Major Project

On

Attribute Based Routing For Query Processing To Minimize Power Consumption in Wireless Sensor Networks

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science & Engineering

By

Patel Bhavik M.
(06MCE023)

Under Guidance of

Dr. S.N. Pradhan

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERCITY OF SCIENCE & TRCHNOLOGY
Ahmedabad 382481
May 2008

This is to certify that Dissertation entitled

Attribute Based Routing For Query Processing To Minimize Power Consumption in Wireless Sensor Networks

Submitted by

Bhavik Patel

has been accepted toward fulfillment of the requirement

for the degree of

Master of Technology in Computer Science & Engineering

Dr. S. N. Pradhan	Prof. D. J. Patel
Professor	Head of The Department

Prof. A. B. Patel

Director, Institute of Technology

CERTIFICATE

This is to certify that the work presented here by **Mr. Bhavik Patel** entitled "Attribute Based Routing For Query Processing To Minimize Power Consumption In Wireless Sensor Networks**"** has been carried out at **Institute Of Technology, Nirma University** during the period **September 2007 – May 2008** is the bonafide record of the research carried out by him under my guidance and supervision and is up to the standard in respect of the content and presentation for being referred to the examiner. I further certify that the work done by him is his original work and has not been submitted for award of any other diploma or degree.

Dr. S. N. Pradhan
Guide,
Professor, Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad.

Date:

ACKNOWLEDGEMENTS

The successful completion of a project is generally not an individual effort. It is an outcome of the cumulative efforts of a number of persons, each having own importance to the objective. This session is a vote of thanks and gratitude towards all those persons who have directly or indirectly contributed in their own specials way towards the completion of this project.

It gives me great pleasure in expressing thanks and profound gratitude to Dr. S. N. Pradhan, M.Tech In-Charge, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout the project. I heartily thankful him for his time to time suggestions and the clarity of the concepts of the topic that helped me a lot during the project.

I would like to give my special thanks to Prof. D. J. Patel, Head, Department of Computer Engineering, Institute of Technology, Nirma University for his continual kind words of encouragement and motivation throughout the Project. I am thankful to all faculty members for their special attention and suggestion towards the project work.

I extend my sincere thanks to my colleagues for their support in my work. I would like to express my gratitude towards my family members who have always been my source of inspiration and motivation.

Bhavik Patel
(06MCE023)

ABSTRACT

A wireless sensor Network consists of sensors implanted in an environment for collecting & transmitting data regarding changes in the environment based on the requests from controlling device or base station using wireless communication. WSNs are being used in medical, military and environment monitoring applications. Query dissemination and gathering of information towards central node are important communication paradigms across all application domains.

Power consumption constraints in WSNs require task specific queries to adopt efficient data centric routing schemes. This project proposes new routing methods for wireless sensor network queries. These methods draw parallels from and extend the notion of making the routing protocol aware of the underlying network semantics for queries. The proposed methods reduce the communication overhead & power consumption by reducing number of sensor nodes participating nodes required to participate in certain queries in order to successfully answer to user specific queries in WSNs.

In most instances of sensor network deployment, the sensory attribute values (say, light, temperature, etc…) often vary within a certain range of possible values termed the sensory range or the attribute-value range. It is possible to design a protocol that takes the query range and the historical value change information into account to determine if a particular node or set of nodes should participate in the query? Making this as the base, the work presented here uses that feature to develop new routing scheme for energy efficient processing of attribute specific queries. TOSSIM (the TinyOS Simulator) is used to implement and simulate routing schemes.

