodal program based on a token machine. We present a progression of simple programs that illustrate the power of Regiment to succinctly represent robust, adaptive sensor network applications.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

1 Introduction

A sensor network represents a complex, volatile, resource-constrained cloud of sensors capable of collaborative sensing and computing. Programming such an entity requires new approaches to managing energy usage, performing distributed computation, and realizing robust behavior despite message and node loss.

One approach is to program the sensor network as a whole, rather than writing low-level software to drive individual nodes. Not only does such an approach raise the level of abstraction for developing novel programs, we argue that the only way to address the complexity of the underlying substrate is through automatic compilation from a high-level language. Today, few computer scientists would doubt the value of high-level languages for programming individual computers, or even groups of machines connected in a traditional network. We wish to take this approach to the next level and provide a *macroprogramming* environment for a network of sensors that automates the process of decomposing global programs into complex local behaviors.

This paper presents a functional macroprogramming language for sensor networks, called *Regiment*. The essential data model in Regiment is based on *region streams*, which represent spatially distributed, time-varying collections of node state. The programmer uses these to express interest in a group of nodes with some geographic, logical, or topological relationship, such as all nodes within k radio hops of some *anchor* node. The corresponding region stream represents the set of sensor values across the nodes in question. The operations permitted on region streams include *fold*, which aggregates values across nodes in the region to a particular anchor, and *map*, which applies a function over all values within a single region stream. Operationally, map requires no communication between elements, whereas fold requires the collapse of data to a single physical point.

Regiment is a *purely functional language* that does not permit input, output, or direct manipulation of program state. Regiment uses monads [19] to indirectly deal time-

varying values. As in other functional language designs, this approach gives the compiler considerable leeway in terms of realizing region stream operations across sensor nodes and exploiting redundancy within the network. The Regiment compiler transforms a network-wide macroprogram into an efficient nodal program based on a *token machine*. A token machine is a simple distributed state machine model in which nodes perform local sensing and computation in response to the arrival of named tokens, which may be received as radio messages or generated internally.

2 Related Work

We use the term macroprogramming to refer to programming the sensor network as a whole, rather than at the level of individual nodes. We argue that programming at this level leads to more concise and robust programs, since global behavior is specified directly. As an intuition, consider that matrix multiply algorithms are far simpler to state in terms of matrices and vectors than as parallel programs implemented in MPI.

For sensor networks, progress in macroprogramming has largely been domain specific. We have seen: languages for global-to-local compilation of spatial pattern formation [21, 17, 8]; Envirotrack [1], which exposes tracked objects as language *objects* (analogous to the way we expose regions); and, of course, database systems for querying sensor data [32, 18].

2.1 Middleware

There have been many attempts to design programming paradigms or run-time services to make application programming for sensor networks easier. These need not necessarily take a "macro" approach. In fact, many of these middleware developments are complementary to macroprogramming, and perhaps usable by a macroprogramming compiler backend. Spatial Programming [7] uses Smart Messages to provide content-based spatial references to embedded resources. For example, the programmer may refer to the first available camera in a given (predefined) spatial region. Other communication abstractions include GHT [25], DIFS [12], SPIN [13], DIMENSIONS [11], and HOOD [31]. Regiment draws on the Abstract Regions [30] model, which provides efficient communication primitives within *local regions* of the network.

2.2 Amorphous Computing

The Amorphous Computing research effort has pursued the broad goal of engineering aggregate behaviors for dense ad-hoc networks (paintable computers, Turing substrates). Their work focuses on pattern formation, taking inspiration from developmental biology. They demonstrate how to form coordinate systems [20], arbitrary two and three dimensional shapes [17], arbitrary graphs of "wires" [8], and origami-like folding patterns [21]. Yet the Amorphous Computing effort has not to date provided a model for *programming* rather than *pattern formation*. In addition, the target platforms envisioned by the Amorphous Computing effort differ significantly from existing wireless sensor networks.

2.3 Database approaches

The database community has long taken the view that declarative programming through a query language provides the right level of abstraction for accessing, filtering, and processing relational data. Recently, query languages have been applied to sensor networks, including TinyDB [18], Cougar [32], and IrisNet [22]. While these systems provide a valuable interface for efficient data collection, they do not focus on providing general-purpose distributed computation within a sensor network. For example, it is cumbersome to implement arbitrary aggregation and filtering operators and arbitrary communication patterns using such a query language. We argue that a more general language is required to fully realize the potential for global network programming.

There has also been a body of work on extending programming languages to deal with database access: database programming languages or DBPLs. Many types of languages have been used in this work, including functional ones. Functional DBPLs include FAD [5] and IPL [2]. Regiment differs from these languages in being explicitely concerned with: distributed processing, spatial processing, streaming data, and with the volatility of its substrate—sensor networks.

2.4 Stream processing languages

Stream processing is an old subject in the study of programming languages. Functional Reactive Programming (FRP) is a recent formulation which uses modern programming language technology (including monads [19] and type classes [28]) to allow purely functional languages to be able to deal comfortably with real time events and time-varying streams. FRP is the inspiration for Regiment's basic type system.

Regiment's problem domain also overlaps with recent work in extending databases to deal with continuous queries over streaming data, such as STREAM [3], Aurora [33], and Medusa [33]. Regiment aims to utilize many optimization techniques developed in this body of work, but at the same time Regiment occupies a slightly different niche—it is not only intrinsically *distributed* (on a volatile substrate) but explicitely *spatial*.

3 The functional macroprogramming approach

The traditional method of programming sensor networks is to write a low-level program that is compiled and installed in each individual sensor. This amounts to a programming model consisting of access to sensor data on the

local node, coupled with a message-passing interface to radio neighbors. In contrast, our macroprogramming model captures the entirety of the sensor network state as a global data structure. The changing state of each sensor originates a stream of data at some point in space. Collectively they form a global data structure.

To express sensing and communication within local groups of nodes, region streams encapsulate subsets of the global network state that can be manipulated by the programmer as single units. They represent the time-varying state of a time-varying group of nodes with some geographic or topological relationship. Communication patterns for data sharing and aggregation can be efficiently implemented within such local regions [30, 31].

3.1 Why a functional language?

We propose that functional languages are intrinsically more compatible with distributed implementation over volatile substrates than are imperative languages. Prominent (call-by-value) functional languages include Lisp, Scheme and OCaml. Functional languages have been used to explore high-level programming for parallel machines—such as NESL [6] and *LISP [26]—and for distributed machines [24]. In our system, we get the most benefit from restricting ourselves to a *purely* functional (effect free), call-by-need language similar to Haskell [16].

Purely functional languages essentially hide the direct manipulation of program state from the programmer. In particular, the program cannot directly modify the value of variables; rather, all operations must be represented as *functions*. Monads [19] allow mutable state to be represented in a purely functional form. For sensor network applications, abstracting away the manipulation of state allows the compiler to determine how and where program state is stored on the volatile mesh of sensor nodes. For example, to store a piece of data reliably, it may be necessary to replicate it across multiple nodes in some consistent fashion. Using a functional language makes consistency moot; immutable values can be freely replicated and cached.

Because functions are deterministic and produce no output, computation can be readily migrated or replicated without affecting program semantics. Another way to state this is that functional programs support *equational reasoning*. Program optimization in such a framework can be cast as semantics-preserving application of general program transformations [23].

Regiment has a host of algebraic properties which can be used together with a static cost model or dynamic profiling information to optimize performance and resource usage.

Another advantage of the functional programs is that it is straightforward to extract parallelism from their manipulation of data. For example, a function that combines data streams from multiple sensors can be compiled into a form that efficiently aggregates each data stream within the network. In addition to such *data parallel* operations, functional programs are implicitly parallel in their evaluation of function arguments [4]. The compiler can automatically extract this parallelism and implement it in a variety of ways, distributing operations across different sensor nodes.

4 The Regiment language

The goal of Regiment is to write complex sensor network applications with just a few lines of code. In this section we describe the Regiment language through several examples. A common application driver for complex coordination within sensor networks is that of tracking moving vehicles through a field of sensors each equipped with a proximity sensor of some kind (e.g., a magnetometer). We start by showing a simple Regiment program that returns a series of locations tracking a single vehicle moving through such a network.

```
let aboveThresh (p,x) = p > threshold
    read node =
       (read_sensor PROXIMITY node,
        get_location node)
in centroid (afilter aboveThresh
                     (amap read world))
```

We use a syntax similar to Haskell. Function applications are written as *f x y*; for example, `amap read world` represents the application of the *amap* function with two arguments: *read* and *world*. One important characteristic of functional languages is that they allow functions to be passed as arguments. Here, *amap* takes the function *read* as argument, and applies it to every value of the region stream *world*; we will discuss the details shortly. `afilter` filters out elements from a region stream that do not match a given predicate, in this case the *aboveThresh* function. And `centroid` is a function that computes the spatial center of mass of a set of sensor readings (where each reading is a scalar value coupled with the *(x,y)* location of the sensor that generated the reading). We assume that every node has access to an approximation of its Euclidean location in real space, though this assumption is not essential to the Regiment language.

So, this program can be interpreted as follows: a region stream is created that represents the value of the proximity sensor on every node in the network; each value is also annotated with the location of the corresponding sensor. Data items that fall below a certain `threshold` are filtered out. Finally, the spatial centroid of the remaining collection of sensor values is computed to determine the approximate location of the object that generated the readings.

4.1 Fundamentals: space and time

Regiment is founded on three abstract polymorphic data types. Polymorphic types are also called *generics*, and are similar in use to C++ templates; they enable generic data structures to be specialized for use with any particular type of data element. Below, the α argument to each type constructor signifies the particular type that it is specialized to hold.

- Stream α — represents a value of type α that changes continuously over time
- Space α — represents a physical space with values of type α suspended in it
- Event α — represents a discrete event that occurs at a particular point in time and that carries a value α when it occurs

The notion of Streams and Events is based on Functional Reactive Programming [10]. In this model, programs operate on a set of time-varying signals. A signal can change its behavior on the arrival of an event. In Regiment, signals become Streams and are used to represent changing sensor state or network status, Spaces represent the physical distribution of information across a network, and Events notify the program of meaningful changes to Streams, allowing triggers.

Because Regiment is a purely functional language, the Stream, Space, and Event types all describe first-class immutable values. This means that values of these types can themselves be passed as arguments, returned from functions, and combined in various ways. Semantically, we can think of each of the three types as having the following meanings:

- Stream $\alpha \approx$ Time $\rightarrow \alpha$
- Space $\alpha \approx$ Location $\rightarrow$ MultiSet α
- Event $\alpha \approx$ (Time , α)

That is, Streams may be formalized as abstract functions that map a time to the value at that time. This is not to say that we would ever *implement* a Stream object as such. Similarly, Spaces may be formalized as functions mapping a location to a set of values existing at that location. Events simply become tuples containing values paired with the associated time of their occurrence.

4.2 Areas, Regions, and Anchors

Until now, we have used "region stream" as an umbrella concept for a changing, distributed chunk of network state. Now we formalize this notion by introducing Regiment's Area and Region types. An Area is a generic data structure for representing volatile, distributed collections of data. A Region is a specific kind of Area used to represent the state of the real, physical network.

We saw before that a Space represents a "snapshot" of values distributed in space at a particular point in time. But we would like for those values—as well as the membership of values in that space—to change over time. To accomplish this we introduce the concept of an *Area*. If we visualize a `Space Int` as a volume with integers suspended throughout, then an `Area Int` would be an animated version of the same thing. The Area data type is built by using Stream and Space together:

Area α = Stream (Space α)

Note that, with this type, an Area's membership and physical extent may change over time. In fact, this type would allow the Area to become an entirely different Space at each point in time. (But the instability would cripple our implementation.) On the other hand, if Area were defined as a Space of Streams rather than Stream of Spaces, then its membership and spatial extent would be fixed but its values varying. Instead, both vary.

Areas are useful constructs, but they don't by themselves provide an initial foothold into the real world. How do we make that first Area? In order to refer to the state of specific sets of nodes in the real world, we define a *Region*, which is an *Area* of *Nodes*. A Node, in turn, is a datatype representing the state of a sensor node in the network at some point in time. It allows access to the node's state, such as its sensor readings, real world location, and the set of other nodes that are part of its communication neighborhood. The precise definition of the Node type, along with its basic operations, are shown in figure 1.

A Region is created as a group of nodes with some relationship to one another such as"all nodes within k radio hops of node N," or "all nodes within a circle of radius r around position X." Regions may be formed in arbitrarily complex ways: using spatial coordinates, network topology, or by arbitrary predicates applied to individual nodes. Hence, Regions may be non-contiguous in space, and their membership may vary over time. The goal of a Region is to get a handle on a group of sensor nodes of interest for the purpose of localizing sensing, computation, and communication within the network. The special region `world` represents all nodes in the network.

One can form a Region by identifying a particular node that acts as the reference point for determining membership in the region: an *Anchor*. The Anchor also acts as the "leader" for aggregate operations in a Region, such as combining values from multiple sensors. Note that the specific node that fulfills the role of Anchor may change over time, for example, if a node fails or loses connectivity to others in the Region. Regiment guarantees that the Anchor object persists across node failures, which may require periodic leader elections.

Examples of Regiment code for forming various Regions:

- **radio_neighborhood hops anch**:
  Forms a Region consisting of all nodes within **hops** radio hops of the given anchor.
- **circle radius anch**:
  Forms a Region consisting of all nodes whose geographical coordinates are within **radius** of **anch.**
- **knearest k anch**:
  Forms a Region consisting of the **k** nodes that are nearest **anch.**

4.3 Basic operations

Regiment defines a number of basic operations on Streams and Areas.

smap *f stream*
amap *f area*

smap applies a function *f* to every data sample within a Stream (across time), returning a new Stream. Similarly, *amap* applies a function *f* across every datum in the Area (across space and time).

afold *f init area*

An Area fold, or *afold*, is used to aggregate the samples from each location in the Area to a single value. The function *f* is used to combine the values in the Area, with an initial value of *init* used to seed the aggregation. *afold* returns a new Stream representing the aggregated values of the Area over time. For example, `afold (+) 0 area` generates a Stream of the time-varying sum of all values in *area*.

afilter *p area*

An Area filter, or *afilter*, pares down the elements of *area* to only those satisfying the predicate function *p*. This filtration must be updates dynamically as the values in *area* change over time.

Regiment also has operations for defining and handling events:

when *p stream*
whenAny *p area*
whenPercent *per p area*

when constructs an Event which fires when the current value of a stream satisfies the predicate *p*. *whenAny*, on the other hand, constructs an Event that fires whenever any single node in an Area matches a predicate *p*. *whenPercent* is similar to *whenAny* but the Event returned only fires when above a certain percentage of elements in the area meet the criteria—potentially an expensive (and difficult to implement) operation.

Using Events, two Streams can be sequenced into a single Stream using the *until* function:

until *event startstream handler*

until switches between Streams. The above call to *until* will produce values from *startstream* until such a time as *event* occurs. At that point, the *handler* (a function) is called on the value attached to the Event occurrence. This handler function must return a new Stream, that takes over producing values where *startstream* left off.

```
type Area a = Stream (Space a)
type Region = Area Node
type Anchor = Stream Node
  – Node: represents a physical mote in the context of a
  – communication network. Provides access to the node
  – state as well as the states of "neighbors".
type Node = (NodeState, [NodeState])

  – NodeState: all the relevent information for a
  – node: id, location, and a set of sensor values
  – (one for each sensor type supported by the node).
type NodeState = (Id, Location, [Sensor])
  – Sensor: force all sensor readings to be floats:
type Sensor = (SensorType, Float)
  – SensorType: predefined enumeration of sensor kinds.
type SensorType =
   PROXIMITY | LIGHT | TEMPERATURE ...

  – Function that returns the NodeState of a Node
get_nstate ::  Node -> NodeState
  – Returns the reading for a given SensorType. For
  – now we assume all nodes support all SensorTypes.
read_nstate ::
   SensorType -> NodeState -> Float

  – And here are two convenient short-hands:
  – Sensing function for Nodes
read_sensor typ nd =
   read_nstate typ (get_nstate nd)
  – Shorthand for reading location (via GPS, etc)
get_location nd =
   read_sensor LOCATION node
```

Figure 1: **Regiment's basic data types** (along with some helpful functions.)

4.4 Spatial operations

Along with these basic operators, Regiment provides several explicitly spatial operations on Areas. For example:

- **sparsify** *percent area*:
  Make *area* more sparse. Each value in the Area flips a biased coin, and stays in the Area with the given probability. This randomization is only done the first time a value enters the Area. The sparse Area is not chaotically recomputed at every time step. **sparsify** can be used, for example, to "weed out" nodes from an overly dense Region.

- **cluster** *area*:
  Cluster a fragmented Area into multiple Areas, each of which is guaranteed to be spatially contiguous. The return type is an Area of Areas.

- **flatten** *area*:
  Flatten takes an Area of Areas and returns a single combined Area. This is the inverse of **cluster**.

- **border** *area*:
  Return a Region representing the set of nodes that form a boundary around the given *area*.

4.5 Example programs

Now we will return to our original example program and examine it in greater detail. Let us start by defining `centroid` using basic Regiment constructs.

```
  – This calcs a weighted avg of vectors.
  – Used to find center of sensor readings.
centroid area =
   divide (afold accum (0,0) area)

  – 'accum' produces a weighted sum.
  – 'wsum' - sum of weights.
  – 'xsum' - sum of scaled locations.
accum (wsum, xsum) (w,x) =
   (w + wsum, x*w + xsum)

  – 'divide' the stream of scaled location
  – values by the sum of the weights.
  – Backslash defines a function.
divide stream =
   smap (\(w,x) -> x/w) stream
```

The `centroid` function takes an area as an input and uses the `accum` function to fold that area down to a stream of sums of sensor readings paired with the scaled locations of each sensor in the region. The `divide` function divides the sum of scaled locations by the sum of the sensor readings. This effectively calculates the center of mass of the locations of those sensors, in a way that recomputes automatically over time.

4.5.1 Tracking multiple targets

Using the **cluster** operation, we can track the location of multiple targets, assuming that the set of nodes near a given target do not overlap:

```
let aboveThresh (p,x) = p > threshold
    read node =
       (read_sensor PROXIMITY node,
        get_location node)
    selected = afilter aboveThresh
                      (amap read world)
    globs = cluster selected
in amap centroid globs
```

This program returns an Area providing approximate target locations for each target being tracked. Note that the number of targets in the Area will vary over time.

4.5.2 Resource efficiency with sentries

As a further refinement, consider a program that only initiates target tracking within the network if any of the nodes on the periphery of the network initially detect the presence of a target. This technique can be used to save energy on the interior nodes of the network, which only need to be activated once a target enters the boundary.

```
let aboveThresh (p,x) = p > threshold
    read node = (read_sensor PROXIMITY node,
                 get_coords node)
    selected  = afilter aboveThresh
                        (amap read world)
    targets   = amap centroid (cluster selected)
    sentries  = amap read (border world)
    event     = whenAny aboveThresh sentries
    handler ev = targets
in until event nullArea handler
```

The last line of the program initiates computation using the **until** primitive. Until `event` fires, the program returns an empty Area (**nullArea**). Once a target is detected by any of the `sentries`, the **nullArea** is supplanted by `targets`, the evaluation of which yields a stream of approximate target locations.

The reader might reasonably be worried that the above program produces a fragile implementation. If even one node in the sentry-border dies, might that let a target through? This depends on the quality of the implementation of the `border` operator. A high quality implementation will respond to failures and have the border sealed again in a bounded amount of time. Also, the programmer may self-insure by making a two layer border as follows:

```
let sent1 = border world
    sent2 = border (subtract world sent1)
    thickborder = union sent1 sent2
    ...
```

4.5.3 Contour finding

The following program computes the *contour* between adjacent areas of the network. Sensor readings on one side of the contour are above a certain threshold, and readings on the other side are below. The contour is returned as a list of points lying along the contour.

```
let mesh = planarize world
    nodesAbove =
       afilter ((>= threshold) .
                (read_sensor SENSTYP))
               mesh
    midpoint nst1 nst2 =
       (read_nstate LOCATION nst1 +
        read_nstate LOCATION nst2) / 2
    contourpoints node =
       let neighborsBelow =
         filter ((< threshold) .
                 (read_nstate SENSTYP))
                (get_neighbors node)
       in map (midpoint (get_nstate node))
              neighborsBelow
    all_contourpoints =
       amap contourpoints nodesAbove
in
  afold append all_contourpoints
```

This program works by pruning the communication graph of the network into an approximately planar form. It then filters out a region of nodes—`abovethresh`—with `SENSTYP` reading above the threshold; this would be all the nodes to one side of the contour. The `contourpoints` function takes a node above the threshold and returns a list of midpoints between that node and each of its neighbors *below* the threshold (on the other side of the contour). Finally, *all_countourpoints* is aggregated by appending together all the individual lists of midpoints, thus yeilding the final countour-line—a Stream of lists of coordinates.

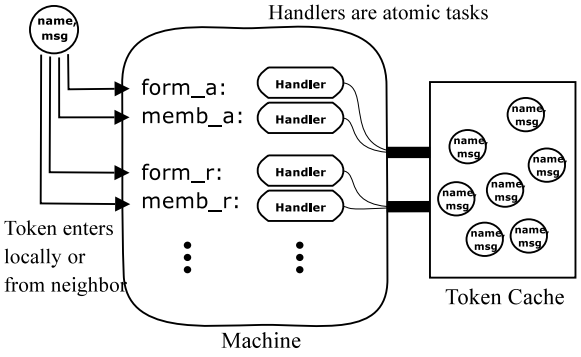


Figure 2: **The Regiment Token Machine model.**

4.6 Feedback and exception handling

Because behavior of the sensor network is stochastic, the response from a region during any time period will involve only a subset of all the nodes that "should" be in that region. The programmer needs feedback on the quality of communication with the region in question. Thus the **fidelity** operator.

fidelity *area*

This operator returns a Stream representing the fidelity of an area as a number between zero and one (an approximation based on the number of nodes responding, spatial density, and estimated message loss).

The programmer will also want feedback about (and eventually control over) the *frequency* of a Stream.

get_frequency *stream*

allows the programmer to monitor the actual frequency of a Stream of values.

Thus, by using these two diagnostic streams, the programmer may set up "exception handlers". This is accomplished by constructing events which fire when fidelity or frequency falls out of the acceptable range. For example, if fidelity drops below a certain level, one may want to switch to a different algorithm.

5 Token Machines

Compiling a global program into node-level code requires an abstract machine model for the compiler to target. The goal of this model is to capture only the essential operations supported by sensor nodes. For this purpose we provide the *Token Machine* (depicted in figure 5). It can be thought of as an intermediate language (IL) between Regiment and the native language and runtime environment supported by individual sensor nodes.

5.1 Tokens and Handlers

A program in the Token Machine model consists of a collection of *token handlers* coupled with local state definitions. Each token handler is associated with a token name and is attached to an atomic task to be executed by a sensor node upon receiving a token matching that name. The Token Machine's concurrency model is similar to TinyOS [14] in that handler tasks may not be blocked or preempted, and run to completion. In many ways, the token machine model is similar to that of Active Messages [27]

Tokens are generated by a node either internally (in response to internal state changes, e.g., a timer interrupt) or by reception of a radio message containing the token identifier and parameters. The most recently received token of each name is cached by the machine. Token handlers can *emit* new tokens by broadcasting a radio message, or *call* local token handlers. Nodes can also *count* the number of times a given token has been received and *clear* the reception count for a given token. Thus Token Machines provide a simple mechanism providing local function calls, remote invocation, and data storage.

One use of tokens is to implement *gradients* [9]. A gradient emanates from a specific origin node with an associated *gradient value*, which is initialized to zero. Each node receiving a gradient token rebroadcasts the token after incrementing the gradient value; each node retains only the lowest-numbered gradient value it has received. A gradient may have an associated *time-to-live* that limits the range of its propagation. Gradients can be used to implement a range of interesting communication patterns, for example, allowing a root node to collect information from all nodes within some communication radius, or allowing nodes to estimate their distance from a set of origin points.