CONTENTS

Certificate II
Acknowledgement IV
Abstract V
Contents VI
List Of Figures VIII
List Of Abbreviations X

Chapter 1 Introduction 1
1.1 General 1
1.2 Motivation 2
1.3 Related Work 4
1.4 Outline of Thesis 6

Chapter 2 Literature Survey 8
2.1 Wireless Sensor Networks' Architecture 8
2.2 Query Processing In Wireless Sensor Networks 11
2.2.1 Introduction 11
2.2.2 Queries 12
2.2.3 In-Network aggregation 13
2.3 TinyOS 14
2.4 NesC Language 16
2.5 TOSSIM 17
2.6 TinyVIZ 19
2.7 TinyDB 21

Chapter 3 Project Scope and Description 23
3.1 TinyOS Routing Algorithm Analysis 23
3.2 TinyOS- WMEWMA Link Quality Estimator 26
3.3 Attribute Based Routing for non-constant attributes 26

3.4 Dynamic Semantic Routing 27
3.5 Project Task Summary 27

Chapter 4 System Design, Architecture and Implementation..... 28
4.1 Complete Picture of Implementation 28
4.2 Project Design 29
4.3 SRT Algorithm Analysis 30
4.4 Range Information Management 31
4.5 DSRT Algorithm Complexity Analysis 32
4.6 Cycle Prevention Techniques:- Hop Count & Layer 32
4.7 Layer Acquisition Algorithm 33
4.8 The Parent Selection Process 33
4.9 Manhattan Distance Based Parent Selection in DSRT 34
4.10 DSRT Parent Selection Process 36
4.11 Path Compression 36
4.12 Code Optimization 36

Chapter 5 Testing Results & Analysis 37
5.1 Test Environment Setup 37
5.2 Evaluation Metrics 37
5.3 Simple Comparison & Analysis 38
5.4 Test Results 40
5.5 Result Analysis & Conclusion 44

Chapter 6 Future Work 46
References 47

LIST OF FIGURES

Figure No.	Caption	Page No.
Figure 1.1	Sensor Network Communication Architecture	2
Figure 2.1	Protocol Stack for WSNs	8
Figure 2.2	A Berkeley MICA Mote	10
Figure 2.3	Hardware Characteristics of a MICA Mote	10
Figure 2.4	Query Template	12
Figure 2.5	Example Aggregate Query	13
Figure 2.6	TinyVIZ Example	20
Figure 2.7	TinyDB Example	22
Figure 3.1	TinyOS Routing Algorithm	24
Figure 3.2	Neighbor Table	25
Figure 3.3	Routing Messages	25
Figure 4.1	System Architecture	28
Figure 4.2	SRT Algorithm	30
Figure 4.3	DSRT Algorithm	31
Figure 4.4	DSRT Routing Message	32
Figure 4.5	Layer Acquisition Algorithm	32
Figure 4.6	A TinyOS Routing Tree	33
Figure 5.1	Light Ranges	38
Figure 5.2	TinyOS/SRT Routing Tree	39
Figure 5.3	DSRT Routing Tree	39
Figure 5.4	Test Case 1	41

Figure 5.5	Test Case 2	41
Figure 5.6	Test Case 3(QR)	42
Figure 5.7	Test Case 3(PR)	42
Figure 5.8	Test Case 4	43
Figure 5.9	Test Case 5(QR)	43
Figure 5.10	Test Case 5(PR)	43
Figure 5.11	Test Case 6	44
Figure 5.12	Test Case 7(QR)	44
Figure 5.13	Test Case 7(PR)	44

LIST OF ABBREVIATIONS

WSNs	Wireless Sensor Networks
TinyOS	Tiny Operating System
NesC	Network Embedded System C
TOSSIM	TinyOS Simulator
TinyVIZ	Tiny Visualization
TinyDB	Tiny Database
SRT	Semantic Routing Tree
DSRT	Dynamic Semantic Routing Tree
WMEWMA	Window Means with Exponentially Weighted Moving Average
BTM	Messages to Build-up Tree
QM	Query Messages disseminated to the whole network
RTT	Round Trip Time for a certain query
LRTT	Last Response Time
FRTT	First Response Time
RP	Response Percentage
PR	Participation Rate
NT	Network Topology

1. INTRODUCTION

1.1 General

Future applications of sensor networks are plentiful. In the intelligent building of the future, sensors are deployed in offices and hallways to measure temperature, noise, light, and interact with the building control system. People can pose queries that are answered by the sensor network, such as "Is Bhavik in his office", or "Is there an empty seat in the conference room?" Another application is scientific research. As an example, consider a biologist who may want to know of the existence of a specific species of birds, and once such a bird is detected; the bird's trail should be mapped as accurately as possible. In this case, the sensor network is used for automatic object recognition and tracking. More specific applications in different fields will arise, and instead of deploying preprogrammed sensor networks only for specific applications, future networks will have sensor nodes with different physical sensors for a wide variety of application scenarios and different user groups.

A sensor network is composed of a large number of sensor nodes that are densely deployed either inside the phenomenon or very close to it. The position of sensor nodes need not be engineered or predetermined. This allows random deployment in inaccessible terrains or disaster relief operations. On the other hand, this also means that sensor network protocols and algorithms must possess self-organizing capabilities. Another unique feature of sensor networks is the cooperative effort of sensor nodes. Sensor nodes are fitted with an onboard processor. Instead of sending the raw data to the nodes responsible for the fusion, they use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data.

The above described features ensure a wide range of applications for sensor networks. Some of the application areas are health, military, and home. In military, for example, the rapid deployment, self-organization, and fault tolerance characteristics of sensor networks make them a very promising sensing technique for military *command*, *control*, *communications*, *computing*, *intelligence*, *surveillance*, *reconnaissance*, and *targeting systems*.

In health, sensor nodes can also be deployed to monitor patients and assist disabled patients. Some other commercial applications include *managing inventory*, *monitoring product quality*, and *monitoring disaster areas*.

Many protocols and algorithms have been proposed for traditional wireless ad hoc networks, they are not well suited to the unique features and application requirements of sensor networks. To illustrate this point, the differences between sensor networks and ad hoc networks are:

- Sensor nodes are prone to failures.
- The topology of a sensor network changes very frequently.
- Sensor nodes are limited in power, computational capacities, and memory.

Many researchers are currently engaged in developing schemes that fulfill these requirements.

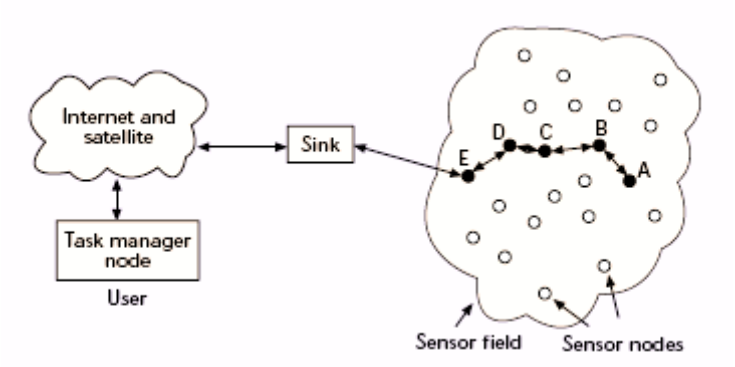


Fig 1.1: Sensor Network Communication Architecture [11]

The sensor nodes are usually scattered in a *sensor field* as shown in Fig. 1.1. Each of these scattered sensor nodes has the capabilities to collect data and route data back to the *sink*. Data are routed back to the sink by a multihop infrastructure less architecture through the sink as shown in Fig. 1.1. The sink may communicate with the *task manager node* via Internet or satellite. The design of the sensor network as described by Fig. 1.1 is influenced by many factors, including *fault tolerance*, *scalability*, *production costs*, *operating environment*, *sensor network topology*, *hardware constraints*, *transmission media*, and *power consumption*.

1.2 Motivation

In this thesis a general query processing middleware is proposed and implemented that operates on top of an existing query-based sensor network.

Approach is motivated by the following design goals. Power consumption constraints in wireless sensor networks require task-specific queries in distributed sensor networks to adopt efficient data centric routing techniques. This project proposes new routing methods tailored for sensor network queries. These routing schemes draws several parallels from and extends the notion of making the routing protocol aware of the underlying network semantics for aggregate queries. Madden et al initially envisioned one such approach [1]. The proposed methods reduce the communication overhead and power consumption by reducing the number of sensor nodes required to participate in certain queries.