In practice, gradients must be refreshed continuously to maintain themselves in the presence of node and link failures. The epoch frequency for gradient-refresh drives the looping behavior of the system. Every epoch, a wave of tokens moves outward, activating the next step of computation. Gradients can be seen as a more general form of the communication model used by directed diffusion [15] and spanning trees in systems such as TinyDB [18].

5.2 Gradient example: implementing folds

As an example of the use of tokens and gradients, consider aggregating the values of a k-radio-neighborhood group of sensors to an anchor node in its center (the Regiment **afold** operator). This operation proceeds in two steps: region formation (which may be amortized over multiple **afold** operations) and data aggregation. To form the region, the anchor emits a *member* gradient with an initial hopcount of 0 and a time-to-live of k radio hops. The token handler for *member* evaluates whether the receiving node is within the region defined by the **afold** operator; in this case, if the gradient hopcount is less than k, then that node considers itself part of the region. Receiving nodes also remember the node from which they received the lowest hopcount version of the *member* token; call this the node's *parent*. Receiving nodes then increment the gradient hopcount and relay the gradient as long as the hopcount is less than the time-to-live.

Aggregating results back to the anchor is performed

with a second gradient operation, called *return*. *return* takes as arguments a local value to aggregate, as well as a token naming an *aggregation function* that combines values as they travel upwards toward the root of the *member* gradient. Each interior node attempts to keep track of the number of children it has. The handler for *return* checks whether all of the node's children have responded with their own *return* token (using the token's reception count), up to some maximum timeout period. Once this condition is met, the *return* handler combines received values from the node's children using the aggregation function and issues a *return* to its own parent with the combined value. These *return* messages include the node's parent ID so that all other nodes will ignore its reception.

5.3 Gradient example: leader election

Gradients also make it straightforward to implement distributed leader election among a group of nodes. All of the nodes participating in the election emit a gradient named *elect*, which includes its local node ID. The token handler on each node remembers the lowest-valued node ID received so far. When the token is received, if the received ID is smaller than the previously stored value, the new ID is remembered *and the gradient token is relayed*. Therefore, all nodes participating in the election initiate gradients, but only the gradient of the lowest-numbered node will continue to propagate. The root of this gradient is the leader.

6 Current status

Regiment poses implementation challenges that are both deep and broad. Presently, we are exploring the feasibility of the basic Regiment primitives through a highly restricted subset of the language. This subset eliminates general purpose function application, and forbids free variables within functions (disallowing closures). Functions may still be defined, but they may only be applied by using the Regiment primitives **afilter**, **afold**, **amap**, and **smap**. We have also postponed typing issues by making our prototype dynamically typed. We have implemented a prototype of the compiler and have demonstrated several example applications running in simulation; we intend to implement a back end compiler to generate TinyOS code from the Token Machine representation, allowing us to test the system on real sensor nodes.

6.1 Compilation strategy

A program in the restricted language is best visualized as a dataflow graph; Figure 3 depicts the graph for a simple program that computes the **smap** of a function g over the **afold** of f over a region of nodes defined by a circle around the point $(30, 40)$. Our compiler generates code for such a data-flow graph by directly translating each *edge* in the graph into some number of token handlers.

Values in the system are divided into *distributed* and *local*. Every distributed value corresponds to some phenomena happening in space. Regions and Anchors are distributed, whereas numbers and functions are local. In figure 3, edges are either solid or dotted depending on whether they carry distributed values or local ones.

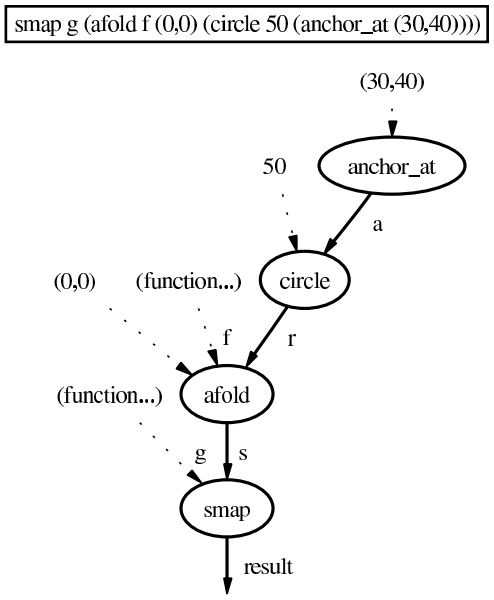


Figure 3: **A Regiment program represented as a dataflow graph.** *Network-distributed values flow along solid edges, and local constant values (including functions) flow along dotted edges.*

We standardize an interface among distributed values such that every distributed value (solid edge) produces at least a *formation* token handler and a *membership* token handler. The former represents an onus to create that Region or Anchor—form the circle, do the filtration, elect the leader—and the latter is a notice that the Area/Anchor is active and the current node is participating in it.

6.1.1 Example walk-through

The example portrayed in figure 3 has four distributed values (**a, r, s, result**) and several local values (**f** and **g** and several numeric constants). Each of the distributed values generates both a formation and membership token, for example, *form_a* and *memb_a*.

Because **a** is the only distributed value produced by a leaf node, *form_a* tokens are seeded into the network initially. They cause nodes to check their distance from the targeted Euclidean coordinate, $(30, 40)$. Nodes that are close enough to that location initiate and participate in a leader election. The token *memb_a* is fired when a node becomes leader.

Because **r** is the next step in the chain beyond **a**, the handler for the *memb_a* token immediately calls *form_r*. Forming **r** is simple; it just requires emitting a single gradient with the token *memb_r*. As nodes receive the *memb_r* token they call the *form_s* token. (Again, simply because it's next in line.) The *form_s* handler begins *return*ing values along the back-trail of the *memb_r* gradient. When they arrive

back at the root, the special return handler calls *memb_s*. The value **s** has successfully been formed. *memb_s* in turn calls *form_result*, which simply applies the function **g** locally to the stream **s**, and we have our result.

This example was simple—at no point did a primitive depend on more than one distributed value. But we hope that it conveys a feeling for the process. It is important to note also that this simple example uses only a push model for the data-flow graph. (Leaf nodes push their results down to the root.) The `until` primitive makes necessary use of the `pull` communication model because it waits for an Event before starting a Stream. The latter stream must have some kind of pull exerted on it to prompt it to begin execution.

7 Future work and Conclusion

Future work will proceed in several directions. Because of the large gap between Regiment's semantics and target architecture, compiling it is a challenge. We will explore the possibility of loosening the restrictions on our initial version of Regiment, providing more general purpose functionality. We plan to investigate both static and dynamic optimizations in terms of resource usage and communication bandwidth requirements for a range of Regiment applications. We believe that the Token Machine model and the use of gradients makes it straightforward to realize good communication locality. We intend to introduce primitives that allow the user to control tradeoffs between energy, latency, and accuracy [29], which are critical for sensor network application designers to consider.

Our vision is that sensor network applications can be expressed in a very high-level *macroprogramming* language that abstracts away the low-level details of sensing, communication, and energy management. We argue that the use of functional programming languages is essential for capturing data parallelism and enabling the compiler to make informed decisions about the scheduling and placement of computation in the sensor network. We have demonstrated some interesting first steps in this direction through the design of Regiment and its underlying runtime model, Token Machines. Regiment provides the ability to programmatically build spatial regions within the network, and use them for localized sensing, computation, and communication.

References

[1] T. Abdelzaher, B. Blum, Q. Cao, D. Evans, J. George, S. George, T. He, L. Luo, S. Son, R. Stoleru, J. Stankovic, and A. Wood. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks.

[2] J. Annevelik. Database programming languages: A functional approach. In *Proc. of the ACM Conf. on Management of Data*, pages 318–327, 1991.

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. *STREAM: The Stanford Data Stream Management System*. 2004.

[4] Arvind and Rishiyur Nikhil. *Implicit Parallel Programming in pH*. Morgan Kaufman, 2001.

[5] F. Bancilhon, T. Briggs, S. Khoshafian, and P. Valduriez. Fad, a powerful and simple database language. In *Proc. Conf. on Very Large Data Bases (VLDB)*, 1987.

[6] Guy E. Blelloch. NESL: A Nested Data-Parallel Language. Technical Report CMU-CS-93-129, April 1993.

[7] Cristian Borcea, Chalermek Intanagonwiwat, Porlin Kang, Ulrich Kremer, and Liviu Iftode. Spatial programming using smart messages: Design and implementation. In *24th International Conference on Distributed Computing Systems (ICDCS 2004)*, March 2004.

[8] Daniel Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, February 1999.

[9] Daniel Coore, Radhika Nagpal, and Ron Weiss. Paradigms for structure in an amorphous computer. Technical Report AIM-1614, MIT, 6, 1997.

[10] Conal Elliott and Paul Hudak. Functional reactive animation. In *Proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP '97)*, volume 32(8), pages 263–273, 1997.

[11] Deepak Ganesan, Ben Greenstein, Denis Perelyubskiy, Deborah Estrin, and John Heidemann. An evaluation of multi-resolution search and storage in resource-constrained sensor networks. In *Proc. the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, November 2003.

[12] Benjamin Greenstein, Deborah Estrin, Ramesh Govindan, Sylvia Ratnasamy, and Scott Shenker. DIFS: A distributed index for features in sensor networks. In *Proc. the First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[13] Wendi Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proc. the 5th ACM/IEEE Mobicom Conference*, August 1999.

[14] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In

Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 93–104, Boston, MA, USA, November 2000.

[15] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proc. International Conference on Mobile Computing and Networking*, August 2000.

[16] S. P. Jones and J. Hughes. Report on the programming language haskell 98., 1999.

[17] Attila Kondacs. Biologically-inspired self-assembly of 2d shapes, using global-to-local compilation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[18] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *Proc. the 5th OSDI*, December 2002.

[19] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 14–23. IEEE Computer Society Press, Washington, DC, 1989.

[20] Nagpal, Shrobe, and Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *2nd International Workshop on Information Processing in Sensor Networks (IPSN '03)*, April 2003.

[21] Radhika Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, June 2001.

[22] Suman Nath, Yan Ke, Phillip B. Gibbons, Brad Karp, and Srinivasan Seshan. IrisNet: An architecture for enabling sensor-enriched Internet service. Technical Report IRP-TR-03-04, Intel Research Pittsburgh, June 2003.

[23] Simon L. Peyton Jones and André L. M. Santos. A transformation-based optimiser for Haskell. volume 32, pages 3–47, 1998.

[24] Pointon, Trinder, and Loidl. The design and implementation of Glasgow Distributed Haskell. *Lecture Notes in Computer Science*, 2001.

[25] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: A geographic hash table for data-centric storage in sensornets. In *Proc. the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, Georgia, September 2002.

[26] G. L. Steele and W. D. Hillis. Connection machine lisp: Fine grained parallel symbolic programming. pages 279–297.

[27] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: a mechanism for integrating communication and computation. In *Proc. the 19th Annual International Symposium on Computer Architecture*, pages 256–266, May 1992.

[28] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad-hoc. In *Conference Record of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 60–76. ACM, January 1989.

[29] Matt Welsh. Exposing resource tradeoffs in region-based communication abstractions for sensor networks. In *Proc. the 2nd ACM Workshop on Hot Topics in Networks (HotNets-II)*, November 2003.

[30] Matt Welsh and Geoff Mainland. Programming sensor networks using abstract regions. In *Proc. the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, March 2004.

[31] Kamin Whitehouse, Cory Sharp, Eric Brewer, and David Culler. Hood: A neighborhood abstraction for sensor networks. In *Proc. the International Conference on Mobile Systems, Applications, and Services (MOBISYS '04)*, June 2004.

[32] Yong Yao and J. E. Gehrke. The Cougar approach to in-network query processing in sensor networks. *ACM Sigmod Record*, 31(3), September 2002.

[33] S. Zdonik, M. Stonebraker, M. Cherniack, U. Cetintemel, M. Balazinska, and H. Balakrishnan. The aurora and medusa projects. *Bulletin of the Technical Committee on Data Engineering*, 2001.

StreamGlobe: Adaptive Query Processing and Optimization in Streaming P2P Environments

Bernhard Stegmaier Richard Kuntschke Alfons Kemper

TU München - Lehrstuhl Informatik III
Boltzmannstraße 3
D-85748 Garching bei München
Germany
<*first_name.last_name*>@in.tum.de

Abstract

Recent research and development efforts show the increasing importance of processing data streams, not only in the context of sensor networks, but also in information retrieval networks. With the advent of various mobile devices being able to participate in ubiquitous (wireless) networks, a major challenge is to develop data stream management systems (DSMS) for information retrieval in such networks. In this paper, we present the architecture of our *StreamGlobe* system, which is focused on meeting the challenges of efficiently querying data streams in an ad-hoc network environment. StreamGlobe is based on a federation of heterogeneous peers ranging from small, possibly mobile devices to stationary servers. On this foundation, self-organizing network optimization and expressive in-network query processing capabilities enable powerful information processing and retrieval. Data streams in StreamGlobe are represented in XML and queried using XQuery. We report on our ongoing implementation effort and briefly show our research agenda.

1 Introduction

In recent years, Peer-to-Peer (P2P) networks have gained huge attention both in the media and the computer science community. This is, on the one hand, due to the stunning success of filesharing systems like, e.g., Napster and Gnutella. But on the other hand, it is also caused by the degree of flexibility these networks provide. For example, they can be used for setting up ad-hoc sensor networks where sensors can join and leave the network at any time, e.g., while moving across the area covered by the respective network. Of course, this does not only hold for the data delivering sensors, but also for the network nodes that query the data streams within the ad-hoc network. In the past, various approaches for finding information, i.e., documents, files, etc., in P2P networks have been studied, which has led to a number of topologies for P2P networks, one example being super-peer networks [28]. Dealing with data streams, finding peers which deliver the required information is not the only task. Additionally, a continuous data flow from data sources to consumers in the network has to be established. An interesting challenge arising in this highly dynamic environment is to develop a distributed, self-organizing system for efficient routing and in-network query processing. We pursue this goal with our *StreamGlobe* system which is based on its predecessor ObjectGlobe [3]. StreamGlobe extends ObjectGlobe—which is mainly focused on distributed query processing for persistent data on the internet—by introducing query processing capabilities on data streams in the network. In our context, data streams are represented in XML and queried (i.e., subscribed) using XQuery. While StreamGlobe is not restricted to sensor networks, we use them as a motivating example in the following.

Consider Figure 1 as an abstract example of a possible application scenario for StreamGlobe. The depicted network contains four so-called *super-peers* (SP_0 to SP_3), forming a stationary super-peer backbone network, and five possibly mobile *thin-peers*, or peers for short, (P_0 to P_4) connected to the backbone. Peers P_0, P_2 and P_3 are a cell phone, a laptop, and a PDA, respectively. These peers are meant to register queries in the network and are therefore at the receiving end of data streams. In contrast to that, peers P_1 and P_4 are sensors delivering their sensor data to the network in the form of XML data streams. Two examples for applications of similar real-life networks would be satellite communication and weather observation. In the former case, orbiting satellites would be the moving sensors—or rather collections of sensors—streaming their data to various receiving stations on the ground for evalu-

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

ation. In the latter case, the sensors would be attached to weather balloons or observation planes, delivering data like temperature, humidity, etc. to enable weather forecasts for different regions.

To illustrate some of the difficulties of query processing in such networks and to motivate our approach, we now introduce a rather simplified real-world example in a little more detail. Let us assume that P_4 in Figure 1 delivers a data stream produced by special sensor suits worn by firefighters in action. The sensors continuously deliver sensor readings containing the corresponding firefighter's identity (`id`), a timestamp (`time`), and the GPS coordinates of the sensor (`x`, `y`), as well as information about the firefighter's vital statistics and the environmental conditions. We have exemplarily chosen to monitor body temperature (`bt`), pulse rate (`pr`), and oxygen saturation (`os`), as well as environmental temperature (`et`), carbon dioxide concentration (`CO2`), and sulfur dioxide concentration (`SO2`). For brevity, we use the following simplified DTD to describe the data stream, although StreamGlobe actually employs XML Schema.

```
<!ELEMENT reading (id, time, x, y,
                   bt, pr, os,
                   et, CO2, SO2)>
<!ELEMENT id (#PCDATA)>
...
```

The remaining elements have analogous DTD entries. Let us now further assume that P_0 and P_2 are devices used by an emergency physician and the fire department, respectively. The former should receive a notification on a cell phone whenever a firefighter's oxygen saturation reaches a critical level. Therefore, the peer represented by the physician's cell phone registers the following XQuery.

```
for $m in stream("firefighters")/reading
where $m/os < 92 or $m/os > 98
return
    <alert>
        {$m/id} {$m/time} {$m/x} {$m/y}
        {$m/os}
    </alert>
```

The fire department wants to monitor the environmental conditions, e.g., to be able to issue a warning if the conditions get critical for the firefighters on site or the residents living nearby. Thus, it registers the following XQuery.

```
for $m in stream("firefighters")/reading
return
    <gas>
        {$m/id} {$m/time} {$m/x} {$m/y}
        {$m/CO2} {$m/SO2}
    </gas>
```

StreamGlobe will handle this scenario as follows. Suppose we want to reduce network traffic. The data of P_4 will be sent to SP_3 where it will be filtered, leaving only the elements `id`, `time`, `x`, `y`, `os`, `CO2` and `SO2` in the stream. The elements `bt`, `pr` and `et` can be removed as they are not needed (i.e. not subscribed) anywhere else in the network, leading to a smaller data stream and reducing network traffic. The resulting stream, containing the combined information for satisfying the queries of P_0 and P_2, is routed to SP_2. Note that up to now, data needed by both P_0 and P_2 has been routed as one single stream through the network. At SP_2, however, the stream has to be split into the—in our case—non-disjoint parts for the two receiving peers. This involves replicating the stream and again filtering the two new streams, resulting in two streams which constitute the final results for the two queries. These are eventually routed to P_0 and P_2 via SP_0 and SP_1, respectively.

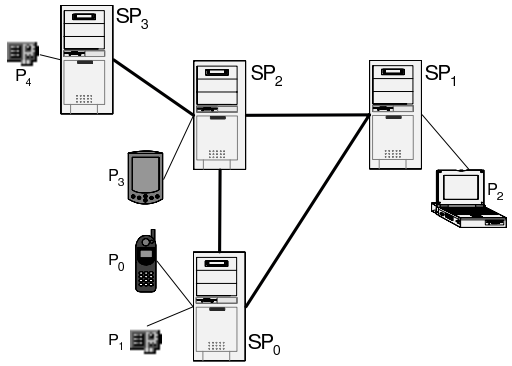


Figure 1: Example Scenario

Decisions such as where to execute which operators in the network and how to route the data streams are made by the StreamGlobe query optimizer. Additional difficulties arise by the fact that the network can change over time by adding or deleting queries and data streams which requires a strategy for continuous or periodic reoptimization. The distinguishing features of StreamGlobe compared to related systems are thereby its self-organizing network, in terms of continuous reactions to dynamic changes in registered data streams and queries, and its routing and optimization approaches for query and network traffic optimization in P2P networks.

The remainder of the paper is organized as follows. Section 2 presents some related work. In Section 3 we give an overview of the StreamGlobe system architecture. Section 4 deals with optimization and query processing in StreamGlobe. In Section 5 we present a brief report on the current implementation status of our StreamGlobe prototype. Finally, Section 6 concludes the paper and gives an outlook on future work.

2 Related Work

In the following, we present an overview of some work related to our StreamGlobe system. In particular, we deal with work in the fields of data stream systems, query processing, network architecture, and grid computing.

2.1 Data Stream Systems

With StreamGlobe being a system that handles and processes data streams, it is worthwhile to take a look at other recent approaches to building data stream systems.

One important project is TelegraphCQ [7]. This is a system that deals with continuously adaptive query processing in a data stream environment. Cougar [30] tasks sensor networks through declarative queries. Aurora [6] is a new DBMS for monitoring applications and constitutes a centralized stream processor for dealing with streaming data. In [10] two complementary large-scale distributed stream processing systems, Aurora* and Medusa, are described. Aurora* is a distributed version of Aurora with nodes belonging to a common administrative domain. Medusa supports the federated operation of several Aurora nodes across administrative boundaries. STREAM [2] incorporates its own declarative query language for continuous queries over data streams and relations. It handles streams by converting them into relations using special windowing operators and converting the query result back into a data stream if necessary. PIPES [20] is a recent public domain infrastructure for processing and exploring data streams.

All of these systems—more or less—focus on special aspects of (adaptive) query processing, load balancing, or quality-of-service management. The major contribution of StreamGlobe is that it does not only efficiently locate and query data streams, but also employs in-network query processing for adaptively optimizing data flow within the network. Thus, StreamGlobe pushes query processing from subscribing clients towards data sources in the network. The optimization is based on data stream clustering derived from clustering the queries in the system. NiagaraCQ [8] intends to achieve a high level of scalability in continuous query processing by grouping continuous queries according to similar structures. In StreamGlobe, we employ a similar multi-query optimization approach to reduce network traffic and to enable efficient query evaluation.

2.2 Query Processing

With respect to query processing, works in the fields of multi-query optimization, as pointed out above, and continuous queries are related to StreamGlobe. Multi-query optimization (MQO) has been addressed in [26]. It pursues the goal of processing multiple queries all at once instead of one query at a time. The main optimization potential lies in the fact that queries may share a considerable amount of common—or at least similar—input data that can be reused for more than one query. Obviously, StreamGlobe in general has to deal with a set of queries simultaneously, thus rendering multi-query optimization an applicable and suitable optimization approach. Also, queries in StreamGlobe are usually continuous queries over data streams. Efficient processing of such queries has been examined in [22]. Query processing in sensor networks has been explicitly addressed in [31].

Multicast in IP, ad-hoc and sensor networks, described for example in [15], routes data towards receiving ends in a way that reduces network traffic by transmitting the same message or document only once for all recipients instead of multiple transmissions, one for each recipient. It is important to point out that our work differs from these approaches in a major way. Instead of only reusing existing messages or documents, our system is able to perform expressive in-network transformations of data streams. Therefore, it can dynamically create appropriate data streams that best fit the queries to be answered while at the same time reducing network traffic.

To achieve this goal, StreamGlobe uses clustering techniques to identify reusable existing data streams in the network that fit newly registered queries. This approach has similarly been applied in the world of persistent data where view materialization and view selection are used to improve the efficiency of query processing [21]. In [29], further algorithms for solving the view materialization problem are devised. Materialized view selection and maintenance have also been examined using techniques of multi-query optimization [23].

As already mentioned, StreamGlobe uses XQuery to query XML data streams. In [11] an XQuery engine called XQRL for processing XQueries on streaming XML data is introduced. In StreamGlobe, we use FluX [19], another XQuery engine for efficiently processing XML data streams. The query containment problem in the context of XML queries, which is relevant for multi-query optimization, has been addressed in [27].

2.3 Network Architecture

Considering network architecture, a lot of work has been done with respect to P2P, Publish&Subscribe, and ad-hoc networks.

P-Grid [1] is a self-organizing, structured P2P system. The notion of self-organization with respect to stream processing and stream routing is also central to StreamGlobe. In [28] the concept of super-peer networks is introduced. These networks are meant to improve the scalability of P2P networks by using a super-peer backbone network. The super-peers usually are powerful servers. Less powerful, possibly mobile thin-peers can register and deregister themselves in the network via the super-peers.

HyperCuP [25] is an approach that uses hypercubes as a network topology in P2P networks. It thereby achieves a logarithmic upper bound for the number of hops needed to get from one super-peer in the network to any other super-peer. This topology is used in [5] to deal with distributed queries and query optimization in P2P systems.