In most instances of sensor network deployment, the sensory attribute values (say, temperature, light, etc…) often vary within a certain range of possible values termed as the sensory range or the attribute-value range. The queries themselves on the other hand often request node participation based on a physical range (e.g. all sensors located east of a particular region). Is it possible to design a protocol that takes the query range and the historical value change information into account to determine if a particular node or a set of nodes should participate in the query response? In this project we describe how the proposed routing schemes enable us to achieve this objective. We first record a value range for sensory attributes (say, temperature between some specified limit). Later, whenever a range query requests information from sensor nodes whose historical range is beyond that of the query predicates, the nodes will automatically not participate in the query. The project implements two routing algorithms in the application layer of TinyOS protocol stacks and compares them with TinyOS routing algorithm based on testing results of different topologies and radio-range models, in terms of message overhead flooded into the network.

Second design goal is motivated by the importance of preserving limited resources, such as energy and bandwidth in battery-powered wireless sensor networks. Data transmission back to a central node for offline storage, querying, and data analysis is very expensive for sensor networks of non-trivial size since communication using the wireless medium consumes a lot of energy. Since sensor nodes have the ability to perform local computation, part of the computation can be moved from the clients and pushed into the sensor network, aggregating records, or eliminating irrelevant records. Compared to traditional centralized data extraction and analysis, In-network processing can reduce

energy consumption and reduce bandwidth usage by replacing more expensive communication operations with relatively cheaper computation operations, extending the lifetime of the sensor network significantly. For example, the ratio of energy spent in sending one bit versus executing one instruction ranges from 220 to 2900 in different architectures .Thus the second main role of the query layer is to perform in-network processing. Different applications usually have different requirements, from accuracy, energy consumption to delay. For example, a sensor network deployed in a battlefield or rescue region may only have a short life time but a high degree of dynamics. On the other hand, for a long-term scientific research project that monitors an environment, power-efficient execution of long-running queries might be the main concern. More expensive query processing techniques may shorten processing time and improve result accuracy, but might use a lot of power. The query layer can generate query plans with different tradeoffs for different users.

Third & final design goal is to support multi-query optimization and tries to reduce the number of queries that are sent to the sensor network by making use of commonalities among queries. This also reduces the number of messages that must be sent in the network, thus the overall energy consumption can be reduced and the battery lifetime extended.

1.3 Related Work

Network routing is crucial to the efficiency of data communication and consequently, has a direct impact on the robustness of the sensor network application. In fact, there are various routing protocols proposed and deployed within different network communication layers. Data-centric protocols often reside in the network layer of the communication stack. This refers to the fact that sources (sensor nodes) aggregate or consolidate the data they gather before they relay the data to the destination. By assigning a data related property with each sensor node, the intended communication can be directed only to a specific number of sensor nodes. Now briefly describe three such related data-centric routing protocols.

• Directed Diffusion [2] introduces the concept of interest- a data structure to describe an event- attributes like event type, time stamp, and location. Every node keeps an entry of all interests it ever sees and gradient, a data structure to

describe the path of the event occurrence. For example interest could include data rate and neighbor nodes. Data propagation is determined by interest and gradient. Suppose node 'A' wants to know something about event 'e', or say interest 'I'. It then floods the interest to all neighbors. Every neighbor/node then checks their local interest and gradient cache to see if one of their interests matches the coming one. If so, send the interest to that neighbor which has the highest data rate in the gradient. Thus, the request of the interest 'I' gets propagated through the network. Directed Diffusion also advocates path reinforcement, which refers to the sink enforcing the routing path on neighbors with the highest event data rate so that the sink receives high quality data from high quality nodes.

• Information directed routing [3] - For task-specific sensor networks (e.g., .a particular type of monitoring), information directed routing provides downstream routing (from sink to source) based on a new information model. The model formulates the measurement of information contribution using Kullback-Leibier divergence and Bayesian filtering method. For all neighbors, the model can calculate information distribution based on time and current belief without knowing the sensor data. Thus routing becomes a shortest path problem where weight of the edges is communication cost between two sensor nodes; and communication cost includes additive information contribution of the current path. Note that since the information contribution is state-dependent, the concurrent path contribution needs to be taken into account.