2.4 Grid Computing

StreamGlobe builds on and extends the Open Grid Services Architecture (OGSA) and its reference implementation, the Globus Toolkit [14] by adding data stream processing capabilities to the grid computing domain. A related approach, also building on Globus, is described in [9]. However, this alternative approach concentrates mainly on data stream analysis and quality-of-service aspects in data stream delivery whereas we primarily focus on self-organization, distributed in-network query processing and optimization.

Another system building on the Open Grid Services Architecture is OGSA-DAI (Open Grid Services Architecture Data Access and Integration) [24]. As the name suggests, this project is concerned with constructing a middleware to

enable the access and integration of data from distributed data sources via the grid. It also contains a distributed query processor called OGSA-DQP. In contrast to StreamGlobe, OGSA-DAI has no special focus on data streams.

3 StreamGlobe Architecture Overview

StreamGlobe constitutes a federation of servers (i.e., peers) which carry out query processing tasks according to their capabilities. The basic architecture of a peer is depicted in Figure 2. The various layers of this architecture will be sketched in the following. Dashed lines mark layers whose presence depends on the capabilities of the respective peer.

3.1 Open Grid Services Architecture

The StreamGlobe architecture is based on grid standards. Grid computing [13] and the associated Open Grid Services Architecture (OGSA) [12] have gained considerable attention recently. Grid computing denotes a distributed computing infrastructure where computers can exchange data and perform large-scale resource sharing over the grid. To achieve this, an architecture for integrating heterogeneous dynamic services while guaranteeing certain quality-of-service requirements is needed. For this purpose, the Open Grid Services Architecture has been developed.

Despite the growing importance of the grid standards, data stream processing in the grid computing context has hardly been investigated so far. We have decided to implement our StreamGlobe prototype as an extension of the Globus Toolkit for grid computing [14]. Globus is a reference implementation of the Open Grid Services Architecture. Our goal is to use existing Globus techniques for our purposes where possible and to integrate the StreamGlobe system and its functionality into the toolkit as an extension of Globus for data stream processing.

The main aspects of Globus that will be used in StreamGlobe are communication mechanisms and *service data elements*. Service data elements can be associated with any service in the grid. They are essentially XML documents satisfying a given XML Schema and describing properties of the service they are associated with. In our context, service data elements will be used for describing data streams and properties like bandwith of network connections, processing capabilities of peers, etc.

3.2 Network Topology

In the OGSA framework, direct communication between all participating grid services is allowed. However, this behavior is not the normal way of communication in networks including mobile devices. It might not even be desirable in a scenario that tries to reduce network traffic as in our case. For instance, mobile sensors will normally communicate via some kind of access point they are connected to. Hence, in StreamGlobe we establish a logical P2P overlay network constituting a federation of heterogeneous peers. Developing a research platform, we do not restrict ourselves to employing a special P2P network topology for StreamGlobe at the moment. The P2P network consists of a set of *peers*. Each peer has a set of other peers as

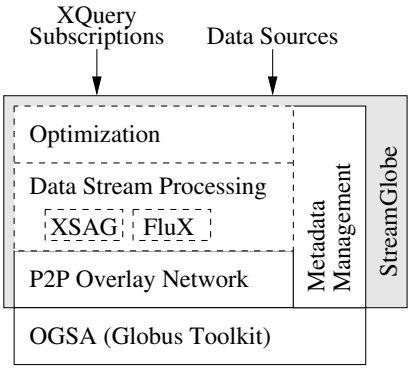


Figure 2: Architecture Overview

neighbors. A peer only interacts with its neighbors, i.e., no direct communication takes place between two peers not being neighbors. If data has to be transferred between two random peers, a *route* between these two peers has to be established such that two successive peers on this route are neighbors and the starting point and the end point of the route are the source peer and the destination peer, respectively. For the implementation of this overlay network, previous work on P2P network topologies can be employed, e.g., a structured approach based on Cayley graphs as used in the HyperCuP [25] topology. Since a major goal is building a network with highly heterogeneous peers with respect to computing power—ranging from small, mobile devices to stationary workstations or servers—, we have to classify peers according to their capabilities. *Thin-peers* are devices with low computational power, like sensor devices, PDAs, cell phones, etc., which are not able to carry out complex query processing tasks. In contrast, *super-peers* are stationary workstations or servers providing enough resources for extensive query processing. These super-peers establish a backbone taking over query processing tasks which cannot be performed by other peers. Thus, they constitute a super-peer backbone network similar to that in [28].

3.3 Client Interface

User interaction in StreamGlobe is depicted at the top layer of Figure 2. StreamGlobe enables clients to specify *subscription rules* for information processing and retrieval using the XQuery language. Subscription rules are registered at certain peers, i.e., normally at the devices users are working with, e.g., their laptops, PDAs, cell phones, etc. In our context, subscriptions are transforming queries and not just queries for retrieving matching files or documents. In fact, StreamGlobe enables expressive transformations of data streams according to registered subscription rules. Thus, it allows clients to flexibly tailor data streams to their individual requirements.

Similarly, data sources also register the provided data streams at a certain peer within the StreamGlobe system. Data streams can be registered in two ways. A data source may register its data stream as an individual stream, which then is published using a unique identifier. Another possibility is registering a data stream as part of a *virtual data*

stream, which again is accessible using a unique identifier and multiplexes all the data of the participating data sources into one single stream. This technique is used in the introductory example to merge the sensor data of all firefighters. The schema of the data streams is specified using XML Schema. Streams are fed into StreamGlobe using *wrappers*, which are running on corresponding peers and transform the data into a suitable format, e.g., by converting raw sensor data to XML.

3.4 Peer Architecture

A more detailed view of the peer architecture is depicted in Figure 3. It basically reflects the structure sitting on top of the P2P network layer of Figure 2. The various components are implemented as cooperating grid services in the OGSA framework. The individual peers exchange control information, e.g., registration of new neighbors, subscriptions, etc., via a top-level interface service, which dispatches the messages to corresponding subsidiary StreamGlobe services, e.g., the optimization or the query engine service. The communication of these services is conducted via the RPC mechanisms of the Globus Toolkit. All services marked by solid rectangles are mandatory for every peer. Dashed boxes mark services that vary between different peers according to their functionality, as mentioned earlier. For example, thin-peers do not incorporate a complete optimization and query execution unit, but only provide basic functionality. A cell phone might for instance only provide functionality for receiving and displaying data streams and a sensor device might only be able to transmit its measurement data.

The metadata management component, which will be discussed further in the next section, interacts with each of the components and provides information needed for network management, optimization, and query execution. Peers exchange XML data streams representing user data over their data ports. The XML data streams are initially parsed by the wrappers and represented as a sequence of SAX events. Special events are interspersed within these streams which are used for internal purposes. For example, synchronization marks are generated whenever the system restructures the data flow to synchronize all affected peers for the change in query execution. Since the Globus Toolkit currently does not provide suitable techniques for transmitting data streams, we use our own protocol based on TCP connections for this purpose.

3.5 Metadata Management

As Figures 2 and 3 suggest, metadata is needed in all layers of the StreamGlobe architecture. The metadata management (MDV) is based on the distributed metadata management of ObjectGlobe [16] and forms a backbone that peers exchange metadata with. In particular, the metadata management component records the following information:

- **Network:** The metadata management records the neighborhood relationships between peers needed for establishing the P2P overlay network.

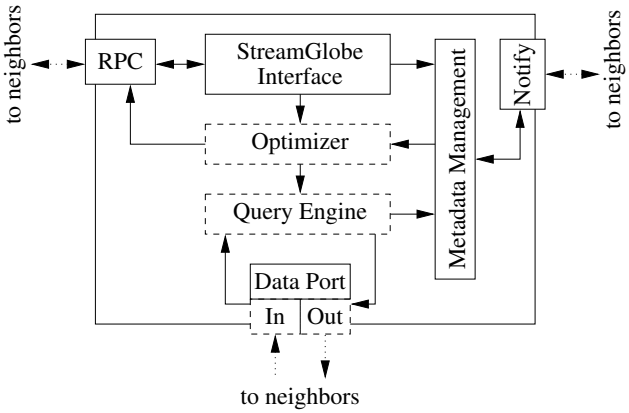


Figure 3: Peer Architecture

- **Subscriptions:** All subscription rules and registered data sources are recorded. For each registered data source, the schema of the data stream is stored. Schemas of data streams are specified using the XML Schema language.
- **Optimization:** The metadata management maintains information needed for optimizing the network. Among others, it maintains properties of network connections, like bandwith and current amount of network traffic. It also maintains the computational capabilities of the peers and statistics of the data streams, i.e., size and cardinality of the elements of a data stream. The statistics can be provided either by the data source itself or by computing them online as the corresponding wrapper feeds the data stream into StreamGlobe.

All metadata is stored locally at a peer in the form of Globus service data elements. For being able to optimize the network, special speaker-peers, which will be introduced in Section 3.6, will need to have more global information about a special set of peers (a certain subnet). In this case, those special peers maintain additional information, e.g., the graph of the network topology of the respective set of peers, or are able to request the desired information from the corresponding peers, e.g., statistics of a certain data stream. To maintain a consistent state, peers have to notify the speaker-peer of changes, e.g., if a peer joins or leaves the network, new subscriptions or data streams are registered or existing subscriptions or data streams are deregistered, etc. Therefore, MDVs of peers register themselves as notification sinks or notification sources at the MDV of their speaker-peer using the notification mechanism of the Globus Toolkit.

3.6 Optimization and Evaluation Strategy

In Section 1, we have briefly introduced our approach of optimizing the data flow in the network using in-network query processing. In the following, we give an overview of the optimization and evaluation strategy we employ in StreamGlobe.

Optimization in a distributed architecture implies several challenges. In order to perform optimization, some metadata about the network—as described in the previous section—has to be available. In a distributed system, there are basically three approaches for performing optimization using such metadata:

1. A single optimizing component has global knowledge of all metadata and performs optimization with a global view of the network.
2. Every peer has only local knowledge of its own metadata (including that its neighbors can be asked for their metadata) and tries to optimize the network by making locally optimal decisions.
3. A hybrid approach, in which special peers have global knowledge of (small) subnets which are individually optimized by the responsible peer.

Since we assume a large, distributed environment, a centralized optimization component as in the first method is infeasible. The second approach fits quite nicely into a distributed P2P network, but it seems unlikely that it will deliver acceptable results. Hence, we focus on the hybrid approach: A selected super-peer, called *speaker-peer*, is responsible for optimizing a certain subnet of the network. Of course, this subnet may include other super-peers that will not actively participate in optimizing this part of the network. With peers joining and leaving subnets, a speaker-peer might decide that a subnet is getting too big (or too small). In this case, the subnet is split into two new subnets and for each new subnet a responsible speaker-peer is elected among the super-peers (or analogously a subnet is merged with a neighboring subnet if it is getting too small). Additionally, by varying the maximum size of a subnet optimized by a speaker-peer, the approaches (1) and (2) can be simulated, which enables an evaluation of all three approaches in terms of optimization quality.

Basically, optimization in StreamGlobe determines the peers at which (at least parts of) the subscriptions are executed and decides how to route the data streams in the network. Optimization has three major goals:

1. Enable users to register arbitrary subscriptions at any (suitable) device regardless of its processing capabilities.
2. Achieve a good distribution of data streams in the network without congesting it with redundant transmissions.
3. Optimize the evaluation of a large number of subscription rules by means of multi-query optimization.

The goals (1) and (3) are accomplished by pushing query execution into the network. Subscription rules, i.e., XQueries, are evaluated using the FluX query engine [19] that was developed in cooperation with our group. The second goal is achieved by placing *filtering operators* on the routes of data streams. These filtering operators are also executed by FluX. They could alternatively be implemented using special filtering techniques such as XSAGs [18]. More details will be presented in Section 4.

Of course, optimization is a continuous process which reoptimizes the system on-the-fly as peers come and go, data sources and subscription rules are registered and deregistered, and data streams change over time.

4 Optimization and Query Processing

In this section, we describe some of our approaches to optimizing network traffic and performing efficient query processing in StreamGlobe. This substantiates the strategy introduced in the previous section.

4.1 Optimization

First, we address the key ideas for achieving the three optimization goals stated at the end of Section 3.6. The first goal is achieved by appropriately pushing subscription evaluation into the network. This is done by executing the subscription as a whole or in part at one or more appropriate peers on a route from the data sources to the peer where the subscription was registered. An appropriate peer is a peer that is able to process the subscription, i.e., has sufficient computing power and is selected by the query optimizer, taking into account optimization goals such as, e.g., reducing network traffic. In order to support powerful subscription rules, the concept of *mobile code* is employed. Besides peers providing a basic set of functionality, users are enabled to include user-defined code in subscription rules, e.g., predicates, aggregation operators, etc. This user-defined code is subsequently instantiated at the peer processing the corresponding part of the subscription.

The second goal is accomplished by using two techniques complementing each other. The first technique is *filtering* of data streams. Filtering is achieved by using either projection (called structural filtering) or selection (called content-based filtering) or both on the elements of a data stream—as described in the example scenario in Section 1—and is performed by *filtering operators*. These filtering operators are executed at peers on the route of the data stream as close to the source of the stream as possible. Thus, the amount of data that has to be transmitted through the network is reduced. The second technique is *data stream clustering*. This term denotes the combination of several similar or equal data streams in the network to form one single stream that serves multiple recipients. Data stream clustering in StreamGlobe works as follows. During the registration of a new query, the system parses the query, identifies its properties and stores them in a suitable data structure. In our case, this will be a Globus service data element. The properties of a query include the data streams needed to answer the query (content aspect), the operations, e.g., projections, selections, joins, etc., used to transform these input streams (structural aspect) and the conditions needed for these operations, e.g., projection attributes, selection and join predicates, etc. All transformed data streams in the system, that where generated by a query, are equally represented by their respective properties. Initial data streams, registered at a super-peer by some data

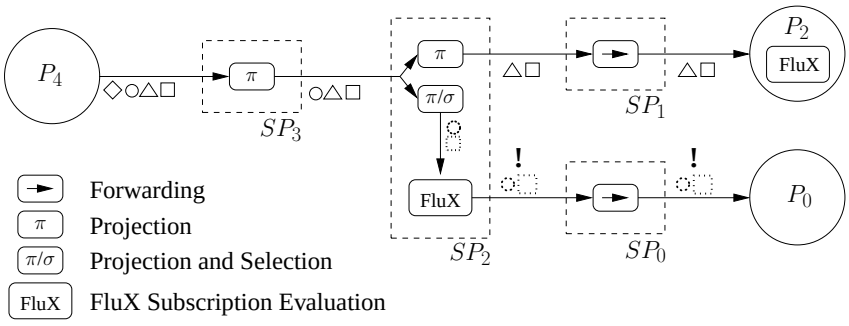


Figure 4: Query Evaluation Plan for the Example Scenario

source, are represented by a unique id. The reason for choosing this properties approach is to get one level of abstraction higher compared to the schema representation of data streams, thus facilitating the comparison of streams and the search for reusable data streams in the network.

During the actual data stream clustering stage, the speaker-peer of the affected subnet looks up all relevant metadata (i.e. service data elements) of existing data streams in its subnet and compares their properties to those of the newly registered query. In a first simple greedy approach, the speaker-peer selects those data streams as input streams for the new query that contain the necessary information for answering the query, contain the least amount of unnecessary information, and have to be routed through the minimum number of peers to get to the recipient. Of course, the decision where to execute certain query operators, e.g., joins, in the network has also to be made. This, along with more sophisticated methods for searching reusable streams and routing them to recipients, is the subject of future research and will be based on an appropriate cost model. Furthermore, we also intend to investigate strategies for reorganizing the network in order to keep the system globally effective even if local evolutions due to network and/or subscription changes lead to a deterioration of global system performance.

Data stream clustering as described above also contributes to fulfilling the third goal of effective multi-query optimization. In every subnet, the speaker-peer analyzes the registered subscriptions and identifies common subexpressions. These common subexpressions are evaluated once in this subnet by executing a subscription rule corresponding to a common subexpression at a suitable peer. Rather than individually evaluating this subexpression in each of the original subscriptions, the subscriptions are rewritten to utilize the newly generated and specialized data stream stemming from the common subexpression. Besides reducing the workload of the affected peers, network traffic might be further reduced. For instance, a common task will be aggregating sensor data. Instead of transmitting the whole dataset to every peer performing the same aggregation, it will be executed near the data source and only the aggregated results, which will constitute a smaller data volume, will be delivered to the respective peers. Furthermore, existing aggregated data streams in the system can be reused to compute more common aggregates similar to the roll up and the cube operations in data warehousing.

Figure 4 shows the query evaluation strategy using the example scenario from Section 1. The symbols at the network connections represent groups of elements. The diamond represents the elements `bt`, `pr`, and `et`, the circle represents `os`, the triangle represents `CO2` and `SO2`, and the rectangle represents `id`, `time`, `x`, and `y`. Projections cause symbols to disappear as their corresponding elements are filtered out of the stream. Selections remove certain instances of elements that do not fulfill the selection predicates which is depicted as dotted symbols. An exclamation mark denotes a change in data representation, e.g., the introduction of the *alert* element at SP_2 in the result for the query at P_0. In our example, the introduction of the *gas* element in the answer for the query at P_2 is supposed to take place at P_2 itself and therefore does not show up in the network. The decision whether to perform the FluX subscription evaluation at P_2, SP_1, or SP_2 is made by the optimizer and is based on factors like computational power and current load factor of peers.

The sample query evaluation plan in Figure 4 depicts the situation after the data stream and the two queries of Section 1 have been registered in the network of Figure 1. Furthermore, the query optimizer has already optimized the queries and integrated them into the system. First, the elements `bt`, `pr`, and `et` are removed from the stream by a projection operator. To reduce network traffic, the optimizer chooses to install the mobile code of the appropriate projection operator as close to the data source as possible. Since the data source P_4 is a simple sensor without query processing capabilities and is therefore not able to perform the projection by itself, the projection operator has to be installed and executed in the network at super-peer SP_3. The resulting data stream is routed only once (as one data stream cluster) to SP_2, although it is needed twice in the system. Therefore, the optimizer decides to replicate the data stream at SP_2 to obtain two identical versions of the stream. The decision of how to route and where to replicate the stream is simply made by pursuing the goal of minimizing the number of hops each stream has to go from its source to its recipient in the network. Of course, more sophisticated optimization goals and routing strategies can be employed here. We will examine this in future work. At

SP_2, the stream with destination P_2, which is the fire department, is again reduced by a projection operator removing element os. The remaining stream is forwarded to P_2 via the shortest path, in this case over SP_1. The rest of the query evaluation, consisting of the introduction of the *gas* element, is performed at P_2 itself. The stream with destination P_0 is also filtered at SP_2, this time using a projection and a selection as demanded by the respective query. Also, the new *alert* element in the query result is already introduced at SP_2. The resulting stream is then forwarded to P_0, again using the shortest path which is via SP_0. In general, the shortest path is not unique and depends on the underlying network topology. In the case of multiple shortest paths, one appropriate path among them is chosen.

Continuing our example from Section 1, we now take a look at a more complicated situation. Let us assume that peer P_1 represents a collection of weather sensors delivering a virtual data stream registered at super-peer SP_0. Each sensor reading contains the identifier of the corresponding sensor (`id`), a timestamp (`time`), the GPS coordinates of the sensor (`x`, `y`), and measurements for wind (`wind`), temperature (`temp`), humidity (`hum`), and air pressure (`ap`). Sensor readings for wind consist of wind strength (`strength`) and wind direction (`direction`). The resulting data stream corresponds to the following DTD.

```
<!ELEMENT reading (id, time, x, y,
                   wind, temp, hum, ap)>
<!ELEMENT id (#PCDATA)>
...
<!ELEMENT wind (strength, direction)>
<!ELEMENT strength (#PCDATA)>
<!ELEMENT direction (#PCDATA)>
...
```

We now further assume that the fire department at P_2 registers a new query at SP_1 in addition to the one already registered in Section 1. This new query requires the data from P_4 to be joined with data from P_1. The fire department is interested in finding out how strong and from which direction the wind blew at the point in time and at the place a gas concentration was measured. Therefore, it joins the data of the gas sensors from P_4 with that of the weather sensors from P_1. The join tries to find for each measured gas concentration a sensor reading for wind strength and direction that was close to the gas measurement in terms of both, the point in time the respective sensor readings where created and the geographical location at which the corresponding sensors where located. This can be achieved by using the bestmatch join operator [17].

One possibility to compute the join would be to filter P_1's data stream accordingly at SP_0 and route the resulting stream directly to SP_1, where the join processing takes place and the result gets delivered to P_2. This would probably be the best solution if no data from peer P_1 is needed anywhere else in the network. However, when P_3 also requests data from P_1, it might be better to route a data stream with the data for both P_2 and P_3 from SP_0 to SP_2 first and then split the stream at SP_2, routing P_3's part directly to P_3. The remaining stream for peer P_2 could then be routed to SP_1, where the join processing could take place. But if the join is known to produce a relatively small result compared to the input streams, it would probably be better in terms of network traffic to process the join already at SP_2 and then route the result to P_2 via SP_1. This is an example of a more difficult decision that has to be made by the StreamGlobe query optimizer.

4.2 Query Processing

Let us now outline some basic concepts used for in-network query processing. Query execution in StreamGlobe focuses on processing streaming data and therefore employs *push-based* evaluation strategies—in contrast to traditional query engines where data is normally "pulled" from subordinate operators, e.g., by using the iterator model.

First, we will explain how filtering operators are executed. As outlined before, filtering operators perform a projection of a data stream on the required parts of the entire schema and a selection according to predicates of a subscription rule. Since the basic schema of the original data stream remains the same[1] (besides discarding unnecessary information), projection can be done on-the-fly without the need of buffering parts of the data stream. Performing selections is somewhat more difficult, because in the worst case data cannot be propagated before the predicate is evaluated, which renders buffering inevitable. Thus, we restrict filtering operators to only employ predicates referring to a single data object of the data stream. Therewith, at most the current data object has to be buffered for being able to propagate the filtered data stream. Hence, we can implement these operators scalably and efficiently using automata-based techniques as described in [18] or the new FluX query engine which was developed in cooperation with our group and will be sketched in the remainder of this section.

In order to evaluate subscription rules on data streams, we employ novel optimization techniques, called *FluX* [19], for minimizing memory buffer consumption during the execution of XQueries on streaming data. FluX is an intermediate language extending the XQuery syntax by event-based processing instructions which enables conscious handling of main memory buffers. The key idea of the FluX query language is the novel *process-stream* statement { `ps` $\$x$: ζ } for event-based (streaming) processing of a substream assigned to a variable $\$x$. It processes the data stream by means of a list of *event-handlers* ζ. Each event handler is of one of the two forms

- `on` a `as` $\$y$ `return` α
- `on-first past(`S`) return` α

with α being an arbitrary subexpression, a being the label of a tag, and S being a set of labels of XML tags. An "`on` a" handler is executed if an opening tag labeled a is encountered in the stream of $\$x$. The subsequent elements

[1] In particular, the order of elements is preserved.

of the data stream are labeled as a substream $\$y$ and used to evaluate the subexpression α (which may in turn be a process-stream statement or traditional XQuery). The latter "`on-first`" handler is executed if no more elements labeled s with $s \in S$ will be encountered in the stream being currently processed and triggers the evaluation of α. In general, an arbitrary query cannot be evaluated purely on-the-fly without buffering, e.g., if the sequence of elements in the query is different from that in the input data stream. Hence, a FluX query consists of a purely streaming part using our novel syntax and of embedded traditional XQuery, which is evaluated on previously buffered parts of the data stream. The main challenge is to rewrite an XQuery into a corresponding FluX query which evaluates this query using as many of the event-based methods as possible and thereby minimizing buffer usage. In [19], an algorithm which utilizes order constraints on the elements imposed by the DTD of the data stream is presented to achieve this goal.