• Rumor routing [4] – improves the efficiency of upstream routing (from source to sink) by eliminating flooding messages. Once an event happens, an agent, which is a long-live packet, will keep track of the event and travel around the network. Supposing there is some event happening in the network, instead of all the nodes witnessing the event and flooding the network with messages, an agent which is a long-live packet keeps list of all events and travels around the network to propagate information happening inside the networks.

In [5] the authors have also recognized the necessity of an abstraction layer between the network and the user. They also propose layered database for sensor networks. Their focus is on aggregations and joins operations in the network using chain of flow blocks. However they do not address the problem of multi-query optimization, i.e., reconfiguring the query plan of running queries. It is not clear how multiple concurrent queries are handled. In principle it might be

possible to reuse results obtained by flow blocks for other queries; however it is unknown how to convert between sampling rates of different queries.
The description given in [6] is restricted to user queries having the same result frequency. In the report the authors present a multi query optimization algorithm for aggregate queries with the main aim in the reduction of the energy consumption. They approach query optimization by a linear reduction technique. Their query manager does not immediately send new queries into the network as they arrive; instead they are gathered and then dispatched together. Then data is captured for several epochs until a new set of user queries is ready to be dispatched. Thus the execution plan needs to be recomputed and propagated anew for every group of queries. The query optimizer described in this thesis adjusts these management cost by choosing between a "universal" or a specific query.

1.4 Outline of thesis

This thesis is organized as follows:

- Chapter 2 provides the brief introduction about wireless sensor node's architecture including with details of protocol stack implemented for sensor or sink nodes. It also includes the brief details about query processing in wireless sensor networks and further enhancements that must be taken into account for research in that direction. It also describes the details about TinyOS (the embedded Operating System specially designed for Sensor nodes), NesC (Network Embedded System C – a programming language for embedded sensor nodes), TOSISM (TinyOS SIMulator), TinyVIZ (Visualization Plug-in for TOSSIM especially for sensor networks) and TinyDB (RDBMS for Senor Networks).

- Chapter 3 explains the working of TinyOS routing algorithm and also analyzes the complexity of the algorithm. It also introduces the schemes to achieve efficient routing in wireless sensor networks for attribute specific queries.

- Chapter 4 contains the details about the implementation of the routing scheme including variants like range management, layer acquisition, parent selection process techniques.

- Chapter 5 presents the results for different test cases in TOSSIM based simulation. It also concludes based on the results achieved for the implementation of the project.

- Chapter 6 provides possible directions for relevant future research.

2. LITERATURE SURVEY

2.1 Wireless Sensor Networks' Architecture

In recent years, rapid advances in wireless networking, micro-electronic devices, and embedded systems have enabled the development of low-cost, low-power, small sized sensor nodes. The sensor technology is suitable for a wide range of applications ranging from environmental control, warehouse inventory, and health care to military environments, establishing ubiquitous wireless sensor networks that rapidly evolve human life.

A sensor network is composed of a large number of sensor nodes with sensing, computing, and communicating capabilities. These nodes are usually powered by battery and operate in an unattended fashion to collect and process information.

The protocol stack used by the sink and sensor nodes is given in Fig. 2.1.This protocol stack combines power and routing awareness, integrates data with networking protocols, communicates power efficiently through the wireless medium, and promotes cooperative efforts of sensor nodes. The protocol stack consists of the *physical layer*, *data link layer*, *network layer*, *transport layer*, *application layer*, *power management plane*, *mobility management plane*, and *task management plane*.

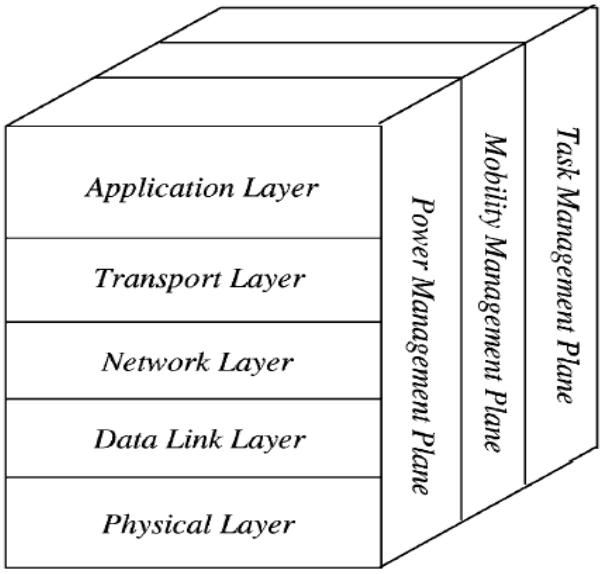


Figure 2.1: Protocol stack for WSNs [11]

The physical layer addresses the needs of simple but robust modulation, transmission, and receiving techniques. Since the environment is noisy and

sensor nodes can be mobile, the medium access control (MAC) protocol must be power-aware and able to minimize collision with neighbors' broadcasts. The network layer takes care of routing the data supplied by the transport layer. The transport layer helps to maintain the flow of data if the sensor networks application requires it. Depending on the sensing tasks, different types of application software can be built and used on the application layer.

In addition, the power, mobility, and task management planes monitor the power, movement, and task distribution among the sensor nodes. These planes help the sensor nodes coordinate the sensing task and lower overall power consumption.

The power management plane manages how a sensor node uses its power. For example, the sensor node may turn off its receiver after receiving a message from one of its neighbors. This is to avoid getting duplicated messages. Also, when the power level of the sensor node is low, the sensor node broadcasts to its neighbors that it is low in power and cannot participate in routing messages. The remaining power is reserved for sensing. The mobility management plane detects and registers the movement of sensor nodes, so a route back to the user is always maintained, and the sensor nodes can keep track of who their neighbor sensor nodes are. By knowing who the neighbor sensor nodes are, the sensor nodes can balance their power and task usage. The task management plane balances and schedules the sensing tasks given to a specific region. Not all sensor nodes in that region are required to perform the sensing task at the same time. As a result, some sensor nodes perform the task more than others depending on their power level. These management planes are needed so that sensor nodes can work together in a power efficient way, route data in a mobile sensor network, and share resources between sensor nodes.

Recent developments in hardware have enabled the widespread deployment of sensor networks consisting of small sensor nodes with sensing, computation, and communication capabilities. Already today networked sensors measuring only a few cubic inches can be purchased commercially, and Moore's law tells that it will be soon seen components that measure $1/4^{th}$ of a cubic inch, running an embedded version of a standard operating system; such sensor nodes have the following resource constraints:

• **Communication:** The wireless network connecting the sensor nodes provides usually only a very limited quality of service, has latency with high variance, limited bandwidth, and frequently drops packets.

• **Power consumption:** Sensor nodes have limited supply of energy, and thus energy conservation needs to be of the main system design considerations of any sensor network application. For example, the MICA motes are powered by two AA batteries, which provide about 2000mAh, powering the mote for approximately one year in the idle state and for one week under full load.

• **Computation:** Sensor nodes have limited computing power and memory sizes. This restricts the types of data processing algorithms on a sensor node, and it restricts the sizes of intermediate results that can be stored on the sensor nodes.

• **Uncertainty in sensor readings:** Signals detected at physical sensors have inherent uncertainty, and they may contain noise from the environment. Sensor malfunction might generate inaccurate data, and unfortunate sensor placement (such as a temperature sensor directly next to the (air conditioner) might bias individual readings.

Figure 2.2 A Berkeley MICA Mote [A]

Processor	4Mhz, 8bit MCU (ATMEL)
Storage	512KB
Radio (RF Monolithic)	916Mhz Radio
Range	100 FT
Data Rate	40 Kbits/sec
Transmit Current	12mA
Receive Current	1.8mA
Sleep Current	5 uA

Fig.2.3. Hardware Characteristics of a MICA Mote [A]

Future applications of sensor networks are plentiful. In the intelligent building of the future, sensors are deployed in offices and hallways to measure temperature, noise, light, and interact with the building control system. People can pose queries that are answered by the sensor network. Another application is scientific research. More specific applications in different fields will arise, and instead of deploying preprogrammed sensor networks only for specific applications, future networks will have sensor nodes with different physical sensors for a wide variety of application scenarios and different user groups.