Rewriting XQuery into FluX is based on generating a *safe* FluX query. That is, an XQuery subexpression of a FluX query operating on buffered data must only reference—e.g., by path expressions or other variables—parts of the data stream which will not be encountered any more after this expression has been evaluated. Thus, the query engine can easily populate buffers with the needed parts of the data stream and provide these buffers for the execution of the buffer-based parts of the FluX query. The second query of our example scenario using the given DTD would be rewritten into FluX as follows.

```
{ps stream("firefighters")
  on reading as $m return
    {ps $m:
      on-first past() return <gas>;
      on id as $id return {$id};
      on x as $x return {$x};
      on y as $y return {$y};
      on CO2 as $CO2 return {$CO2};
      on SO2 as $SO2 return {$SO2};
      on-first past(*) return </gas>; } }
```

This FluX query is purely event-based (outputting the values of the substreams in the "`on`" handlers can be done on-the-fly) and hence needs no buffering at all. "`on-first past(*)`" is a shortcut for the set S containing all possible labels in this substream and is therefore executed after all other elements have been written. More details on FluX together with an experimental evaluation can be found in [19].

Summarizing, FluX enables query evaluation on data streams with very low memory consumption and thus provides for a scalable evaluation of subscription rules. However, some subscription rules might possibly need unbounded buffering, e.g., subscriptions containing joins or special aggregates. In such cases, unbounded buffering is precluded by requiring users to specify window constraints. These allow for a scalable execution on infinite data streams.

5 Implementation Status

As of the writing of this paper, we have implemented the basic infrastructure of StreamGlobe, building on the Globus Toolkit, and we are able to establish an overlay P2P network between peers. We have also completed a prototype implementation of the FluX streaming query engine for evaluating subscription rules. This query engine is currently being integrated into the StreamGlobe system. At the moment, the optimization techniques of Section 4 are developed and implemented. A first prototype system of StreamGlobe including all the basic features presented in this paper will be operational by the end of the year.

6 Conclusion and Future Work

In this paper, we have described the ongoing development of our StreamGlobe system. StreamGlobe is focused on meeting the challenges that arise in processing data streams in an ad-hoc P2P network scenario. It differs from other data stream systems in not only efficiently locating and querying data streams, but also optimizing the data flow in the network using expressive in-network query processing techniques. This is basically achieved by pushing operators for query processing into the network. Continuous reoptimization leads to an adaptive and self-optimizing system which enables users to carry out powerful information processing and retrieval. StreamGlobe builds on and extends the Globus Toolkit, a reference implementation of the Open Grid Services Architecture (OGSA) for grid computing, and serves as a research platform for our future work.

Future research will cover further topics in query processing on streaming data, optimization methods for distributed data stream processing, load balancing and quality-of-service aspects [4] in a distributed data stream management system. In detail, this will include augmenting the FluX query engine to support windowing operators like aggregations and joins. It will also comprise improving the optimization component by taking into account reorganisation issues to keep the system effective as well as synchronization aspects, e.g. for distributed join processing on various streaming inputs. Furthermore, we will continue to examine routing approaches for our hierarchical network organisation and conduct advanced research concerning the combination of multiple query processing operators, predicate comparisons in the context of query clustering, and the minimization of memory requirements during query evaluation. Eventually, support for content-based query subscriptions will be added to StreamGlobe.

Acknowledgments. Franz Häuslschmid and Angelika Reiser provided helpful comments on earlier revisions of this paper. Christoph Koch, Stefanie Scherzinger, and Nicole Schweikardt realized together with one of the authors the FluX [19] query engine.

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-

organizing structured P2P system. *ACM SIGMOD Record*, 32(3):29–33, Sept. 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, Mar. 2003.

[3] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal*, 10(1):48–71, Aug. 2001.

[4] R. Braumandl, A. Kemper, and D. Kossmann. Quality of Service in an Information Economy. *ACM Transactions on Internet Technology*, 3(4):291–333, Nov. 2003.

[5] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner. Distributed Queries and Query Optimization in Schema-Based P2P-Systems. In *Proc. of the Intl. Workshop On Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, Sept. 2003.

[6] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 215–226, Hong Kong, China, Aug. 2002.

[7] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, Dallas, TX, USA, May 2000.

[9] L. Chen, K. Reddy, and G. Agrawal. GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In *Proc. of the IEEE Intl. Symp. on High-Performance Distributed Computing*, Honolulu, HI, USA, June 2004. To appear.

[10] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable Distributed Stream Processing. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

[11] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan, and G. Agrawal. The BEA/XQRL Streaming XQuery Processor. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 997–1008, Berlin, Germany, Sept. 2003.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, June 2002. http://www.globus.org/research/papers/ogsa.pdf.

[13] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The Intl. Journal of Supercomputer Applications and High Performance Computing*, 15(3):200–222, Aug. 2001.

[14] The Globus Alliance. http://www.globus.org.

[15] Q. Huang, C. Lu, and G.-C. Roman. Spatiotemporal Multicast in Sensor Networks. In *Proc. of the Intl. Conf. on Embedded Networked Sensor Systems*, pages 205–217, Los Angeles, CA, USA, Nov. 2003.

[16] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. A Publish & Subscribe Architecture for Distributed Metadata Management. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, pages 309–320, San José, CA, USA, Feb. 2002.

[17] A. Kemper and B. Stegmaier. Evaluating Bestmatch-Joins on Streaming Data. Technical Report MIP-0204, Universität Passau, 2002.

[18] C. Koch and S. Scherzinger. Attribute Grammars for Scalable Query Processing on XML Streams. In *Proc. of the Intl. Workshop on Database Programming Languages*, pages 233–256, Potsdam, Germany, Sept. 2003.

[19] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. Schema-based Scheduling of Event Processors and Buffer Minimization on Structured Data Streams. In *Proc. of the Intl. Conf. on Very Large Data Bases*, Toronto, Canada, Aug. 2004. To appear.

[20] J. Krämer and B. Seeger. PIPES - A Public Infrastructure for Processing and Exploring Streams. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 925–926, Paris, France, June 2004.

[21] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 95–104, San José, CA, USA, May 1995.

[22] S. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries over Streams. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 49–60, Madison, WI, USA, June 2002.

[23] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized View Selection and Maintenance Using Multi-Query Optimization. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 307–318, Santa Barbara, CA, USA, May 2001.

[24] OGSA-DAI. http://www.ogsadai.org.uk.

[25] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP – Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In *Proc. of the Intl. Workshop on Agents and Peer-to-Peer Computing*, pages 112–124, Bologna, Italy, July 2002.

[26] T. K. Sellis. Multiple-Query Optimization. *ACM Trans. on Database Systems*, 13(1):23–52, Mar. 1988.

[27] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 539–550, Paris, France, June 2004.

[28] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, pages 49–60, Bangalore, India, Mar. 2003.

[29] Y. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 136–145, Athens, Greece, Aug. 1997.

[30] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *ACM SIGMOD Record*, 31(3):9–18, Sept. 2002.

[31] Y. Yao and J. Gehrke. Query Processing for Sensor Networks. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

Active Rules for Sensor Databases

M. Zoumboulakis, G. Roussos and A. Poulovassilis

Birkbeck University of London
Malet Street WC1 7HX
London UK
{mz, gr, ap}@dcs.bbk.ac.uk

Abstract

Recent years have witnessed a rapidly growing interest in query processing in sensor and actuator networks. This is mainly due to the increased awareness of query processing as the most appropriate computational paradigm for a wide range of sensor network applications, such as environmental monitoring. In this paper we propose a second database technology, namely active rules, that provides a natural computational paradigm for sensor network applications which require reactive behavior, such as security management and rapid forest fire response. Like query processing, efficient and effective active rule execution mechanisms have to address several technical challenges including language design, data aggregation, data verification, robustness under topology changes, routing, power management and many more. Nonetheless, active rules change the context and the requirements of these issues and hence a new set of solutions is appropriate. To this end, we outline the implications of active rules for sensor networks and contrast these against query processing. We then proceed to discuss work in progress carried out in project Asene that aims to effectively address these issues. Finally, we introduce our architecture for a decentralized event broker based on the publish/subscribe paradigm and our early design of an ECA language for sensor networks.

1 Introduction

Application development for sensor and actuator networks presents unique challenges since it has to address the complexities of distributed and often decentralized operation, the highly resource constrained nature of network nodes and the highly transient nature of network topology [4]. Moreover, applications must operate unattended for prolonged periods of time and still maintain their integrity and quality of service.

In recent years it has become clear that the investigation of higher level computational paradigms is necessary so as to abstract the complexity of systems development and offer application developers with a more amenable programming framework. To this end, query processing has attracted considerable interest and is rapidly becoming a popular computational paradigm for a plethora of sensor network applications. This approach has been seen to address well the complex requirements of application development in sensor networks in a variety of applications including environmental monitoring, distributed mapping and vehicle tracking [5, 12]. Prototype sensor network query processors have been implemented in Tiny DB [11] and Cougar [17] systems.

In this paper we argue that another database technology that may provide an appropriate computational model for a distinct set of sensor and actuator network applications is *event-condition-action (ECA) rules* [15] (also referred to as active rules). Indeed, sensor and actuator network applications often operate in one of either modes:

- in event-driven applications, for example detection of forest fires, security management or product detection in ubiquitous retailing [9], the system remains inactive until an event is generated in one of the nodes; then the event propagates through the system which subsequently initiates appropriate actions in response to this event,
- in demand-driven applications, for example environmental monitoring [5], activity is initiated in response to external requests, usually in the form of queries.

While query processing matches well the characteristics of the later class of applications, an ECA rule-

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

based approach offers a better fit for applications with execution profile that corresponds to the first pattern above. In such applications, the system needs to provide a timely response to events and although in principle this would still be possible using a sensor network query processor, its deployment would unnecessarily consume the limited resources by regularly checking for events that may not have occurred.

In the following Section we discuss ECA rules as a computational model for reactive systems with particular reference to sensor and actuator networks. We then proceed to compare more traditional ECA technologies with the novel needs of sensor networks. In Section 4, we discuss the requirements of ECA rules in this context and highlight the differences to the more well established sensor network query processors. Finally, we introduce the architecture of the Asene system for Active SEnsor NEtworks. We conclude with a discussion of work in progress and major challenges ahead.

2 Active Rules as a Model for Computation in Sensor Networks

We begin by examining in more detail the structure of a reactive sensor and actuator network application. A reactive application must be able to detect the occurrence of specific events or changes in the network state, and to respond by automatically executing the appropriate application logic. For example, in a security monitoring scenario sensors capable of detecting specific chemicals are deployed in the area under observation, for example a customs and excise enclosure in a port area. In addition to the sensor nodes, a smaller number of actuator nodes are also deployed with the capability to trigger alarms when activated. In this case, there is little scope for fixed network infrastructure due to the transient nature of most objects within the enclosure and the use of heavy machinery. The aim is to program the application so that when specific events are observed and specific conditions are met the network reacts in a predetermined way, for example when the concentration of particular chemical factors are observed and their concentration exceeds a set threshold within a small area the alarm in this and neighboring areas are activated.

ECA rules [15] are one way of implementing this kind of functionality. An ECA rule has the general syntax

```
on event
if condition
do actions
```

The *event* part specifies when the rule is triggered. The *condition* part is a query which determines if the sensor network is in a particular state, in which case the rule fires. The *action* part states the actions to be performed if the rule fires. A side effect of these actions may be that further events are generated, which may in turn cause more ECA rules to fire.

In sensor and actuator networks in particular the action part of an ECA rule may be either logical or physical. For example, the action may be to signal an actuator node to activate the alarm, or it may be a notification for a node to initiate a particular control sequence [16].

There are several advantages in using ECA rules to implement this kind of functionality compared to direct implementation in application code [2, 14]:

- ECA rules allow an application's reactive functionality to be specified and managed within a rule base rather than being encoded in diverse programs, thus enhancing the modularity, maintainability and extensibility of applications.
- ECA rules have a high-level, declarative syntax and are thus amenable to analysis and optimization techniques which cannot be easily applied if the same functionality is expressed directly in application code.
- ECA rules are a generic mechanism that can abstract a wide variety of reactive behaviors, in contrast to application code that is typically specialized to a particular kind of reactive scenario.

To illustrate the use of active rules to model reactive functionality we note that the application logic described at the beginning of this section could be encapsulated within the following rule

```
ON UPDATE toxicity
IF AVG(toxicity) > thres WITHIN radius r1
DO ACTIVATE alarm WITHIN radius r2
```

3 Sensor Networks and Traditional Active Database Systems

ECA rules in the context of a sensor and actuator network present a number of novel challenges against the traditional database view [8]. In traditional active database (and web-based) systems the condition and action parts of an ECA rule are most often tightly coupled, that is the execution model of a particular rule is [E][CA]: a database object is monitored and when modified in a predetermined way an event is generated. Rules whose event parts match this event are then triggered and, if their conditions hold, their actions are scheduled for execution. In all cases, execution of the condition query and the action part is driven by the same application logic. Hence, ECA functionality is tightly coupled and coordinated. Moreover, such systems are generally administered by database experts and often implement advanced failure-tolerance features, including clustering, power backups and replicated communication channels.

Sensor and actuator networks consist of a large number (often several hundreds) of loosely-coupled node elements [1]. Each node operates fairly independently and can make its own decisions about its wake-up/sleep cycle and the data it accepts to forward to nearby nodes. Nodes may also have different capabilities, for example sensors may be able to detect temperature, humidity, changes in the magnetic field of the Earth, different types of biosensing and so on. Actuators may be biomanipulators, microvalves and micropumps or they can simply be electrical switches. In addition to sensor and actuator nodes, nodes that have the sole purpose of providing communications and computational assistance may also be introduced in the system. In all cases, nodes will have high failure rates which may result in network fragmentation, that is the separation of network segments into isolated islands of system functionality.

Sensor and actuator networks are deployed in ad-hoc ways and thus the resulting topologies may be highly irregular and with highly heterogeneous density and connectivity patterns. Furthermore, the topology may often change rapidly during its pre-deployment, deployment, and re-deployment phases and possibly at very high speed. This is in stark contrast to traditional database management systems which assume that connectivity is fairly fixed and network topology is rarely of concern and dealt with outside the database management system.

Last but not least, sensor and actuator nodes are very limited in power, computational capability and holding capacity and are normally unavailable for regular repair or frequent battery recharge. Although Moore's law predicts that node capabilities will increase rapidly, they will always be less powerful than other embedded, portable or hand-held computing devices and most importantly battery power available for their operation will remain limited for the foreseeable future.

4 Challenges for Active Functionality in Sensor Networks

In this paper we propose that ECA rules can provide a natural computational paradigm to sensor and actuator network applications that require reactive behavior. While sensor network query processors (SNQP) [3, 5, 11] have proven very successful in providing appropriate abstractions for user interaction, ECA rules address the problem of unattended system behavior and can effectively model application logic in autonomic situations[1]. In the context of such applications, the system is required to provide a timely response to events at the lowest communications and computational cost. Although potentially a SNQP could be used for this type of application, in practice it would unnecessarily consume limited resources by regularly checking for events that may not have occurred. Indeed, SNQPs primarily address data acquisition from a relatively small number of vantage points. ECA rules may provide an effective and efficient mechanism to support reactive behavior by localizing control and by providing a mechanism to react to events rather than proactively test whether a particular event has occurred.

This difference in scope between SNQP and ECA rules implies that the two systems have very different execution profiles which also means that they also have very different requirements. In the following paragraphs we attempt to outline the most critical differences between the two approaches and in the following section we discuss our current work in trying to address the novel requirements of ECA execution within project Asene.

- **Vantage Points.** SNQPs assume that queries are initiated at a single or a relatively small number of vantage points, with data aggregation potentially carried out at a few intermediate locations, the so-called storage points. In ECA rules any sensor in the network may generate an event which may be used by any actuator also potentially placed at any network location. Thus, an ECA rule may fire at any node location within the network and may also activate any node within the network.

- **Communication Pattern.** SNQPs collect data in regular patterns which sensor nodes can use to synchronize and agree on wake-up/sleep cycles. ECA rules are reactive and thus rules fire at unpredictable, irregular intervals. Hence, wake-up/sleep schemes that can support this asynchronous mode of operation are required. Moreover, this irregular pattern implies that nodes consume power at different rates and for this reason node failure is more irregular and harder to predict.

- **Routing.** SNQPs currently mostly use tree-based routing mechanisms that flood the network at least once, during the tree construction stage. In this context the communications overhead placed by the route discovery stage is justified by the relatively large amount of data that is being collected. An ECA rule processor is characterized by small, incremental updates rather than a single data collection step and thus

[1]The scope of active functionality as described here should not be confused with the so-called event queries supported by Tiny DB. Event queries aim at providing user control over data acquisition so that users can register their interest for specific results returned by an acquisitional query and specify additional queries that should be carried out in response. Hence, supporting generic reactive functionality is well beyond the scope of event queries.

the route discovery stage of tree-based algorithms would dominate the communications cost. Consequently, globally optimal routes would probably not optimize power consumption for the network as a whole and localized routing algorithms could be more efficient [7].

- **Data Model.** SNQPs currently view the sensor network as a single data space. ECA rules require an alternative data model which distinguishes between the different types of objects that are being observed and generate events. In the following section we propose a mechanism for the construction of separate data spaces based on the so-called topic channels.

- **Aggregation.** Aggregation in ECA is carried out at the signal rather than the query layer which is the norm for SNQP. Although the mathematical techniques used for aggregation in SNQP [6] can also be used in ECA rule processing, this is done at a lower layer and within a particular topic channel in an approach akin to collaborative signal processing in distributed environments.

- **In-network storage.** Although both systems clearly benefit from in-network storage, SNQP develops hierarchical-directional mechanisms based on the tree-based routing algorithms employed, whereas ECA rules benefit from decentralized-flat and schemes at the topic channel level.

- **Network Segmentation.** ECA rules execute within the a specific network locality and thus can be relatively resistant to network segmentation for example due to loss of connectivity caused by intermediate node failure. ECA rules may still fire despite their isolation from a sink controller.

5 A System Architecture for ECA in Sensor Networks

One of the major challenges in implementing an ECA rule based architecture for sensor and actuator networks is the distribution of events in a computationally efficient manner. In this section we introduce the Asene approach to support ECA functionality in sensor and actuator networks.

Asene is built on top of *event channels* which are also viewed as data object primitives. An event channel has two elements: a collection of nodes that monitor the same attribute and associated algorithmic mechanisms that coordinate node operation. Within an event channel nodes carry out collaborative signal processing and data aggregation and are responsible for in-network storage and event generation. Finally, the node components of an event channel encapsulate internal structures that maintain shared descriptions of the channel.

Event channels are also responsible for the distribution of events following the so-called publish/subsribe (P/S) paradigm [13]. P/S systems are commonly used to bring together data sources and information consumers by transparently delivering events from the first to the second. In Asene, event channels are responsible for maintaining a list of subscribers to the particular event and for sending notifications. Thus, subscriber nodes may move freely and re-attach to the channel at alternative locations. Effectively, an event channel functions as a decentralised event broker following the P/S jargon.

The particular characteristics of sensor and actuator networks make them especially compatible with the P/S paradigm, in particular with regard to the need for in-network storage and processing:

- P/S systems are characterized by the same basic properties as sensor and actuator networks; that is, communication is anonymous, inherently asynchronous and multicasting in nature. P/S systems are also capable of quickly adapting to changing network topologies.

- P/S systems can support the decentralized operation of event management and delivery, transparently for both sensor and actuator nodes. This is particularly important since computation in sensor and actuator networks is highly asymmetric and thus local adaptability and local control is of great importance.

- The P/S anonymity property in particular implies that communicating nodes are not required to identify the party they wish to communicate with (that is, subscribers need only describe the characteristics of the events they want to receive instead of naming a specific publisher to receive events from) and thus data aggregation may be implemented transparently for the end application. Moreover, the anonymity property implies that flexile wake-up/sleep cycles can be developed since delivery of events to subscribed recipients does not depends on a single sensor node.

- Conceptually P/S systems deliver events to multicast groups, a communications mode that is a good fit for the provision of incremental updates to aggregation operators constructed on top of role-based spatial hierarchies of sensor and actuator networks nodes. The power saving potential of these multi-resolution data aggregation schemes can be considerable and more importantly their effectiveness increases rapidly with the number of nodes in the system. Moreover, it is possible to achieve relatively high performance by using the periodic beaconing performed of most medium access and topology control protocols for update delivery across a particular topic channel.

The properties that make P/S suitable for use in sensor and actuator networks also suggest a natural way to support node failures as a feature of the system rather than as a fault. Indeed, in this context data aggregation is performed independently by each node. Hence, loss of updates will affect accuracy locally and nodes will continue computation with whatever data available, on a best effort basis. This is a distinct advantage over techniques originating from more tightly coupled systems, where there would be a need for roll backs and data cleansing operations which are not appropriate in the case of sensor and actuator networks.

One of the expected advantages of this architecture is that it allows for complex wake/sleep schemes while at the same time maintaining a good quality of service via replication of the in-network stored data and of the subscription information.

The use of event channels as the core building block for Asene allows for the full decoupling of the [E], [C] and [A] components of ECA rules. Also note that queries associated with the condition part of an active rule can be answered locally and in some cases the data required could be disseminated at the same step as the event itself. It is also worth observing that new functionality can be introduced in the system via the simple insertion of new condition nodes, that is nodes that are responsible for checking for specific conditions in response to event notifications. Finally, constructing activation channels is also a viable alternative although often the expected number of actuator nodes would be much smaller than the number of sensors and it is probably not as cost efficient as an approach.

5.1 Heterogeneity

An interesting observation on the effects of the Asene architecture is that significant operational benefits may be achieved if heterogeneous sensor and actuator networks are constructed. Heterogeneity in this case is seen primarily in communication capability and in terms of the range of communication. Inserting a few nodes that have longer range capabilities (but also higher power consumption) can significantly increase the robustness of the event channel by increasing the connectivity across node clusters.

5.2 Composite Events

Using event channels as the main mechanism for data dissemination also suggests a clear way for constructing rules with composite events: the condition node needs only subscribe to all corresponding event channels. Compare this against the difficulty of dealing with multidimensional data in the context of SNQP.

6 Discussion and Conclusions

In this paper we have argued that, in addition to query processing, ECA rules is a database technology that may provide an appropriate computational model for a distinct set of sensor and actuator network applications. However, ECA rules in this context present several challenges which we highlighted in previous sections. We have also introduced Asene, an ongoing research project that aims to establish ECA rules as the common mechanism for the description of reactive functionality in sensor and actuator networks.

The current version of Asene supports simple event channels built on top of Tiny OS [10] primitives and a simple ECA language. We are currently developing further our algorithms for the efficient construction of event channels in sensor networks. Our focus is on a single-step approach that identifies all members of all registered event channels in a particular network and thus removes the need for duplication of the bootstrap phase. We are also improving on the data structures used to represent the internal state of a particular event channel and maintain the list of active subscriptions. Our work aims to balance the need for low communication between nodes and the asynchronous nature of event generation with regard to the wake-up/sleep node cycles. We are planning to conduct extensive experiments with the prototype implementation to better understand the tradeoffs involved.

In addition to the development of efficient and effective event channel management mechanisms, a second major objective of the Asene project is the definition of an appropriate lightweight ECA language that satisfies the requirements of the application domain. The brief example presented in Section 2 in the context of a security management application is taken from the current version of Asene. Clearly, further work in understanding the performance implications of the different constructs is required and balanced against language expressivity.

The next step for Asene is the integration of advanced aggregation algorithms and the study of localized routing algorithms for event dissemination. In doing so we favor a multi-resolution approach similar to the aggregation schemes discussed in [6] but more appropriate for the structure of our event channel construction algorithms. Finally, we intend to further investigate the relative merits of different routing strategies for event dissemination based on localized network descriptions. We anticipate both approaches to offer significant reduction in resource demands from the network.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci. Wireless Sensor Networks: A Survey, *Computer Networks*, Vol. 38, pp. 393–422, 2002.