2.2 QUERY PROCESSING IN WIRELESS SENSOR NETWORKS [7]

2.2.1 Introduction

Given the view of a sensor network as a huge distributed database system, it would like to adapt existing techniques from distributed and heterogeneous database systems for a sensor network environment. However, there are major differences between sensor networks and traditional distributed and heterogeneous database systems.

First, sensor networks have communication and computation constraints that are very different from regular desktop computers or dedicated equipment in data centers, and query processing has to be aware of these constraints. One way of thinking about such constraints is the analogous interaction with the file systems in traditional database systems, Database systems bypass the file system buffer to have direct control over the disk. For a sensor network database system, the analogous counterpart is the networking layer, and for intelligent resource management we have to ensure that the query processing layer is tightly integrated with the networking layer. Second, the notion of the cost of a query plan has changed, as the critical resource in a sensor network is power, and query optimization and query processing have to be adapted to take this optimization criterion into account.

While developing techniques that address these issues, it must not forget that scalability of our techniques with the size of the network, the data volume, and the query workload is an intrinsic consideration to any design decision.

2.2.2 Queries

SELECT {attributes aggregates}
FROM {Sensor data S}
WHERE {predicate}
GROUP BY {attributes}
HAVING {predicate}
DURATION time interval
EVERY time span e

[Fig 2.4 Query Template][C]

A sensor node has one or more sensors attached that are connected to the physical world. Example sensors are temperature sensors, light sensors, or PIR sensors that can measure the occurrence of events (such as the appearance of an object) in their vicinity. Thus each sensor is a separate data source that generates records with several fields such as the id and location of the sensor that generated the reading, a time stamp, the sensor type, and the value of the reading. Records of the same sensor type from different nodes have the same schema, and collectively form a distributed table.

The sensor network can thus be considered a large distributed database system consisting of multiple tables of different types of sensors. Sensor data might contain noise, and it is often possible to obtain more accurate results by fusing data from several sensors. Summaries or aggregates of raw sensor data are thus more useful to sensor applications than individual sensor readings. For example, when monitoring the concentration of a dangerous chemical in an area, one possible query is to measure the average value of all sensor readings in that region, and report whenever it is higher than some predefined threshold.

It believes that declarative queries are the preferred way of interacting with a sensor network. Rather than deploying application-specific procedural code expressed in a Turing-complete programming language, It believes that sensor network applications are naturally data-driven, and thus it can abstract the functionality of a large class of applications into a common interface of expressive queries. It considers queries of the simple form shown in query template, and we leave the design of a suitable query language for sensor networks to future work. It also extends the template to support nested queries, where the basic query block shown in Figure 3 can appear within the WHERE or HAVING clause of another query block. Query template has the obvious

semantics: The SELECT clause specifies attributes and aggregates from sensor records, the FROM clause specifies the distributed relation of sensor type, the WHERE clause filters sensor records by a predicate, the GROUP BY clause classifies sensor records into different partitions according to some attributes, and the HAVING clause eliminates groups by a predicate. Note that it is possible to have join queries by specifying several relations in the FROM clause.

```
SELECT AVG (R.concentration)
FROM ChemicalSensor R
WHERE R.loc IN region
HAVING AVG (R.concentration) > T
DURATION (now, now+3600)
EVERY 10
```

[Fig 2.5 Example Aggregate Query][1]

One difference between this query template and SQL is that our query template has additional support for long running, periodic queries. Since many sensor applications are interested in monitoring an environment over a longer time-period, long-running queries that periodically produce answers about the state of the network are especially important. The DURATION clause specifies the life time of a query and the EVERY clause determines the rate of query answers: It computes a query answer every e seconds. The process of computing a query answer is also known as a round.

2.2.3 In-Network Aggregation [1]

In Wireless Sensor Networks (WSNs), energy efficient query processing is typically performed by in-network query processing. In in-network query processing, a query gradually gets resolved within the network as sensors provide their sensed data. However, in-network query processing demands (i) the dissemination of detailed query information to all sensors, and (ii) sufficient query processing power at all sensors. These limit the application of in-network processing to certain types of queries, such as average or maximum value computation, only.