[2] J. Bailey, G. Papamarkos, A. Poulovassilis and P. T. Wood. An Event-Condition-Action Rule Lan-

guage for XML, in A. Poulovassilis and M. Levene (eds.) *Web Dynamics*, Springer-Verlag, to appear, 2004.

[3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems, *Proceedings of the Second International Conference on Mobile Data Management*, Hong Kong, 2001.

[4] D. Estrin, R. Govindan, J. Heidemann and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks, *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, Washington, USA, pp. 263-270, 1999.

[5] J. Gehrke and S. Madden. Query Processing in Sensor Networks, *IEEE Pervasive Computing*, Vol. 3, No. 1, pp. 46-55, 2004.

[6] J. M. Hellerstein , W. Hong, S. Madden, K. Stanek. Beyond Average: Toward Sophisticated Sensing with Queries, F. Zhao and L. Guibas (eds.) *Proceedings of Second International Workshop Information Processing in Sensor Networks*, IPSN 2003, Palo Alto, CA, USA, April 22-23, pp. 63 - 79, 2003.

[7] A. Helmy. Location-free Contact Assisted Poer-Efficient Query Resolution for Sensor Networks, *Mobile COmputing and Comuunications Review*, Vol. 8, No. 1, pp. 27-47, 2004.

[8] K. Kulkarni, N. Mattos, and R. Cochrane. Active database features in SQL3, in N. Paton (ed.) *Active Rules in Database Systems*, Springer-Verlag, pp. 197-219, 1999.

[9] P. Kourouthanassis and G. Roussos. Developing Consumer-Friendly Pervasive Retail Systems, *IEEE Pervasive Computing*, Vol. 2, No. 2, pp. 32–39, 2003.

[10] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer and D. Culler. The Emergence of Networking Abstractions and Techniques in TinyOS, *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, March 29-31, San Fransisco, CA, 2004.

[11] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. The Design of an Acquisitional Query Processor for Sensor Networks, *SIGMOD*, San Diego, CA, 2003.

[12] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad hoc sensor networks, in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[13] C. Mascolo, L. Capra, and W. Emmerich. Middleware for Mobile Computing (A Survey), *Lecture Notes in Computer Science*, Vol. 2497, 2003.

[14] G. Papamarkos, A. Poulovassilis and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web, *Proc. VLDB'03 Workshop on Semantic Web and Databases*, Berlin, September 2003.

[15] N. Paton and O. Diaz. Active Database Systems, *ACM Comp. Surveys*, Vol. 31, No. 1, pp. 63–103, 1999.

[16] Y. Yemini, A.V. Konstantinou and and D. Florissi. NESTOR: An Architecture for Self-Management and Organization, *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 5, pp. 758–766, 2000.

[17] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks, *Sigmod Record*, Vol. 31, No. 3, 2001.

A Framework for Spatio-Temporal Query Processing Over Wireless Sensor Networks

Alexandru Coman Mario A. Nascimento Jörg Sander

Department of Computing Science
221 Athabasca Hall
University of Alberta
Edmonton, Canada
{acoman,mn,joerg}@cs.ualberta.ca

Abstract

Wireless sensor networks consist of nodes with the ability to measure, store, and process data, as well as to communicate wirelessly with nodes located in their wireless range. Users can issue queries over the network, e.g., retrieve information from nodes within a specified region, in applications such as environmental monitoring. Since the sensors have typically only a limited power supply, energy-efficient processing of the queries over the network is an important issue. In this paper, we introduce a general framework for distributed processing of spatio-temporal queries in a sensor network that has two main phases: (1) routing the query to the spatial area specified in the query; (2) collecting and processing the information from the nodes relevant to the query. Within this framework, different algorithms can be designed independently for each of the two phases. We also propose novel algorithms for this framework, one for the first phase and two for the second phase. In an extensive experimental evaluation we study the performance of these algorithms in terms of energy consumption, under varying conditions. The results allow us to recommend the most energy efficient solution, given a network and a spatio-temporal query.

1 Introduction

Recent technological advances, decreasing production costs and increasing capabilities have made sensor networks suitable for many applications, including environmental monitoring, biological contamination detection, warehouse management, traffic organization and battlefield surveillance.

Today's sensors are no longer just simple sensing devices wired to a central monitoring site, but they have computation, storage and wireless communication capabilities. Most of these devices are battery operated, which highly constrains their life-span, and it is often not possible to replace the power source of thousands of sensors. Energy efficient data processing and networking protocols must therefore be developed to make the long-term use of such devices possible. While the network research community has studied energy efficient protocols in the context of ah-hoc networks, the database community has been confronted mostly with time and size constraints, but rarely with energy limitations. Therefore, the ability to apply traditional data processing techniques in sensor networks is limited, and different solutions must be found.

In this paper we focus on energy efficient query processing in sensor networks. In particular, we are interested in answering historical spatio-temporal queries such as "What was the humidity yesterday morning in Lake Annete area?". We study this problem in a sensor network where each sensor is only aware of the existence of the other sensors located within its communication range, and the query can be initiated locally at any sensor. There are two main reasons for allowing query initiation at any node. First, using only designated sensors as query originators causes an unbalanced energy use among sensor nodes, leading to faster energy depletion at the designated sensors, as well as the sensors located in their proximity, as these nodes would participate in the processing of most queries. Second, nodes could become unavailable for various reasons, such as energy depletion, hardware failure or hostile environment. To the best of our knowledge, historical query processing in such a sensor network environment has not been investigated before. The advantages of this environment are network robustness, a balanced use of sensors' energy resources and a wide range of application scenarios that can take advantage of the proposed solutions.

An application where such a sensor network enviroment can be used is micro-climate monitoring in national parks. The sensor nodes could be deployed from a plane over a forest area. Upon activation, each node would start observing periodically various physical phenomenons, such as temperature and humidity of air and soil. Park rangers patrolling through the forest can access the network through any node in their proximity using a notebook or iPAQ.

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

For instance, when certain events such as forest disease or small fires are observed, the ranger could query the network about historical observations, which may help him understand what have caused such events or learn about other areas that are threatened by similar events.

We propose the STWin framework for processing historical spatio-temporal queries in sensor networks, i.e., queries that specify the spatial area and temporal range the answers must belong to. As sensor nodes spend most of their energy supply during communication [1], we aim at minimizing the amount of data exchanged among nodes during query processing. Our framework has two phases. First, we search for a path from the query originator node to a sensor located within the query's spatial window. Second, the located sensor assumes query coordinator role, gathers the answers from all query relevant sensors and ships them back to the query originator. We use a greedy routing algorithm in the first phase, while for the second phase we propose two algorithms, one based on parallel flooding, the other using a depth first strategy.

In summary, the contributions of this paper are:

- we study the processing of spatio-temporal queries in a sensor network where each node only knows about the network nodes located within its wireless range;
- we introduce the STWin query processing framework;
- we propose three algorithms to be used within the STWin framework;
- we evaluate experimentally the performance of these algorithms and discuss their benefits and drawbacks.

The remainder of this paper is organized as follows. Section 2 describes some of the research work related to ours. Section 3 presents the sensor network settings as well as the characteristics of the query and data. Section 4 details our solution for spatio-temporal query processing. Section 5 presents the experimental evaluation of the proposed algorithms, and Section 6 concludes the paper.

2 Related Work

In this section we discuss a few works related to our current investigations. As sensor networks research lays at the intersection of networks, systems and databases, we describe a few projects addressing data management issues in the sensor environment from the plethora of publications addressing various aspects of sensor networks.

The Cougar project [22] is one of the most related to ours as it also investigates techniques for query processing over sensor data. However, unlike ours, their research focuses on a sensor networks environment where there is a central administration that knows the location of all sensors. In [4] the authors focus on defining a sensor database model for processing long-running queries, which are modelled as persistent views over the distributed sensor database. A central optimizer has the tasks of building a query plan and disseminating it to the relevant sensor nodes. In a similar environment but with emphasis on energy efficient query processing, they extend their work in [23] and analyze a wider range of problems, such as routing and crash recovery, basic query plans and in-network aggregation.

In [13], Madden et al. focus on query processing in a sensor environment where the information about the existing sensors is available in a catalog. Sensor nodes simply collect and transmit the raw data to the powered sensor proxies that are in charge of further processing and routing the answers to the users. The authors focus on minimizing the used energy by adapting the sensors' sampling and data package transmission rates. They introduce the Fjord architecture for management of multiple queries. Designed for Berkley Mica motes and running on top of TinyOS, TinyDB [15] is a distributed query processor that runs on each of the sensor nodes. The authors focus remains on optimizing data acquisition for long-running queries, no data being stored locally at the nodes. To reduce the energy consumption, they also propose TAG [14], an aggregation service for networks of TinyOS motes.

Beaver et al. [2] propose a solution for in-network aggregation, which exploits the temporal correlation in a sequence of sensor readings to reduces the energy used during query processing. Their solution, called TiNA, also allows the user to specify a temporal coherency tolerance when an approximate answer is sufficient, which lowers the energy costs. Similar to TinyDB and Cougar, TiNA is designed for a sensor environment where sensors simply forward their measurements to answer a long-running query, without storing any historical data.

Directed Diffusion [12] proposes a data-centric framework for query processing. Their sensor network environment is similar to ours in the sense that the query can be originated at any node, and nodes are only aware of their neighborhood. Different from us, nodes do not store historical data and sensing is only performed in response to a query request. In Diffusion, nodes request data by sending *interests* for specific data, which is then collected by the source nodes and shipped to the originator node. Intermediate nodes can cache and aggregate data, as well as direct new interests based on the cached data. The Directed Diffusion uses flooding to find paths from the query originator node to the data source nodes. Path reinforcements are used for selecting a small number of "good-paths" over which sensed data is returned. This scheme creates multiple paths for delivery, which increases the robustness of delivery at increased energy costs.

A system that focuses on query processing over historical data is DIMENSIONS [8, 9]. The authors focus on multi-resolution summarization of data using wavelet transform and construct a sensor hierarchy over the network. While temporal summarization is performed in each node, each level in the sensor hierarchy deals with another resolution of summarization. Their solution is suitable for data-mining, where a query can first look at the data at a coarse resolution and then focus on a region of interest at a finer resolution. The hierarchical scheme is applied in a grid sensor network where clusterheads (nodes storing coarser resolution in the sensor hierarchy) are dynamically selected based on their location in the grid.

The authors in [11] focus on sensor data processing, and propose solutions for data stream joins over the sensor data in tracking or monitoring application. The performance of their solution decreases sharply with increasing number of sensors, with more evaluation being required to establish the validity of their methods for large scale sensor deployments. In [6], the authors propose one of the first index

structures for sensor networks. The solution is based on the R-tree and it seems to be energy and time efficient, but no evaluation is presented. Xu and Lee [21] propose a window-based query processing technique in a network of moving sensors, where sensors only take measurements and provide data upon users' request. Though interesting, their solution has no experimental evaluation.

In the area of networks research, much work has been done in ad-hoc wireless networks. One of the most relevant issues for efficient query processing in sensor networks is position based routing, that is, network routing when the destination node is known and addressed by means of its location. We refer the reader to several surveys [16, 20, 10] on techniques for position based routing in ad-hoc networks. In our scenario, a sensor node is only aware of the network nodes located within its wireless communication range, which complicates the routing decisions. In such a case position based routing algorithms with guaranteed delivery cannot be readily re-used.

3 Background and Settings

We assume a sensor network with fixed nodes that have equal roles in the network's functionality. Each node has a CPU, long term storage, its own energy source and it is connected to other members of the network through wireless communication. All sensor's components have limited capabilities and the power source can be depleted quickly if not used efficiently.

Due to the wireless network characteristics, a node can communicate directly only with the sensors located within its wireless range, which form its neighborhood. A node can send a message individually to one of its neighbors, or it can use broadcasting to send the message simultaneously to all of its neighbors. When a message has to be sent to all or most neighbors, it is cheaper to broadcast the message than to send it individually to the neighbors. A sensor communicates with nodes other than its neighbors using a multi-hop routing protocol over the network. There are two main types of messages in query processing: query messages (which transport the query) and answer messages (which transport the query answers).

Each node knows its location (e.g., it may use GPS during node activation), as well as the location of its neighbors (collected during network activation). Sensor nodes may have several sensors, e.g., for temperature, humidity, magnetism, and light. In this paper we consider sensors that observe the state of a measured entity at the sensor location only. This is different from range sensing (e.g., movement sensing used in tracking [7, 19]), which measures the state of an entity not necessarily located at a sensor's position, but in its vicinity. Sensors take measurements periodically, and the collected values are stored locally for future querying. Data collection is performed continually, which can be viewed as an infinite stream. As infinite data cannot be fully stored, we adopt the stream storage solutions for fixed storage space proposed in [24]. The solution uses temporal aggregations over the data stream at multiple time granularities. The aggregation level for a data record is dependent on the age of the record, with only the most recent data fully stored. The aggregation levels and their granularity are decided before the network deployment and are dependent on the measured data and the storage size. Similar to any approximation technique, the adopted storage solution may not be able to provide useful data if the query requires high quality answers. Such stream storage solutions need to be used only when sensor nodes use small sampling rates or generate complex data per sample. For instance, a sensor node with 1 megabyte of memory measuring temperature once every 5 minutes could store more than 1 year of raw data, which is beyond the expected lifetime of some of the sensor devices currently available.

Each sensor node stores and manages locally its measurements. Each measurement has attached to it a timestamp corresponding to the time of measurement. Each type of sensor has associated a measurement interval, which defines the interval between successive data collections and is identical for all sensor nodes. We consider the data in the sensor network as a specialized distributed database, with temporal data stored in a node's database. Each node has a location, which gives spatial properties to data. Thus, on a global view, the sensor networks is a distributed database storing spatio-temporal data.

We are interested in processing historical spatio-temporal queries, denoted by *HSTQ(qID,sw,tw)*, where *sw* represents a spatial window, *tw* represents a temporal window and *qID* uniquely identifies a query. The answer to the *HSTQ* query is formed by all sensor measurements from the given area *sw* during the time range *tw*. Sensor nodes have equal capabilities and therefore a query can originate at any node with query answers located in some (possibly all) of the nodes. Some sensor network scenarios [15, 22] consider the so-called long-running queries, where a user wants the continuously monitor the measured entities. We do not consider this type of query in this paper.

4 Spatio-Temporal Query Processing

Given a historical spatio-temporal query *HSTQ(qID,sw,tw)* at a sensor node, the problem is to efficiently locate and retrieve the answers, given the limited knowledge each node has about the overall network. As a major constraint on sensor nodes is their limited energy supply, we focus on energy efficient techniques. It has been shown that the energy required by sensing and computation is up to three orders of magnitude less than the energy used for communication [17]. Therefore we use the energy cost of communication as the measure of efficiency. This cost is proportional to the number and the size of exchanged messages.

In this section we discuss first a basic query processing algorithm for sensor networks. Next, we present an original framework for processing spatio-temporal queries and, within this framework, we propose three new algorithms.

4.1 Basic Query Processing Algorithm

A straightforward way to locate the query answers, which we call FullFlood, is contacting every network node. The query originator node broadcasts the query to its neighbors, which in turn broadcast the query to their neighbors, and so on, until all nodes have received the query. Due to query message broadcast, each node will receive the same query several times. For each query, a node processes only the first query message received, discarding subsequent query messages. When a query is received, the node first broadcasts the query, then it selects the lo-

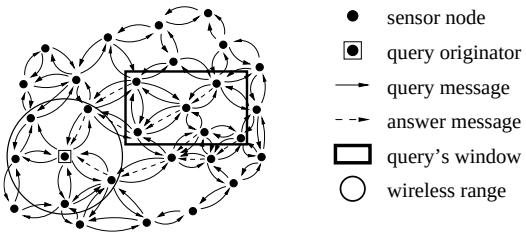


Figure 1: The FullFlood algorithm - message flow

cally stored data relevant to the query (if any), it waits for its neighbors' answers and merges them with its own, and finally it returns the answer to the neighbor that it received the query from. Once the query originator node has received the relevant data from all nodes, it can answer the query. The messages flow for FullFlood algorithm is shown in Figure 1.

The FullFlood algorithm is guaranteed to find the query answer for a connected sensor network, but it incurs high communication costs due to the large number of messages required to contact all nodes. The algorithm is similar to a parallel breadth first search in a network graph, where sensor nodes are vertices and edges represent direct communication links between sensors. Assuming there is no communication delay, the query will reach each node on the shortest path (with respect to number of hops) from the query originator. As query messages are broadcast along all paths, the first message reaching a node must have travelled over the shortest path. After a query is processed locally, each node returns the answer to the neighbor it first received the query from, and therefore answers are returned over the shortest path to the query originator.

4.2 Query Processing with STWin

If there is only one node relevant to the query, the optimal solution is contacting the node on the shortest path from the query originator and returning the answers over the same path. When the query answer involves several nodes, communicating with these nodes on the shortest path between the query originator and each of them is no longer optimal. Figure 2 shows an example. Forwarding the query over the shortest paths (routes (a) and (c)) requires 6 query messages in order to reach both relevant nodes, while route (b) requires only 5 messages. On the other hand, returning the nodes' answer over the shortest path is still optimal (assuming there is no aggregation of answers). As the energy usage is proportional to the message size and the same amount of answer data must be transferred over any of the possible return paths, sending the answers over the shortest path is the cheapest. Finding an optimal solution requires each network node to know the network layout, as well as possibly expensive local computation for finding the optimal route for each particular query. Due to sensors' limitations, it is not feasible for each node of a large sensor network to find and store the full network layout, as well as make expensive processing. On the other hand, contacting all sensor nodes as in FullFlood algorithm is not the most energy efficient approach.

A heuristic solution for query processing is contacting only the query relevant nodes, and a few extra nodes for routing the query and the answer if the query originator

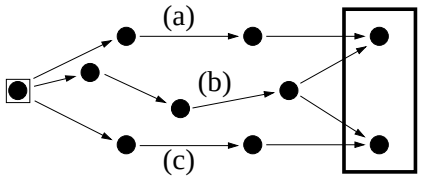


Figure 2: Query routing example

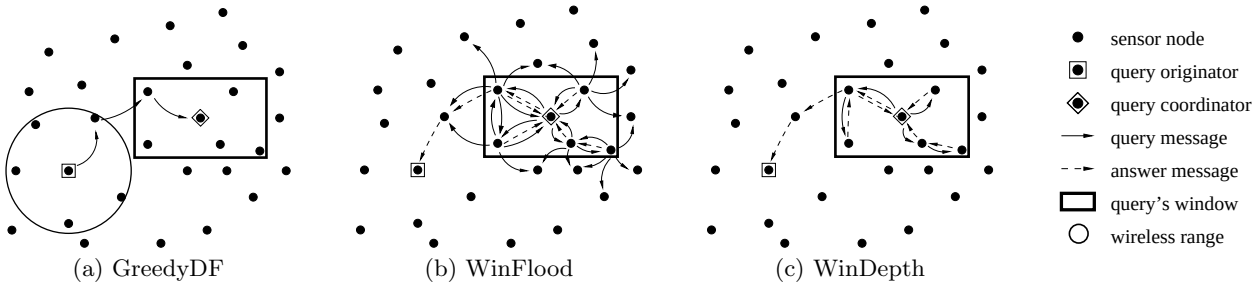
is not located inside the query's spatial window. A heuristic contacting only a subset of all network nodes should use a lower number of messages than FullFlood, which may lead to lower energy consumption. An additional advantage of such a solution is reduced network congestion, which improves the query response time. Also, if only a subset of the network nodes is involved in processing each query, then several queries could be efficiently processed simultaneously in different parts of the network. We propose the STWin (Spatio-Temporal WINdow) framework for query processing in which such a heuristic can be implemented. In this framework, we divide the query processing into two phases, one for locating a path from the query originator node to a sensor inside the query's spatial window, the other for gathering the query answer from the relevant nodes and returning it to the query originator.

- **Phase 1:** Given a query at a node N_Q, called query originator, we want to find a path to a node located in the query's spatial window. This node will assume query coordinator role N_C for Phase 2.
- **Phase 2:** The coordinator node N_C initiates the query processing within the query's spatial window. The processing algorithm must locate all relevant nodes, gather the results and return them to the query coordinator N_C. The coordinator will then return the answer to the query originator node N_Q on the routing path discovered in Phase 1.

These two phases form a general query processing framework, where various algorithms can be used in each phase. In the following we propose one algorithm for the first phase and two algorithms for the second phase.

4.2.1 Phase 1: GreedyDF

The *GreedyDF* algorithm uses a greedy technique to find a routing path from the query originator node to a node N_C located at the center of the query's spatial window. Other possibilities for choosing N_C exist, and which node is the best to select as coordinator for a query is an open question. Choosing the center node is a good compromise between the likelihood of a heuristic to find at least a node in the query area and the length of the path over which answers from the coordinator node will be returned to the query originator node. The query originator forwards the query to its neighbor located closest to N_C, which in turn forwards the query to its neighbor closest to N_C, and so on. If node N_C is found, then node N_C initiates Phase 2. The routing may reach a sensor node that is closer to N_C than any of its neighbors, in which case the query cannot be forwarded. If the reached node is located in the query's area, the node assumes coordinator role N_C and initiates Phase 2, else an empty answer is returned. The *GreedyDF* algorithm uses a small number of messages, but



(a) GreedyDF (b) WinFlood (c) WinDepth

Figure 3: The algorithms within the STWin framework - message flow

it does not guarantee that a routing path to a node in the query's spatial window will be found. Greedy-based routing methods for position based routing in ad-hoc networks have been shown to nearly guarantee delivery for dense network graphs, but to fail frequently for sparse graphs [20]. Variants to this heuristic would include using a different neighbor selection method or backtracking the search when query forwarding cannot be done. We choose to not use backtracking solutions as they cannot guarantee answer location within a small number of steps, while ultimately degenerating to a slow network flood with higher communication costs due to the extra messages required for the backtracking steps. The message flow for the *GreedyDF* algorithm is depicted in Figure 3(a).

4.2.2 Phase 2: WinFlood and WinDepth

For the second phase of STWin we propose two algorithms. The *WinFlood* algorithm consists of a constrained parallel flooding, where a node broadcasts the query to its neighbors only if its own location is inside the query's spatial window. The constrained flooding starts at the query coordinator node N_C and stops when the query reaches nodes outside the spatial window. Figure 3(b) show the message flow for the *WinFlood* algorithm. The *WinFlood* algorithms is similar to a window-constrained parallel breadth first search in the network graph.

An alternative solution is the *WinDepth* algorithm, which is based on the depth first search policy. In *WinDepth* each node may forward the query only to those neighbors located within the query's spatial window. When a node receives a query, it adds its node ID in the query header so that the query path is remembered. Then it selects a neighbor located within the spatial window that has not received the query yet (determined based on the query header), and forwards the query to this neighbor. When the neighbor returns the partial query answer, the node checks again if there is any other of its neighbors that is relevant to the query and has not received it yet. If there is such a neighbor, it forwards the query to this node and waits again for the neighbor's answer. This process is repeated until all of a node's neighbors located within the window have answered the query, at which point all the partial answers received are merged with the locally stored answers and the new partial answer is returned to the neighbor that the node received the query from. The message flow for the *WinDepth* algorithm is shown in Figure 3(c).

The *WinFlood* algorithm uses broadcast messages to forward the query, while in *WinDepth* nodes send individual messages to neighbors located within the window. As the cost of one broadcast message is generally lower than the cost for a group of one-to-one messages, it may be cheaper to use broadcasting and stop the query forwarding when an exterior node is reached. An advantage of *WinFlood* is that it is faster than *WinDepth* for the same number of contacted nodes and likely more cost efficient within a small window due to the use of broadcast messages. On the other hand, *WinDepth* contacts a smaller number of nodes, which makes more nodes available to answer other queries, and it causes less network congestion, which helps improve the query response time if several queries are processed simultaneously in the network.

In the following section we evaluate experimentally the proposed algorithms and discuss the effects of several factors on the energy used during query processing.

5 Experimental Evaluation

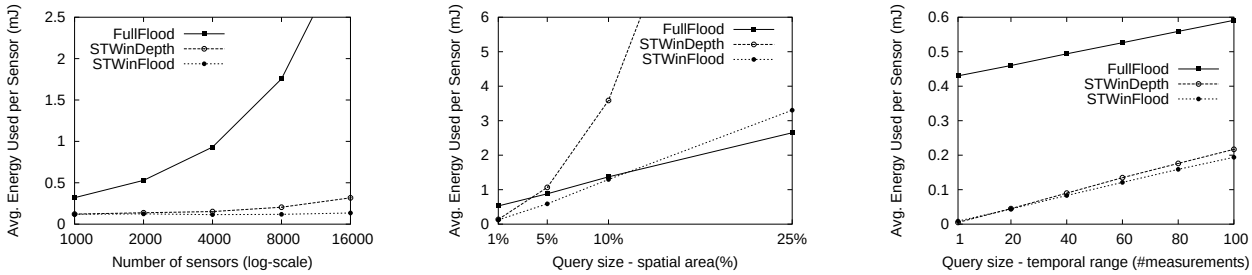
We implemented a sensor network simulator in order to study the performance of the presented algorithms. The sensors' placement follows a uniform distribution over a two dimensional region. We represent a historical spatio-temporal query *HSTQ* by the coordinates of a spatial area (*sw*), a temporal range (*tw*) and its query ID (*qID*). The query's spatial window covers 1% of the monitored region (that is 10% on each spatial coordinate), unless otherwise noted. The temporal window covers 60 measurements, where each measurement is represented by a <*value,time-stamp*> pair. A summary of query and sensor network parameters and their default values used in our experimental evaluation is presented in Table 1.

We compare the algorithms in terms of the average energy used per network node for communication while processing a query. According to [18], the energy used to transmit and receive one bit of information in wireless communication is given by:

$$\text{Energy}_{\text{transmit}} = \alpha + \gamma \times d^n \quad (1)$$

$$\text{Energy}_{\text{receive}} = \beta \quad (2)$$

where d is the distance to which a bit is being transmitted, n is the path loss index, α and β capture the energy dissipated by the communication electronics and γ represents the energy radiated by the power-amp. In our experiments, we use the following values for these parameters [3]: $\alpha = 45\ nJ/bit$, $\beta = 135\ nJ/bit$, $n = 2$, and $\gamma = 10\ pJ/bit/m^2$. As typical sensors do not have sophisticated communication electronics capable of adapting the transmission range [5], we assume all messages are transmitted as far as the wireless communication range. In our



(a) (b) (c)

Figure 4: The effect of several parameters on the average energy used per network sensor for the investigated algorithms

Parameter	Default Value
Area covered	1000x1000 meters
Wireless range	50 meters
Number of sensors	2000
Tuple size <*value, time-stamp*>	8 bytes
Query size	24 bytes
Query (spatial window)	1% (of area)
Query (temporal window)	60 measurements

Table 1: Parameters of query and sensor network

experiments we only measure the energy used to transmit and receive messages. We focus on the energy efficiency of the query processing algorithms and make the measurements independent of the characteristics of the MAC layer (for instance 802.11 radios consume as much energy in idle mode as for receive mode, while other radios may switch to a low-energy state when idle).

For the algorithms within the STWin framework, we call STWinDepth the combination of *GreedyDF* with *WinDepth*, and STWinFlood the combination of *GreedyDF* with *WinFlood*. All experimental measurements are averaged over 100 randomly generated sensor networks, with 10 random queries over each network.

First, we investigated the effect of node density on the performance of *GreedyDF*. For networks with 2000 or more nodes, *GreedyDF* is able to find a routing path from the query originator to a node inside the query's spatial window for most of the tested networks. In the majority of the successful cases, the reached node is located in the proximity of the center of query area. To have a fair comparison, the following measurements consider the energy used by an algorithm while processing a query only when each algorithm located all answers for that query.

Figure 4(a) presents the effect of the number of sensors on the energy usage of each algorithm. As node density increases, FullFlood sends a larger number of messages to nodes not relevant to the query, which leads to higher energy costs. The increase in sensor density leads to an increase in the number of nodes holding relevant data, which affects the costs of all algorithms, as a larger answer set must be returned to the query originator. With more nodes available for routing, the *GreedyDF* algorithm may be able to find a shorter path to the query coordinator node, an advantage for both STWinDepth and STWinFlood as less energy will be used for locating the query coordinator and a shorter path is used to return the answers from the coordinator node to the query originator. On the other hand, the coordinator node will send a larger answer set to the query originator in both STWinDepth and STWinFlood. The increase in the number of relevant nodes affects more STWinDepth than STWinFlood. This is due to the depth first policy used by *WinDepth* for query routing, as this policy contacts most relevant nodes on one query forwarding path. This behavior causes the larger answer set to be returned over a longer path to the query coordinator, which increases the energy usage.

The negative effects of this behavior of STWinDepth can be also seen in Figure 4(b), where the query size is increased. A larger query area affects the FullFlood algorithm less than the other two methods as only the communication cost for returning the answers increases for this algorithm, while the energy used for locating these answers stays constant. With the query's spatial window increasing, STWinFlood uses flooding over a larger set of nodes, ultimately degenerating into the FullFlood algorithm for large query windows. In both STWinDepth and STWinFlood, the answers from a larger spatial window are sent back to the query originator over a longer path (as the answers are first collected by the coordinator node) compared to the FullFlood method, which returns all answers over the shortest path. For large query windows, FullFlood uses less energy per node than STWinFlood for 2000 nodes, but STWinFlood performs better than FullFlood for large queries in denser networks (the corresponding graphs are not shown due to space limitations).

Figure 4(c) shows the effect of a query's temporal range on the energy consumption. A variation in the query's temporal range only affects the size of the answer messages, and leads to a linear variation of the energy used by these messages. The increase in energy usage is the smallest for FullFlood as the algorithm returns the answers over the shortest path to the query originator. STWinFlood performs better than STWinDepth because the relevant answers are returned on the shortest path to the query coordinator in STWinFlood, and both algorithms share the answer return path (discovered by *GreedyDF*) from the query coordinator node to the query originator.

6 Conclusions

While the technological advances have lead to sensors with reduced sizes and increased capabilities, the sensor data management is still in its incipient stages. The challenges are multiple, and the database research has to move its focus from considering time as a main optimization goal toward energy efficiency or a combination of both time and energy. The size of the database is no longer a primary

challenge, with the focus moving to the distributed nature of the database and query processing.

In this paper we made a few steps toward energy efficient query processing in a sensor network environment where each sensor is aware of only its neighbors. In this scenario, we proposed the STWin query processing framework, where the query is first forwarded to a query coordinator node within the query's spatial window, followed by an efficient query processing involving only the relevant nodes. Within this framework, we proposed the *GreedyDF* algorithm for the first phase, and *WinDepth* and *WinFlood* algorithms for the second phase.

The experimental results showed that STWinFlood is more energy efficient in most situations than simple flooding as well as the solution involving just depth-first based algorithms. Only for very large query windows in networks with low sensor densities, the FullFlood algorithm performs slightly better in terms of energy usage, and it is more robust for locating all relevant answers (however, it causes network congestion, reducing the network's capability to process several queries simultaneously). STWinFlood performs only slightly better than STWinDepth for small query windows, but the difference in performance dramatically increases for queries over large areas. An advantage of STWinDepth is that there are at most two nodes working in each query processing step, which allows the rest of the network to process other queries or simply sleep to save energy. For most cases, STWinFlood has shown low energy usage, and therefore we recommend it for sensor networks where each node is only aware of the other nodes located within its wireless range. The STWinFlood combines the strengths of both depth first and breadth first techniques while limiting their drawbacks.

In this paper we introduced techniques for query processing when the user in interested in retrieving all the relevant information. In other situations, an aggregated query answer may be sufficient. We are currently investigating new algorithms within the STWin framework that would allow efficient in-network aggregation during query processing.

Acknowledgments. This work was partially supported by NSERC. We would like to thank Ioanis Nikolaidis for his valuable comments and fruitful discussions, and the anonymous reviewers for their very useful suggestions to improve our work.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):392–422, 2002.

[2] J. Beaver, M.A. Sharaf, A. Labrinidis, and P.K. Chrysanthis. Power-aware in-network query processing for sensor data. In *Proc. of Hellenic Data Management Symposium*, pages 1–17, 2003.

[3] M. Bhardwaj. Power-aware systems. Master's thesis, MIT, 2001. http://www-mtl.mit.edu/research/icsystems/uamps/pubs/theses/.

[4] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Proc. of IEEE Conference on Mobile Data Management*, pages 3–14, 2001.

[5] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. Energy-efficient data management for sensor networks: A work-in-progress report. In *Proc. of Upstate New York Workshop on Sensor Networks*, 2003.

[6] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of International Conference on Peer-to-Peer Computing*, pages 32–39, 2003.

[7] Q. Fang, F. Zhao, and L. Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo Alto Research Center, 2002.

[8] D. Ganesan, D. Extrin, and J. Heidemann. DIMENSIONS: Why do we need a new data handling architecture for sensor networks. In *Proc. of Workshop on Hot Topics in Networks*, 2002.

[9] D. Ganesan, B. Greenstein, D. Perelyubskiy, D. Extrin, and J. Heidemann. An evaluation of multi-resoultion storage for sensor networks. In *Proc. of International Conference on Embedded Networked Sensor Systems*, 2003.

[10] S. Giordano, I Stojmenovic, and L. Blazevic. Position based routing algorithms for ad hoc networks: A taxonomy. In X. Cheng, X. Huang, and D.Z. Du, editors, *Ad Hoc Wireless Networking*. Kluwer, 2003.

[11] M.A. Hammad, W.G. Aref, and A.K. Elmagarmid. Stream window join: Tracking moving objects in sensor-network databases. In *Proc. of International Conference on Scientific and Statistical Database Management*, pages 75–84, 2003.

[12] C. Intanagonwiwat, R. Govindan, D. Extrin, and J. Heidemann. Directed diffusion for wireless sensor networking. *IEEE Transactions on Networking*, 11(1):2–16, 2003.

[13] S. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *Proc. of International Conference on Data Engineering*, pages 555–566, 2002.

[14] S. Madden, M.J. Franklin, and J.M. Hellerstein. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. of Symposium on Operating Systems Design and Implementation*, pages 131–146, 2002.

[15] S. Madden, M.J. Franklin, and J.M. Hellerstein. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, pages 491–502, 2003.

[16] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, 2001.

[17] V. Raghunathan et al. Energy aware wireless microsensor networks. *Signal Processing Magazine*, 45(2):40–50, 2002.

[18] T. Rappaport. *Wireless Communications: Principles and Practice*. Prentice-Hall Inc., 1996.

[19] S. Shakkottai, R. Srikant, and N. Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In *Proc. of INFOCOM*, 2003.

[20] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communications Magazine*, 40(7):128–134, 2002.

[21] Y. Xu and W.C. Lee. Window query processing in highly dynamic sensor networks: Issues and solutions. In *Proc. of Workshop on GeoSensor Networks*, 2003.

[22] Y. Yao and J. Gehrke. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31(3):9–18, 2002.

[23] Y. Yao and J. Gehrke. Query processing in sensor networks. In *Proc. of Conference in Innovative Data Systems Research*, 2003.

[24] D. Zhang, D. Gunopulos, V.J. Tsotras, and B. Seeger. Temporal and spatio-temporal aggregations over data streams using multiple time granularities. *Information Systems*, 28:61–84, 2003.

Mission-Critical Management of Mobile Sensors (or, How to Guide a Flock of Sensors)

Goce Trajcevski, Peter Scheuermann*
Department of ECE
Northwestern University
Evanston, Il
{goce,peters}@ece.norethwestern.edu

Herve Brönnimann
Department of CIS
Polytechnic University
Brooklyn, NY
hbr@poly.edu

Abstract

This work addresses the problem of optimizing the deployment of sensors in order to ensure the quality of the readings of the value of interest in a given (critical) geographic region. As usual, we assume that each sensor is capable of reading a particular physical phenomenon (e.g., concentration of toxic materials in the air) and transmitting it to a server or a peer. However, the key assumptions considered in this work are: 1. each sensor is capable of moving (where the motion may be remotely controlled); and 2. the spatial range for which the individual sensor's reading is guaranteed to be of a desired quality is limited. In scenarios like disaster management and homeland security, in case some of the sensors dispersed in a larger geographic area report a value higher than a certain threshold, one may want to ensure a quality of the readings for the affected region. This, in turn, implies that one may want to ensure that there are enough sensors there and, consequently, guide a subset of the rest of the sensors towards the affected region. In this paper we explore variants of the problem of optimizing the guidance of the mobile sensors towards the affected geographic region and we present algorithms for their solutions.

1 Introduction and Motivation

The management of the transient *(location,time)* information for a large amount of mobile users has recently spurred a lot of scientific research. It began with the investigation of the trade-offs in updating the information vs. minimizing the look-up time of a particular user's location (see [21] for a survey) and ranges to many aspects of modeling, efficient storage and retrieval, and processing of a novel types of spatio-temporal queries in the field commonly known as Moving Objects Databases[1] (MOD).

On the other hand, a challenging research field which recently emerged is the management of a *sensor-generated* data. Sensors are low-cost devices which are capable of measuring a value of a particular physical phenomenon and, eventually, transmitting it within a limited range. They may also have some limited processing power and can be mobile and deployed in a certain geographical area. Networks of sensors have already been deployed in the real world [13] and very active research efforts are being undertaken both in industry and academia [14]. Various aspects of interest for managing the sensor-generated data have been investigated (e.g., battery-life management, communication management of ad-hoc networks, stream-like management of the sensor-generated data, etc.) and a recent collection reporting the status of different research works is presented in [17] and [18].

Although mobility in sensor networks has been addressed in the context of communication protocols for ad-hoc and peer-to-peer networks (e.g., [16, 22]), we believe that the mobility dimension plays an additional important and unexplored role in the overall topic of the sensor data management, which is the basic motivation for our research. This particular work is based on the fact that the spatial range for which the quality of the readings that a sensor can guarantee is limited and we tackle the problem of how to deploy a *sufficient* number of sensors in a given region. The motivational scenario is the one of disaster management in a homeland security setting and can be described as follows. Assume that a set $\mathcal{S}$ of *mobile* sensors is de-

*Research partially supported by NSF grant IIS-0325144

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004), Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

[1] An recent collection of results is presented in [15]

ployed in a large geographic area in order to monitor particular value(s) of interest, e.g., the temperature and the concentration of toxic materials in the air. In case a certain subset $\mathcal{S}_k$ of the sensors, co-located in a given region, report readings which exceed a given (pre-defined) tolerance threshold, we would like to ensure the quality of readings of the sensors' data for the *critical region*. In order to do so, we would like to ensure that there are enough many sensors inside that particular critical region and our goal is to optimize the guidance of a subset of m sensors from $\mathcal{S} \setminus \mathcal{S}_k$ towards the interior of the critical geographic region.

Throughout this work, we investigate few variations of the problem of optimizing the guidance of the set of sensors towards the critical region, in order of their increasing difficulty. In some way, our work can be viewed as a step towards adding spatio-temporal context awareness in managing sensor data.

The rest of this paper is organized as follows. Section 2 formally introduces the terminology used. In Section 3 we introduce the concept of *critical times* with respect to the guidance of mobile sensors and we present three variants of the problem of reachability with respect to the critical region. These variants are used as a basis for the problems addressed in Section 4, where we have more realistic requirements of the placement of the sensors within some optimal timeframe and we also consider the spatial limit on the validity of the data read by a particular sensor. Section 5 presents yet another variation of the problem of optimizing the guidance of the set of mobile sensors with respect to a critical geographical region which, in a sense, is the "opposite" of the problems presented in Sections 3 and 4. Section 6 gives a brief overview of the relevant literature and in Section 7 we present concluding remarks and we outline some areas for the future research work.

2 Preliminaries

In this section we formally introduce the terminology used in the rest of the paper.

We assume that we are given a set of distributed mobile sensors $\mathcal{S} = \{s_1, s_2, \ldots s_k\}$, where each s_i is represented as an ordered pair $s_i = ((x_i, y_i), v_i)$. (x_i, y_i) denotes the location of the sensor s_i and v_i denotes its speed. Typically, the sensors motion plan for the future (future trajectory) or the past completed motion (past trajectory) can be represented as a polyline in 3D space: (x_{i1}, y_{i1}, t_{i1}), ..., (x_{in}, y_{in}, t_{in}) [24], however, without loss of generality, we omit this modeling aspect from the paper.

Each sensor s_i periodically reports the reading val_i of the value of the physical phenomenon that it is observing. Let C_v to denote a value which is a tolerance-threshold for the monitored physical value.

Definition 2.1 *A sensor s_j is called* hot *if it reads a value $val_j > C_v$.*

Assuming that, at a certain time instance, a subset of the set of the sensors have read values greater then C_v, we have the following definition:

Definition 2.2 *Given a subset $\mathcal{S}_k \subseteq \mathcal{S}$ of sensors $\mathcal{S}_k = \{s_{j1}, s_{j2}, \ldots, s_{jk}\}$ such that $(\forall i)(val_{ji} \geq C_v)$, the* critical region *$C_R$ of $\mathcal{S}_k$ (also denoted as $C_R(\mathcal{S}_k)$) is defined as the convex hull of the set of 2D points $\{(x_{j1}, y_{j1}), (x_{j2}, y_{j2}), \ldots, (x_{jk}, y_{jk})\}$.*

Defining the critical region as a convex hull of the locations of the hot sensors is justified by several "natural" properties (c.f. [20]):

- The convex hull is the convex polygon with the smallest area which encloses a set of (planar) points.
- The convex hull is the convex polygon with the smallest perimeter. which encloses a set of (planar) points.
- The concept of the convex hull and the algorithms for its computation are very well studied topics in the field of Computational Geometry.

The concepts introduced in Definition 2.1 and 2.2 are illustrated in Figure 1. The hot sensors are indicated by the dark disks and the white disks indicate the sensors which not hot. As mentioned in Section 1, for the purpose of ensuring certain *quality* of the readings of the sensor-data, we would like to ensure that there are "enough many" (application specific) sensors in the critical region and we are focusing on minimizing the time that it takes to deploy them. In the rest of the paper, we present the algorithms which handle different variants (due to different constraints/requirements) of the problem of optimal deployment of mobile sensors in a given critical region.

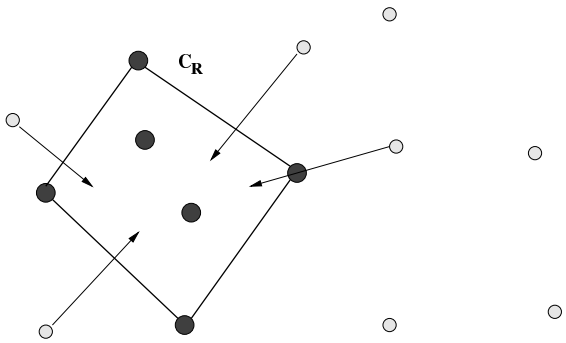


Figure 1: Guiding the sensors towards the critical area

3 Critical Times

Now we proceed with a few variations of the first category of problems of optimizing the deployment of sensors in a given critical region. In this section we address issues related to arrival of mobile sensors in the interior of the critical region which, as it turns out, are important for the settings of the problem(s) that we consider in Section 4.

3.1 Minimizing the Arrival Time

The simplest variation of the problem is the one which assumes that in order to ensure the desired quality of the sensor data (i.e., the desired coverage of the critical region) it suffices to have m sensors in the interior (or, on the boundary) of C_R. For this case, we would like to ensure that the *time* it takes for a desired number of sensors to *arrive* inside C_R is *minimized*. Figure 1 illustrates a scenario where we have six hot sensors and, in order to ensure the desired coverage of C_R we need a total of ten sensors. Thus, we need to bring four more sensors inside C_R, in a quickest possible manner. The problem can be formally stated as follows:

Problem 3.1 Minimal Arrival Times (MAT):
Given: *An integer m; a set of sensors $\mathcal{S}$; and a subset $\mathcal{S}_k \subseteq \mathcal{S}$;*
Goal: *Minimize the time for which it can be guaranteed that there are m sensors in $C_R(\mathcal{S}_k)$.*

Let $k =\mid \mathcal{S}_k \mid$ and observe that if $m \leq k$ we have already satisfied the quality requirements. Let t_{ai} denote the minimal time-value for which the sensor $s_i \in \mathcal{S}$ can reach the boundary of C_R, which we will call its *critical arrival time – cat_i*. Obviously, if $(x_i, y_i) \in C_R$, then $cat_i = 0$. The minimal time for a particular sensor $s_i \in \mathcal{S}$, which is outside C_R, to reach the boundaries of C_R, is actually equivalent to the time it takes for a circle centered at (x_i, y_i) and with radius $v_i \cdot t_i$ to intersect C_R.

Clearly, after constructing the convex hull for the location-points of the hot sensors (the ones in $\mathcal{S}_k$), one only needs to determine the set of m closest points to $C_R(\mathcal{S}_k)$ and get their arrival times $cat_{a1} \leq cat_{a2} \leq \ldots \leq cat_{am}$. In order to achieve our goal, we need at least $cat = cat_{am}$ time units. The asymptotic complexity of this approach is bounded by $O(n\log k)$, since the determination of the minimal distance from a point to a given convex region with k edges[2] can be achieved in $O(\log k)$ [20]. Observe that *cat* is the lower bound on the time that we need to ensure that there are m sensors *anywhere* inside $C_R(\mathcal{S}_k)$.

Let us point out that in case the critical region $C_R(\mathcal{S}_k)$ is defined as a circle, the diameter of which is the diameter of the set of location-points of the sensor in $\mathcal{S}_k$, the complexity of calculating *cat* reduces to a linear ($O(n)$) time.

[2]The convex hull of the set of k points is a polygon which may have up to k edges/vertices [2, 20].

3.2 Minimizing the Furthest-Point Reachability Time

The next variation of the problem of ensuring the quality of the sensors' readings in the critical region considers the upper bound on the time it takes to bring the desired number of sensors (m) inside the critical region. Once again, we have a problem of selecting a subset of $(\mathcal{S} \setminus \mathcal{S}_k)$ of size m, except now the selection criterion for the purpose of ensuring the quality of the readings is different. Formally:

Problem 3.2 Minimal Furthest-Point Reachability (MFR):
Given: *An integer m; a set of sensors $\mathcal{S}$; and a subset $\mathcal{S}_k \subseteq \mathcal{S}$;*
Goal: *Minimize the time for which it can be guaranteed that each of the m sensors has reached the furthest point (with respect to its current location) in the interior of $C_R(\mathcal{S}_k)$.*

Let t_{fi} denote the minimum time-value for which the sensor $s_i \in \mathcal{S}$ can reach the furthest point in C_R with respect to its location (x_i, y_i). We will call it its *critical furthest-point time – cft_i*. In a manner similar to the calculation of the *cat* time, in $O(n \log k)$ we can obtain the set of m sensors such that $t_{f1} \leq t_{f2} \leq \ldots \leq t_{fm}$. If we set $cft = t_{fm}$ then cft is, in a sense, an upper bound on the time it takes to ensure that there are m sensors *anywhere* inside $C_R(\mathcal{S}_k)$.

Now we proceed with the more desirable (and more complicated) setting of optimizing the critical time.

3.3 Critical Covering Time (cct)

The formal statement of this problem is specified as follows:

Problem 3.3 Minimal Interior Reachability (MIR):
Given: *An integer m; a set of sensors $\mathcal{S}$; and a subset $\mathcal{S}_k \subseteq \mathcal{S}$;*
Goal: *Minimize the time cct for which it can be guaranteed that there exists a subset $\mathcal{S}_m \subseteq \mathcal{S}$ of m sensors that can be brought in the interior of $C_R(\mathcal{S}_k)$ in such a manner that any point inside $C_R(\mathcal{S}_k)$ can be reached by some sensor in $\mathcal{S}_m$ in time $\leq cct$.*

Obviously, the goal of the **MIR** problem is to minimize the time-value for a given m – the number of sensors which ensures the quality of the readings in a given critical region. However, one may very naturally be interested in the dual optimization problem, which can be formulated as:

Problem 3.4 Minimal Number of Sensors (MNS):
Given: *A time-value cct; a set of sensors $\mathcal{S}$; and a subset $\mathcal{S}_k \subseteq \mathcal{S}$;*
Goal: *Minimize m, such that a subset $\mathcal{S}_m \subseteq \mathcal{S}$ with m elements exists, for which any point within $C_R(\mathcal{S}_k)$ can be reached by some sensor in $\mathcal{S}_m$ in time $\leq cct$.*

However, this is an instance of the set-cover problem, which is NP-complete [10]. Even this particular instance (disk-covering in 2D) is NP-complete, although it can be approximated within a constant factor [5], as opposed to logarithmic at best for the general set cover. Thus, the best solution one can hope for is a heuristic solution. One possible approach is to relax the limit of m and ask how *all* the sensors can achieve the desired covering of C_R (i.e., set $m = n$). In this case, the decision problem **MIR** amounts to constructing the union of all the n disks and checking if it covers C_R. This can be done in $O(n \log^2 n)$ time, as the union of disks (even of different radii) has complexity $O(n)$ [3, Ex 3.6]. Let us point out that by applying binary searching, one can determine the $\mathcal{S}_m$ and *cct* up to any accuracy (using **MIR**). The exact value can be determined in $O(n\text{polylog}n)$ time by designing a parallel version of the decision algorithm and using parametric searching.

4 Spatial Limits on the Validity of Readings and Sensors Placement

Based on the results presented in Section 3, in this section we present a more stringent set of requirements, which are more realistic for practical purposes. The key assumption is that the readings of each sensor are *valid* only within a *limited* area, which is represented as a *disk* with radius r centered at a sensor's location-point. Building up on the results in the previous section, first we will discuss a variation in which we assume that there are enough sensors to cover the critical region C_R and, subsequently, we address the more realistic setting of limited number of available sensors.

4.1 Full Coverage of C_R

Instead of having m sensors inside C_R, our goal now is to minimize the time for which C_R can be entirely covered by disks of radius r centered at the sensors location-points. We assume that we have sufficient number of disks to ensure the coverage of C_R. The problem can now be stated as follows:

Problem 4.1 Minimal Full Coverage Time (MFC):
Given: *A set of sensors $\mathcal{S}$ and a region C_R;*
Goal: *Determine the minimal time (denote it mrt – minimal routing time), such that a subset $\mathcal{S}_m \subseteq \mathcal{S}$ exists which can be moved inside C_R in such a manner that every point in C_R is at distance $\leq r$ from the location-point of some sensor $s_{mi} \in \mathcal{S}_m$.*

Observe that the problem has some implicit requirements – we need to determine the *trajectory* of each mobile sensor s_{mi} and (recall that) we do not even have the limit for m set in advance. If we let $A(C_R)$ denote the area of the critical region and ε denote the maximal percentage of overlap between two disks that a user allows[3], then we can have a reasonable *lower bound* on m calculated as $A(C_R)/(\pi \cdot (1-\varepsilon))$. Clearly, the more sensors we have available, the smaller value of *mrt* we can obtain.

The formulation of the corresponding dual-like problem can be specified as:

Problem 4.2 Time-Limited Full Coverage (TFC): Given: *A set of sensors $\mathcal{S}$; a region C_R; time-value (limit) mrt;*
Goal: *Determine the minimal m such that m sensors can be placed inside C_R in such a manner that C_R can be covered by disks of radius r centered at the sensors location-points.*

Obviously, the techniques presented in Section 3 cannot be directly applied in these settings. However, they can still give us some useful bounds. Let (x_i, y_i) denote the "current" location of the i-th sensor (which is, before it is routed towards C_R). Then *mrt* must be large enough such that the union of the disks centered at each (x_i, y_i), with respective radii $r + v_i \cdot mrt$, covers the entire region C_R. This is equivalent to the requirement that *any* point of C_R (interior + boundary) can be reached by *at least* one sensor. Thus, a reasonable lower bound for *mrt* is the critical coverage time – *cct* (c.f. Section 3).

This is illustrated in Figure 2, where for simplicity we have assumed that the only hot sensors are on the vertices of C_R (we do not indicate their coverage area). White disks indicate the initial location and the area in which the readings of a particular sensor are valid and dashed circles indicate the boundaries that a particular sensor can reach for a valid reading within time t.

We propose two heuristic solutions. The first one, which is trying to cater the worst case, can be explained as follows. At the time $t = cct$, at which *all* points of C_R can be reached by some sensor, pick a point that was reached (covered) last – this point takes *longest* time, but will *have to* be covered. Remove the corresponding covering disk(s) and repeat the procedure to the leftover of C_R. This is illustrated by the

[3] Observe that some overlap will be inevitable, e.g., even if we are to cover a unit disk D with disks of radius $\rho < 1$, we have that the limit (as $\rho \to 0$) of the ratio of the area of the disk D and the sum of the areas of all the covering disks is $\frac{3 \cdot \sqrt{3}}{2\pi}$ (c.f. [9]).

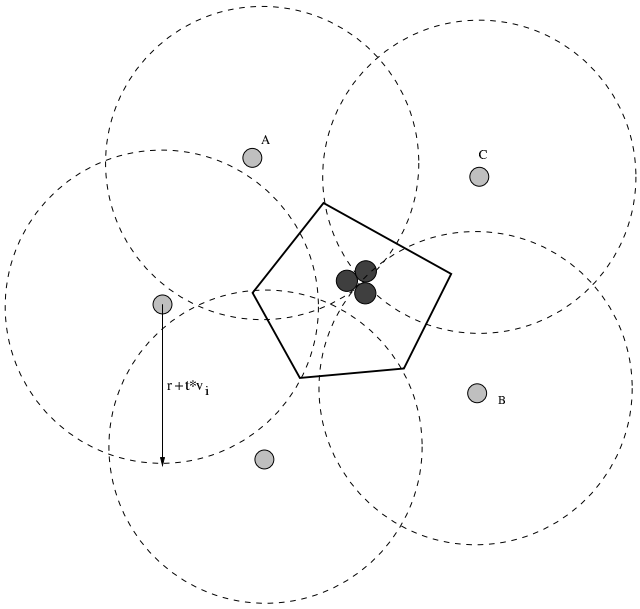


Figure 2: Spatial Coverage of the Critical Region

disks A, B and C in Figure 2, where the dark disks indicate their final positions and are to be removed from the cover. If the sensors are well distributed, we expect that we can cover all the other points within time *cct*. This may not be the case however, since removing the corresponding disk increases the *cct* of the remaining C_R. This can happen if some point previously reached before *cct* by the removed sensor now needs to be reachable by another sensor. In such cases, if there is not close-enough sensor, the new *cct* will increase. This heuristic is also not guaranteed to give a optimal number of covering sensors, since it starts from the center and works the covering towards the boundary of C_R. In fact, we expect it to yield twice the optimal number of sensors. Note, however, that it will not be worse than four times the optimal number of sensors, since the centers of the disks are chosen outside the union of the already chosen final disk positions. A standard packing/covering argument implies that the halved circles with the same center are disjoint, and an area-based argument justifies the claim.

Our second heuristic tries to address precisely the problem of minimizing m. Essentially, the above solution is suboptimal because it only looks at sensors locally, and one by one. Trying to look at the entire picture, we can decide a priori the final location of the sensors by computing a minimal covering in the shape, say, of a honeycomb, or using an incremental algorithm to add the disks one by one. Note that standard arguments, similar to the one above, can be used to imply that the number of disks in such a packing is within a small constant ϕ from the optimal number. Next, we compute a Euclidean minimum matching [19] between the sensors and the final positions, which tells us which sensors go where. As a last step, in order to "refine" the solution, we may even want to apply a local perturbation scheme for the purpose of optimizing the critical covering time *(cct)*, while retaining the covering property[4]. All of the above can be carried out in $O(mn)$ time. As a last observation, let us point out that using more sensors than the number m found by the cover, we can expect to lower the value of *mrt*.

4.2 "Fair" Coverage of C_R with Limited Number of Sensors

The last variant of the problem of covering the critical region corresponds to the realistic settings of having limited resources available. The initial assumption for this section was that there are *enough* sensors available for the coverage of C_R. In other words, depending on the value of the valid coverage area r of an individual sensor's readings, we assumed that the value of m is large enough so that C_R is fully covered with m (partially overlapping) disks. However, similar to the scenarios considered in Section 3, we now assume that we have a limit on m – the number of sensors that can be deployed inside C_R, each with some valid area of its readings. In this case, the question becomes *how* to select the *locations* for each of the m sensors inside C_R so that we can guarantee that, whenever needed, any point within C_R can be reached by one of the m sensors within "reasonable time". Again, we will have a subsequent step to handle, which is, which of the n sensors in $\mathcal{S}$ should be the ones to be placed in the chosen m locations inside C_R. More formally, now we have to solve:

Problem 4.3 Fair Coverage Problem (FC):
Given: *A critical region C_R and an integer m;*
Goal: *determine the locations of a set of m points $\mathcal{P} = \{ p_1, p_2, ..., p_m\}$ in the interior of C_R such that the* time *for which every point on inside or on the boundary of C_R can be reached by a sensor located at some p_i is minimized;*
Plus, its "next stage" of selecting which *m sensors should be guided in each p_i so that the mrt is minimized too (Euclidean minimum matching [19] again).*

To handle **FC** we obtain a fair distribution by finding a value r' ($r' \geq r$) such that C_R can be covered with at most m disks of radius r'. Once we have determined the locations of the centers of the m disks, placing a sensor in each center ensures that even C_R is not entirely covered, any point not covered can be covered by moving one of the sensors by the smallest amount possible (i.e., in minimal time). Again, we need $O(mn)$ time to carry out the solution.

[4]Observe that in this solution the number m of sensors needed to cover is essentially dictated by the value of r – the radius of validity of sensors' readings.

5 "Potpourri": Escape From the Critical Region

Now we briefly turn our attention to the "inverse-image" of the problems considered in the previous two sections. Namely, instead of optimizing the deployment of a sufficient number of sensors inside the critical region, we analyze the case when one would actually want to ensure that the sensors from the interior of the C_R (both hot and non-hot ones) are guided outside C_R as soon as possible. Such setting is of interest for scenarios like, for example, when the values read by the hot sensors could indicate that there may be a fire in a certain geographic area.

Again, we assume that the readings of each sensor are valid within a disk-like area with radius r, centered at the sensor's location-point. In order to ensure some quality of the monitoring of the values along the critical region's boundary, we would like to guide each individual sensor at distance r from the boundary of C_R. Formally, the problem that we address in this section can be stated as follows:

Problem 5.1 Minimal Escape Time (ET):
Given: *A critical region C_r and a subset of sensors $\mathcal{S}_{CR} \subseteq \mathcal{S}$ such that each $s_a \in \mathcal{S}_{CR}$ has its location-point inside (or, on the boundary of) C_R.*
Goal: *Minimize the time that it takes to move all the sensors from $\mathcal{S}_{CR}$ at distance r from C_R.*

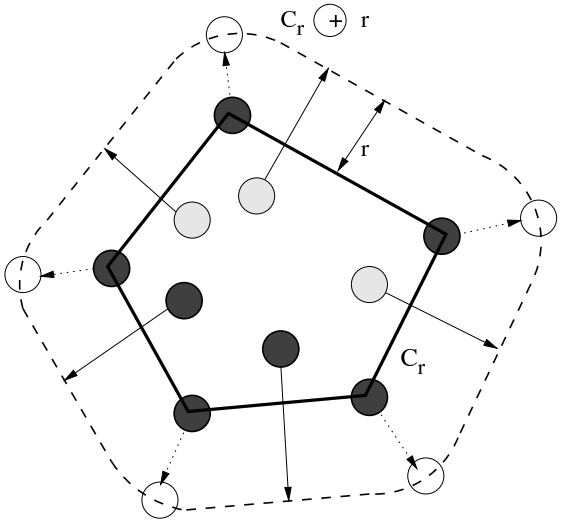


Figure 3: Escaping from the critical region

The illustration is provided in Figure 3. Solid disks indicate the initial locations of the (hot and non-hot) sensors inside C_R. Since the quality of the readings of each sensor is guaranteed within a disk of radius r, for safety, we want to guide the sensors at distance r from C_R's boundary. For that, we first need to determine, what is commonly called, the *Minkowski Sum* of the region C_R with a disk with radius r. We will use $C_R \oplus r$ to denote the operation of Minkowski Sum and informally[5] it can be described as the region obtained when the disk with radius r is "swept" along the boundary of C_r. The empty disks in Figure 3 indicate the final locations of the sensors which were initially the vertices of C_R.

Since the construction of the convex hull (C_R) is assumed to yield a polygon with $O(k)$ edges/vertices (in time linear in k), the construction of the Minkowski Sum of a convex polygon (C_R) with a disk with radius r can be done in $O(k)$ (c.f. [2]). Similarly to the discussion in Section 3 (**MAT** problem), for each sensor's location-point in C_R, we can find the closest point on the boundary of $C_R \oplus r$ in $O(\log k)$. Assuming that (worst case) initially all the sensors from $\mathcal{S}$ were in C_R, the time-complexity of the algorithm for solving the **ET** problem is $O(n \log k)$.

In case certain quality of the sensor's readings needs to be ensured by placing a given number of sensors on the boundary of $C_R \oplus r$, we can apply some of the variations of the guidance problems that we considered in Section 3 and 4, respectively.

6 Related Literature

MOD researchers have addressed many aspects of interest for management of spatio-temporal data. Largest efforts were made in the area of indexing a collection of moving objects for a purpose of efficient query processing, however, MOD-related problems turned out to have many challenging aspects: modeling/representation based on different ontologies and algebraic types; linguistic aspects; novel query types and their processing algorithms [15]. In this work, we addressed a novel aspect of a "semantic-based" management of moving objects were the semantics of the problem was motivated by the settings of sensor data management.

Mobility aspects in a data-motivated settings have been addressed from perspective of ad-hoc and P2P networks. However, most of the works are targeted towards organizing structures which would ensure a dissemination of information (communication) and effectiveness of routing [22]. Some Computational Geometry techniques (dual space transformation) have been employed for efficient tracking of mobile sensors, which enable efficient communication and power management [16]. Our work is, in a sense, orthogonal to the existing results because we focused on the guidance of a set of mobile sensors for the purpose of quality assurance of the data read by those sensors in a given geographic region.

Two works which are close in spirit to ours are presented in [11] and [23]. In [11] the authors consider

[5]Formally, the Minkowski Sum of two sets of points P_1 and P_2 can be specified as $P_1 \oplus P_2 = \{p_1 + p_2 \mid p_1 \in P_1, p_2 \in P_2\}$, where the summation is of *vector* p_1 with *vector* p_2 (c.f. [2]).

some spatio-temporal correlation with the quality of the data read. They introduce the notion of *swarms*, which are nodes with higher processing capabilities than the regular sensor nodes and address the problem of efficient guidance of the swarms towards the location(s) of a *hot* static sensor(s). Our work is, in a sense, complementary to the one in [11] – we address the problem of ensuring that there are enough many sensors brought in a given critical region. On the other hand, [23] considers the problem of limited transmission range and arrangements of the nodes in ad-hoc network which will ensure probabilistic bound on connectedness. However, we consider the aspect of limited range of the sensor readings for the purpose of ensuring different quality criteria.

7 Concluding Remarks and Future Work

We have addressed the problem of the efficiency of ensuring some quality of data-readings by a set of mobile sensors in a given critical geographic region . We presented different variations of the problem and derived algorithms for their solution. Currently we are focusing on obtaining comparative experimental results for our heuristics.

The work that we presented here is part of a larger research effort that we are currently undertaking in the area of context-aware MOD. Our MOD database consists of information about mobile users (e.g.,their motion plan, preferences, etc.), information about static objects of interest, as well as the information collected by the various sensors. This database is maintained in a distributed fashion, with the current sensor data being kept by various sensor nodes and the historical sensor data being accumulated at some sensor servers which can be mobile themselves, in a similar spirit to the concept of *swarms* (c.f. [11]). Our system handles continuous queries and notifications which need to be re-evaluated when there are some changes in the motion plans of the users or in the environmental context. The sensor data falls into the environmental context dimension and this data is used in order to detect which objects in the users' database need to be notified of the changes in the environment or which objects' trajectories need to be modified accordingly. For example, some unusually high temperatures and low winds detected by the sensors are used to detect a fire. The system will then check if there are any outstanding requests for user notifications that need to be triggered. In this case, only the users who have requested to be notified of a fire within a certain geographic area are notified. We observe here that the individual readings of the sensor nodes need to be aggregated at the coordinating sensor servers, so that some intelligent reasoning can be performed there, such as the fact that a fire has been detected. Thus, the sensor data can be viewed as consisting of a number of data cubes, each having at minimum the time and location dimensions.

The "correlation" we considered in this paper was between the (critical) geographic region determined by the set of sensors which simply report a value past certain threshold and the number of sensors in that region. However, one may observe that we did not consider the issues of the limited communication range and the limited mobility (e.g., a road-map in an urban environment), which are parts of our ongoing work.

We envision a lot of interesting topics in the field of sensor data management which can benefit from the extensions of some of the existing works in the database research and can, in turn, pose challenges for database researchers:

• Uncertainty – The problem of imprecision of the values in the MOD with respect to the real-world values of the entities represented has been addressed both in the context of modeling and processing nearest-neighbor and range queries [7, 24]. The problem of imprecision of sensor data has also been tackled in [8]. What are the consequences when the uncertainties in both context dimensions *(location,time)* and *data* values are brought together? What are the queries that can be posed and how can they be processed?

• Data reduction – Although not explicitly, the problem of data reduction can be viewed as a "flip-side of the coin" of uncertainty management. Reducing the size of the data set with deterministic bounds on the query error has been addressed independently in the MOD settings [6] and the stream-like database settings where the number of passes over the data should be minimized and yet the sample retained should exhibit a bound on the query-errors [4]. What is the impact of the difference of the context dimensions (semantics of the data read vs. location and time of the sensor) on the algorithms which could reduce the total size of the data kept in a database?

• Computational Geometry Techniques – In this work we have already utilized some results from CG literature. Is there are room for more collaborative results between the database and the CG researchers in the context of sensor data management? We believe so – to a large extent. In particular, one of the immediate challenges of our results is the efficient management of mobile critical regions (e.g., the fire is spreading dynamically). The MOD researchers have already addressed the issue of algebraic modeling of moving polygons (c.f. [12]) and the CG researchers have already addressed the issue of incrementally computing the convex hull of a set of moving points with known motion plans (c.f. [1]). Another extension of our work is how to manage the *mobile swarms* (c.f. [11]) and *mobile sensors* in the context of quality of reading and processing of sensor-generated data, which can readily be categorized as a "mobile version"

of the clustering problem.

References

[1] P. Agarwal, L. Guibas, J. Hershberger, and E. Veach. Maintaining the extent of a moving point set. In *WADS*, 1997.

[2] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of minkowski sums. *Computational Geometry*, 21(1-2), 2002.

[3] J.D. Boissonat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, 1998.

[4] H. Brönnimann, B. Chen, M. Dash, and Peter Scheuermann. Efficient data reduction with ease. In *SIGKDD*, 2003.

[5] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete and Computational Geometry*, 14, 1995.

[6] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. In *DIAL-POMC*, 2003.

[7] R. Cheng, D.V. Khalashnikov, and S. Prabhakar. Querying imprecise data in moving objects environments. *IEEE-TKDE*, 16(7), 2004. (to appear).

[8] R. Cheng and S. Prabhakar. Managing uncertainty in sensor databases. *SIGMOD Record*, 32(4), 2003.

[9] S. R. Finch. *Circular Coverage Constants*. Cambridge University Press, 2003.

[10] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to Theory of NP-Completeness*. W.H.Freeman, 1979.

[11] M. Gerla and K. Xu. Multimedia streaming in large-scale sensor networks with mobile swarms. *SIGMOD Record*, 32(4), 2003.

[12] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. Lorentzos, E. Nardeli, M. Schneider, and J. R. R. Viqueira. Spatio-temporal models and languages: An approach based on data types. In *Spatio-Temporal Databases: the Chorochronos Approach*. 2003.

[13] J. Hellerstein, W. Hong, and S. Madden. The sensore spectrum: Technology, trends and requirements. *SIGMOD Record*, 32(4), 2003.

[14] J. Hill, R. Sczeczyk, A. Woo, S. Hollar, and D.C.K. Pister. System architecture directions for networked sensors. In *ASPLOS*, 2000.

[15] M. Koubarakis and T. Sellis et al., editors. *Spatio-Temporal Databases: The CHOROCHRONOS Approach*. Springer (LNCS 2520), 2003.

[16] J. Liu, P. Cheung, L. Guibas, and F. Zhao. A dual-space approach to tracking and sensor management in wireless sensor networks. In *WSNA*, 2002.

[17] L. Liu, editor. *SIGMOD Record*, volume 32. ACM Press, 2003.

[18] L. Liu, editor. *SIGMOD Record*, volume 33. ACM Press, 2004.

[19] J. O'Rourke. http://cs.smith.edu/ orourke/TOPP/P6.html.

[20] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 2000.

[21] E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(4), 2001.

[22] V. Ramasubramanian, Z.J. Haas, and E.G. Sirer. Sharp: A hybrid adaptive routing protocol for mobile ad hoc networks. In *MobiHOC*, 2003.

[23] P. Santi, D.M. Blough, and F. Vainstein. A probabilistic analysis for the range assignment problem in ad hoc networks. In *MobiHOC*, 2001.

[24] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS)*, 2004. (to appear in 29(3)).

KPT: A Dynamic KNN Query Processing Algorithm for Location-aware Sensor Networks

Julian Winter Wang-Chien Lee

Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
Email: {jwinter, wlee}@cse.psu.edu

Abstract

An important type of spatial queries for sensor networks are K Nearest Neighbor (KNN) queries. Currently, research proposals for KNN query processing is based on index structures, which are typically expensive in terms of energy consumption. In addition, they are vulnerable to node failure and are difficult to maintain in dynamic sensor networks. In this paper, we propose KPT, an algorithm for dynamically processing KNN queries in location-aware sensor networks. KPT shows great potential for energy savings and improved query latency. Since the tree infrastructure is constructed only temporarily, KPT is less vulnerable to sensor node failure.

1 Introduction

Recent research on accessing data available in sensor networks has been focused on index structures, data storage, routing algorithms, data dissemination and aggregation techniques [2, 4, 6, 8, 7, 9, 15]. A major goal of these proposals is to support various types of queries posed to a sensor network from any location. A query is transmitted from the query source to the sensor nodes or network locations that contain the data needed to satisfy the query. The results (i.e., data collected at the sensor nodes) are then aggregated (if allowed) and returned back to the query source. The main requirement for query processing is to incur as little energy expenditure as possible without dropping the queries or sacrificing execution latency.

Spatial queries such as window/range queries and k nearest neighbors (KNN) search are particularly relevant to sensor network applications because the data needed for these applications is often geographically distributed in the network. Several approaches have been proposed that support window/range queries in sensor networks [5, 14], while a preliminary study of the KNN queries in sensor networks, called *Peer-Tree*, has just started [1]. Peer-tree, a distributed index structure based on the design principle of R-trees, ignores the fact that sensor nodes are susceptible to radio interference, signal attenuation, and fading. As a result of these radio problems index structures are difficult to implement in sensor networks and expensive to maintain in terms of energy consumption. This paper introduces the KNN Perimeter Tree (KPT) Algorithm for supporting KNN queries. KPT exploits the fact that KNN queries are geographically-based to achieve energy savings and increased fault tolerance. A preliminary performance evaluation is given to demonstrate the capabilities of KPT. For this paper we are assuming a stationary, location-aware sensor network. KPT assumes that sensors are aware of their geographical neighbors needed to support geographical routing. Sensor data is stored using local storage which can be organized as cache lines based on sensing event types. A given sensor can aggregate data over a period of time; for example a line in the cache may represent the sensing data of a minute.

This paper is organized as follows. In Section 2 we introduce KNN queries in sensor networks and review relevant research efforts. Then we introduce the KPT algorithm in Section 3 and its analysis in Section 4. Finally, Section 5 concludes this study and discusses the future work.

2 Related Work

In this section we describe the background of sensor networks, KNN queries and related research contributions.

2.1 KNN Query in Sensor Networks

k-Nearest Neighbor (KNN) queries of spatial data have been an interesting research topic for some time [10, 11]. A KNN query is initiated by a query source node and involves finding the k spatially nearest objects to a given query point

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
http://db.cs.pitt.edu/dmsn04/

within the sensor network. Centralized or distributed index structures such as the R-tree have provided support for KNN queries [3]. However, in the context of sensor networks, technical issues such as node failures (caused by depleted energy resources or communication problems) make such index structures unwieldy and inefficient for executing KNN queries.

KNN queries can be classified into two types for sensor networks. For Type 1 queries, we assume that all sensor nodes locally store sensor data and are able to answer a specific query constrained by a geographical query condition. For example, assume that a query desires the k nearest temperature readings to some query point and all sensor nodes have a sensing component to measure temperature. In this case, the query needs to be transmitted to the k geographically nearest sensor nodes to the desired query point. The KNN nodes sample the temperature data and return it back to the query source node.

For Type 2 KNN queries, we assume that some additional query condition precludes the ability of all sensors to satisfy a query despite being located inside the desired geographic region. Type 2 queries request sensor data about the k nearest events to some given query point. These event locations are unpredictable and therefore determining which k sensors to transmit the query to for execution is more complicated than Type 1. In this paper, we consider only Type 1 KNN queries and leave support of Type 2 KNN queries as future work.

2.2 Geographical Routing

We assume for this research that sensor networks are stationary and location aware and that sensor nodes are knowledgeable about neighbor nodes within their radio range. Given these assumptions, several algorithms exist that can route messages towards geographic locations.

The Greedy Perimeter Stateless Routing (GPSR) algorithm is a geographical routing algorithm which operates in two modes in location-aware sensor networks: *greedy* mode and *perimeter* mode [4]. In greedy mode, the forwarding node forwards the message to the neighbor nearest the destination. If no such neighbor exists, the algorithm switches to perimeter mode, which, given a planarized graph of the network topology, routes messages around voids in the network. GPSR can be employed for routing Nearest Neighbor (NN) queries in sensor networks. Given a desired location, GPSR can continue to route the query message until the NN to the query point is reached. The nearest neighbor sensor node can be confirmed by routing in perimeter mode around the query point. Due to this nice property, GPSR was selected as the routing protocol for implementing KPT.

2.3 Peer-Tree

To the best of our knowledge, Peer-Tree (PT) is the only other proposal in the literature that is able to support KNN queries. Peer-tree applies the decentralized R-tree index structure to ad-hoc sensor networks in order to support location-based queries [1].

Like with the R-tree, the sensor network is partitioned into Minimum Bounding Rectangles (MBRs). Each MBR covers a geographical region and includes as a member any sensor node inside that area. The clusters are then organized in a hierarchical fashion until one overall cluster geographically spans the entire network. For each cluster, a specific node is designated as a clusterhead, which knows the location and ID of all sensors that belong to the MBR cluster. Furthermore, it knows the location and ID of the clusterheads of any child clusters and its parent clusterhead. Although the authors do not discuss the physical layer of the network topology directly, it is logical that that the authors assume the clusterhead can communicate with all nodes within its MBR as well as its parent.

In Peer-Tree, queries do not originate at the root of the tree, but come up from the level 0 child nodes since it is desirable to allow queries to be spawned from random locations in the network. NN queries can be locally scoped to include only the largest MBR necessary for satisfying the query. For handling NN queries, the source node routes the query message to its clusterhead. The clusterhead determines whether the query point is within its MBR. If so, the clusterhead then begins the algorithm for finding the NN. If it is not, the clusterhead forwards the query to its parent for processing. Eventually a clusterhead is reached that covers the area that contains both the query source and query point. This clusterhead becomes the Peer-Tree root node for processing the query.

The traditional branch-and-bound algorithm [10] is executed by the root node. Beginning with the child MBRs of the root, the partition list is sorted by MINDIST and the Peer-Tree is recursively traversed while a NN leaf node candidate is maintained and used for pruning MBRs. Supporting KNN queries with Peer-Tree is more complicated and not discussed by the Peer-Tree authors. For Peer-Tree to execute the query, it must be sent to the parent of the highest clusterhead required for finding the NN in order to guarantee that all candidate nodes will be evaluated (unless the query is already at the root clusterhead). At this point, the same branch-and-bound technique is employed except that a sorted buffer of at most k nearest neighbors is maintained and pruning is done according to the distance of the furthest nearest neighbor in this buffer.

There are several problems with the Peer-Tree approach. First, query messages must typically be routed through several layers of clusterheads. Transmission between clusterheads is executed largely independently of the physical geographic direction and distance. Depending on the network topology and the locations of clusterheads, it is possible that many unnecessary hops are included when routing messages towards query points. Furthermore, the clusterheads become communication bottlenecks where network congestion is likely (depending on the rate of submitted queries) especially if the distances between clusterheads is large and additional transmitting power is required. Ad-

ditionally, adding hierarchical infrastructure to sensor networks is inherently problematic since sensor networks are highly unstable. To handle the issue of fault tolerance, the authors propose using a lease period for all clusterhead nodes so that the hierarchical infrastructure is re-evaluated periodically.

3 KNN Perimeter Tree

Our hypothesis is that geographical routing algorithms such as GPSR can be used to approach shortest-path routing such that overall improved performance and fault tolerance is possible for KNN queries. Minimizing the individual responsibilities of sensor nodes makes the network less vulnerable to failure since there are no critical nodes in the network. Furthermore, less communication is necessary to maintain index or topology information in the network.

The KNN Perimeter Tree (KPT) builds upon GPSR [4] for processing KNN queries. KPT is deployed at all sensor nodes during network deployment. GPSR can successfully deliver messages to the nearest neighbor of any query point in the network. Since data is only available at the sensor nodes that generate them, a query need only be routed to the sensor nodes that own the data. All nodes in the network may participate in processing/forwarding queries.

The KPT algorithm can be broken down into phases as follows:

1. find the nearest neighbor and a maximum KNN boundary;
2. find $k-1$ nearest neighbors;
3. disseminate and execute query;
4. return result.

3.1 Find NN and a Maximum KNN Boundary

The query message is geographically routed from the query source towards the query point specified in the query. Based on GPSR, the message will eventually reach the geographically nearest neighbor to the query point. This node is designated as the *home node* of the KNN query. The home node is assigned temporary responsibilities for organizing the dissemination of the query and processing the results. This responsibility does make the home node vulnerable to node failure however only for the short duration of the time needed to process the query.

To avoid flooding a query to the whole network, a maximum KNN boundary is estimated to restrict the search space for finding the remaining $k-1$ nearest neighbors. We consider several approaches for determining this boundary while the query message is being routed to the home node. These approaches seek to determine a circular boundary in terms of a radius distance centered at the query point which is guaranteed to contain the KNN sensor nodes and the approaches have different tradeoffs.

An intuitive approach (called SUMDIST) for determining the boundary is to add the position of each sensor node on the forwarding path from the query source to the home node to a list in the query message. When the home node is reached, the distance between the home node position and the k-th position in the list serves as the maximum boundary. This approach has a higher communication cost since up to k locations are transmitted along with the query at every hop. For large values of k, this cost can be large.

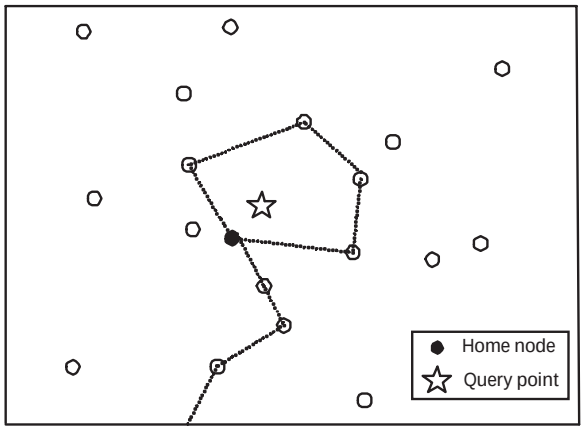


Figure 1: KPT home node and perimeter

A second approach (called MHD-1) includes only a counter variable, and a maximum hop distance (MHD) value which represents the largest distance value for any one hop on the route between the query source node and the home node. The counter variable is incremented at each forwarding hop until it reaches k. MHD always maintains the largest hop distance visited. After the query message reaches the home node, the maximum KNN boundary value can be determined by multiplying the MHD value by k. The advantage of this approach is that the cost of determining the maximum KNN boundary is less than the naive approach since only a few values are transmitted with the query message (independent of k). However, the search boundary is likely to be larger (and thus less efficient) than the boundary obtained from the naive approach.

An improvement on the second approach (called MHD-2) is to minimize the MHD value by plotting the hop distance along the direct path between the query source and query destination using geometry instead of taking the direct hop distance between neighbor nodes. However, the location of the query source node has to be added to the query message at an additional energy cost.

An assumption that is made for all three methods is that at least k hops occur on the route between the query source and the home node. Therefore, it is necessary to consider the case when fewer than k hops occur. To solve this problem we estimate the boundary by taking the MHD value and multiplying it by k (even for the naive approach). We believe that this estimation should be fairly good for many cases; however in implementing the KPT algorithm, we must consider the case when the estimation fails.

Figure 1 demonstrates the state of the KPT algorithm after the query has been routed to the nearest neighbor home node and the perimeter has been established. The query point is illustrated with a star and the home node which connects the incoming geographical route with the perime-

ter route is solid.

3.2 Find $k-1$ Nearest Neighbors

Given that the query is at the home node which knows the maximal KNN boundary, the next step is to determine the IDs and locations of the $k-1$ nearest neighbor nodes. A naive approach is to simply flood the query to all nodes within the circular KNN boundary centered at the query point. However, flooding expends excess energy, particularly if nodes are densely packed with much overlapping of radio and sensing ranges.

We propose the Perimeter Tree which is designed to reduce the number of total messages required to determine the $(k-1)$-NN nodes and for disseminating the query to them. The philosophy of this approach is to divide the boundary circle into regions for each of which a minimum spanning tree can be constructed that is rooted at a perimeter node. The subtrees expand in the direction away from the destination. The individual trees are bounded by the circular boundary and the two subtree boundaries on both sides.

The perimeter nodes that encircle the query point each make up a root of a minimum spanning tree that expands away from the destination and is bounded by the circular KNN boundary. The perimeter nodes are determined when the query message is transmitted by the home node in GPSR perimeter mode to validate the home node as the NN to the query point similar to the Perimeter Refresh Protocol in GHT [9]. At each hop around the perimeter, the midpoint on the line between Perimeter nodes is computed and by plotting a line from the query point through the midpoint to the circular boundary the subtree boundaries are determined, similar to a Voronoi cell [12].

The next step is to establish the spanning trees in each of the bounded areas that are rooted at the perimeter nodes. The goal is to build a tree with as few messages transmitted as possible and with also the shortest possible latency. By having multiple trees rooted at the perimeter nodes instead of one tree rooted at the home node the maximum height of the trees is reduced which reduces the overall query latency, although in highly irregular networks balancing the tree may not be possible which would affect the query latency but not the correctness. The construction of the tree begins with the perimeter root node which knows the query point, the two subtree boundaries (the midpoints between it and its two perimeter neighbors) and the circular KNN boundary. At a minimum, this information is transmitted to its potential children along with other information specified in Phase 3. In a tree, nodes only have one parent and belong to a certain level of the tree. Finally, a child node responds to its parent after hearing from its children and transmitting all node level information including node IDs and locations. This information is forwarded to the perimeter root which then transmits it to the home node. The home node then has all the locations of all nodes within the circular KNN boundary which it can then sort by their distance from the query point and thus determine the KNN node set.

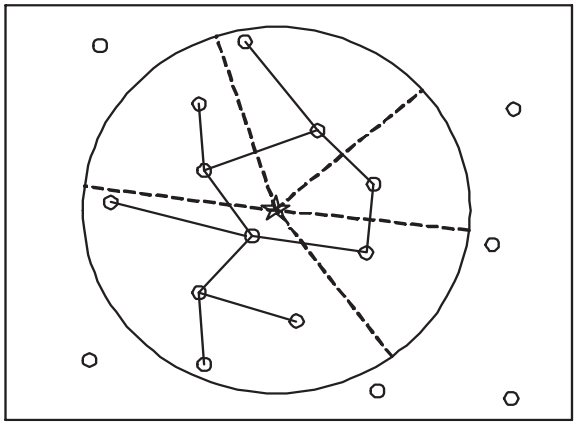


Figure 2: KNN Perimeter Tree

The perimeter boundaries are employed in order to keep the tree as balanced as possible and thus reduce the overall query latency. However, strictly enforcing this boundary for construction of the tree may exclude nodes that are within the circular boundary but are out of communication range of all potential parent nodes within its median boundary. Therefore we allow nodes to select a parent outside its tree boundary, but only if it does not hear a request from another potential parent from within its tree boundary. Although it may be possible for a sensor node to exist within the circular boundary and be completely disconnected from all other nodes within the circular boundary, it is unlikely. Furthermore, this would tend to happen towards the edge of the circular boundary reducing the probability that the disconnected node belongs to the KNN set.

Figure 2 demonstrates the state of the KPT after the Perimeter Tree has been established. The perimeter nodes are used to construct the tree boundaries to minimize the total height of the tree.

3.3 Disseminate and Execute Query

After Phase 2, the home node is aware of the IDs and locations of the KNN nodes. The next step is for the query to be disseminated for execution. A naive approach is for the home node to unicast or multicast using the Perimeter Tree the query to the KNN nodes. In order to reduce the overall latency, we propose combining the query dissemination with the Perimeter Tree establishment from Phase 2. As the Perimeter Tree is constructed, the actual query is transmitted to all tree members for automatic execution. This approach should have drastically improved latency, but less efficient energy performance since more than the KNN nodes actually execute the query. Imposing a quota system on the number of nodes to execute a query per subtree can reduce the execution cost without increasing the latency. The quota estimation method assigns the top q nodes of every subtree to execute the query automatically where q is a quota estimation defined in Equation (1) and p is the number of perimeter nodes and c is an adjustable parameter which trades off the quota size and the number of retransmissions needed when quota estimations fail.

$$q = \frac{k}{p} + c \quad (1)$$

The q value is set by the perimeter root node and decremented as it is assigned to nodes farther down the tree. The nodes assigned to execute the query do so and return the results back to the home node as the tree is constructed. The remaining nodes in the tree that are not assigned by the quota to execute the query automatically simply return location information.

After the tree is constructed, the home node receives the $p \times q$ results along with all the location and ID results from all nodes within the circular KNN boundary. The home node determines the KNN node set and whether the quota results include all necessary data to satisfy the KNN query. If any members of the KNN node set did not return quota estimation results, then the quota failed and must be resolved. The resolution can be handled simply by unicasting the query to the missing nodes and routing the results back, adding additional overhead and latency and is thus undesirable. The c parameter can be adjusted by experiment to determine the appropriate quota size. Flooding is used to execute the query if the circular boundary is underestimated using one of the MHD methods which adds considerable energy and latency costs. However, we feel that this situation will be rare.

3.4 Return Results

After the home node has collected the query results, it needs to transmit them back to the query source by unicasting the results geographically using GPSR. The Perimeter Tree can be destroyed after the location information has been returned to the home node. We reiterate that the Perimeter Tree only exists for a short period of time and therefore is only vulnerable to node failure very briefly unlike Peer-Tree.

4 Preliminary Performance Analysis

To give an idea of the capabilities of KPT versus Peer-Tree, we performed a mathematical analysis on both approaches in terms of the number of messages required to execute a query. For the analysis, we assume that nodes are uniformly distributed. To determine the cost processing KNN queries with KPT and Peer-Tree, we define some parameters which are listed in Table 1.

For analyzing the performance of KNN query processing, we break the execution into three phases for both KPT and PT:

- Phase 1 consists of the number of messages required to reach the home node for KPT or the Peer-Tree MBR root node.
- Phase 2 represents the cost of executing the query by getting the query to the KNN nodes and returning the results back to the Phase 1 home node.

Variable	Definition
h	Height of Peer-Tree
l	Average distance between nodes
n	Number of nodes in network
x	Number of nodes in KNN PT MBR
f	MBR fanout ($.69 \times M$)
s	Square axis of network ($s \times s$)
d	Average query distance
k	Number of nearest neighbors required
m	Minimum children per MBR
M	Maximum children per MBR
P_i	Probability a PT node is accessed at level i

Table 1: Summary of Parameters for Analysis

- Phase 3 represents the cost of returning the query results back to the query source node.

Estimating the query execution cost for KPT is fairly simple. For phases 1 and 3, we can estimate the number of hops required to route a message to the query source node and the home node and back by using the expression $\frac{d}{l}$. For phase 2, we estimate the number of messages as two messages per node inside the circular boundary. We can compute the average number of nodes inside the circular boundary by dividing the area of the circular boundary by the average area per sensor node (density) and thus we define the number of messages as $2 \times (\pi \times (k \times l)^2)/((s^2/n))$.

Performance analysis of Peer-Tree is more complicated. We refer to the analysis of KNN queries for R*-Trees [13] which is similar to Peer-Tree except that message transmissions are used instead of disk accesses when information from a node is needed. For phases 1 and 3, the number of messages required to transmit the query message to the root parent node and the results back is the number of levels in the tree from level 0 to the level of the root parent. The level of the root parent is one above the smallest MBR that contains the query point and the query source node. For estimating the size of the smallest MBR that contains the query point and the source node we assume an average square-shaped MBR where the query distance d makes up half the bisecting hypotenuse with an area of $2 \times d^2$. The number of sensor nodes contained within the parent of the MBR that spans the source node and query point can be estimated as $x = (h \times 2 \times d^2)/((s^2)/(n))$. We can determine the height of the tree needed to execute the query as $h = 1 + \lceil \log_f(\frac{x}{M}) \rceil$ [13].

For computing the cost of phase 2 for Peer-Tree, we use the same formula for node accesses defined as $\sum_{i=0}^{h-1} (n_i \times P_i)$ where h is the height of the tree, P_i is the probability that a node at level i is accessed and n_i is the total number of nodes at level i [13]. Due to the space constraints of this paper, we leave the details to [13]. Two messages are required for each node access, one to deliver the query and one for a response.

For constructing experiments using the mathematical analysis the following default parameters were used. A network size of 100×100 meters^2 was used with a node

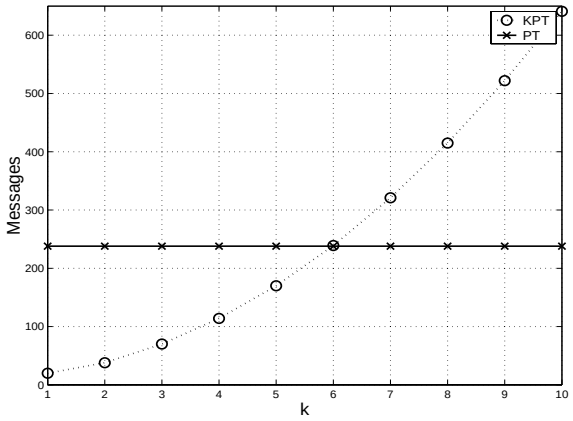


Figure 3: Experiment 1: Effect of k

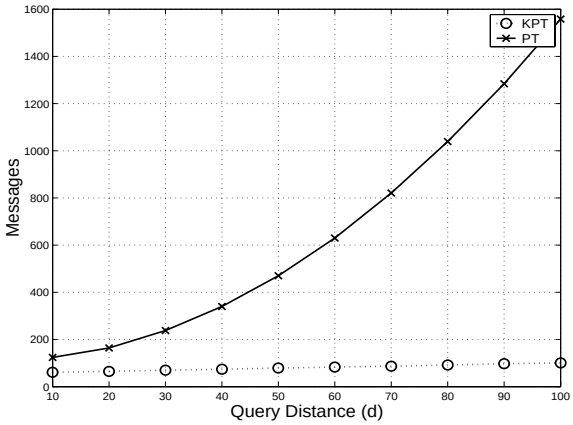


Figure 4: Experiment 2: Effect of query distance

density of 500 uniformly distributed sensors. The average query distance used was 30 meters with a k value of 3. For Peer-Tree, each MBR contained between 3 and 6 children. The metric used for analysis was simply the number of messages required to execute the query for KPT and Peer-Tree.

Figure 3 demonstrates the effect of k on the performance of KPT and Peer-Tree. The results show that while Peer-Tree is not affected by the value of k, KPT performs better for lower k values, specifically with k smaller than 6. This makes sense since the larger the k value, the larger the circular query boundary which includes more nodes in the query.

Figure 4 shows the effect of the query distance on the execution performance of both approaches. The effect of the query distance on KPT is minimal; only a very small linear increase for KPT while Peer-Tree suffers an exponential increase in the number of messages as the query distance increases. This is due to the fact that the size of the spanning parent MBR grows much larger and the height of the tree increases as well. Although not demonstrated here, Peer-Tree is also affected by the size of the child node capacity and the node density of the network.

We acknowledge that this analysis is primitive by simply counting the number of messages of an individual query and does not take into account that the messages for Peer-Tree would likely have to be transmitted at a higher power level and are thus more expensive. The size of the messages, per-bit cost of transmission and query execution costs are also not considered here. Most importantly, this analysis assumes that all required infrastructure for Peer-Tree is in place, i.e., the considerable cost for constructing and maintaining the tree is not demonstrated. Nonetheless, KPT is able to perform often significantly better than Peer-Tree for executing KNN queries. Fault tolerance to node failure is also not demonstrated. Considering fault tolerance and actual energy consumption will be demonstrated through simulation in our future work.

5 Conclusion

We believe that KPT shows potential for improving performance in terms of energy consumption and latency for processing KNN queries in sensor networks. Our preliminary analysis shows that KPT can achieve significant energy savings over Peer-Tree in terms of the number of messages required to execute a KNN query without even comparing the costs required to construct and maintain the Peer-Tree infrastructure when compared to the minimal neighbor information required for geographical routing. Additionally, although not demonstrated through analysis, KPT intuitively is more fault tolerant than Peer-Tree.

For the future work of this project, simulation experiments are under construction that are designed to back up the claims of this paper. Additionally, further improvements of KPT may be possible if assumptions can be made about the node distribution. Furthermore, we intend to also investigate the use of KNN queries in mobile sensor network environments by employing routing protocols for dynamic networks. Finally, we intend to consider supporting Type 2 KNN queries with KPT.

References

[1] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proc. of the 3rd IEEE International Conference on Peer-to-Peer Computing*, Linkping, Sweden, September 2003.

[2] B. Greenstein, D. Estrin, R. Govindan, S. Ratnasamy, and S. Shenker. DIFS: A distributed index for features in sensor networks. In *Proceedings of the IEEE ICC Workshop on Sensor Network Protocols and Applications*, Anchorage, AK, April 2003.

[3] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD Conference*, pages 47–57, 1984.

[4] B. Karp and H.T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, 2000.