A query plan for a simple aggregate query can be divided into two components. Since queries require data from spatially distributed sensors, it needs to deliver records from a set of distributed nodes to a central destination node for aggregation by setting up suitable communication structures for delivery of

sensor records within the network. This part of a query plan is its communication component, and the destination node is the leader of the aggregation. In addition, the query plan has a computation component that computes the aggregate at the leader and potentially computes already partial aggregates at intermediate nodes. Recall that power is one of the main design desiderata when devising query processing strategies for sensor networks. If it coordinating both the computation and communication component of a query plan, it can compute partial aggregates at intermediate nodes as long as they are well-synchronized; this reduces the number of messages sent and thus saves power. Consider here three different techniques on how to integrate computation with communication:

Direct delivery: This is the simplest scheme. Each source sensor node sends a data packet consisting of a record towards the leader, and the multi-hop ad-hoc routing protocol will deliver the packet to the leader. Computation will only happen at the leader after all the records have been received.

Packet merging: In wireless communication, it is much more expensive to send multiple smaller packets instead of one larger packet, considering the cost of reserving the channel and the payload of packet headers. Since the size of a sensor record is usually small and many sensor nodes in a small region may send packets simultaneously to process the answer for a round of a query, it can merge several records into a larger packet, and only pay the packet overhead once per group of records. For exact query answers with holistic aggregate operators like Median, packet merging is the only way to reduce the number of bytes transmitted.

Partial aggregation: For distributive and algebraic aggregate operators, it can incrementally maintain the aggregate in constant space, and thus push partial computation of the aggregate from the leader node to intermediate nodes. Each intermediate sensor node will compute partial results that contain sufficient statistics to compute the final result.

2.3. TinyOS

The major bottleneck of the motes themselves is the power consumption and the low computational power. TinyOS has been designed to meet the requirements of sensor networks. Sensor values have to be processed in real time to avoid data loss. As the hardware contains physical parallelism, the OS must provide some kind of multithreaded architecture.

Even systems that call themselves real time need some hundreds of processor cycles to perform a context switch. On a small microprocessor like the Atmel AVR, used in both mica-family motes and the BTnode, this is unacceptable and would lead to data loss. TinyOS does not use a stack-based threaded architecture but an event based architecture. This allows it to have only one stack and a single execution context. In case a hardware component wants to deliver data, it calls an event that will preempt running tasks.

TinyOS [A] is based on nesC, a C dialect that is imperative at low level and declarative at high level. Applications are component based, code is encapsulated in components that are defined by the interfaces they provide and by those they can consume. Interfaces are bidirectional, they do not only define the commands that have to be implemented by the lower level that implements the interface, but also events that have to be implemented by the higher level that uses the component implementing the interface. The component based model decouples API and implementation and hides the specific properties of an implementation. At compile time, implementations can be substituted by different implementation, either in hardware or in software, only the wiring has to be changed. This makes TinyOS very flexible.

A core feature of TinyOS is the scheduling. TinyOS uses a simple queue with length 7 and a two level scheduling. Events have high priority and can preempt tasks, which have low priority. To avoid blocking scenarios, events and commands are expected to do only state transmissions and leave complex computations to tasks, which can be preempted if necessary. Hardware interrupts thrown by timers or by the radio module map to first level events. The paradigm for network transmissions in TinyOS is active messaging. Messages contain a handler address and on arrival this handler is called. Each node is expected to run the same handler code and can either redirect the message to a neighbor, if it is not the receiver, or start some local events as reaction to the message. Transmissions are best effort, if more is needed, it is up to the application to implement more sophisticated features like flow control or encryption. To send packages over the network, TinyOS uses multi-hop routing instead of point-to-point connections to save transmission power. Route discovery is done by 2-hop broadcast and topology discovery is based on shortest path from each node to the base station. A typical application for TinyOS is TinyDB, a RDBS interface for TinyOS sensor networks. In general, SQL

commands are transmitted from the base station down the tree to every node. Intermediate nodes collect the data from the children and transmit the aggregated data to the root node.
Sensor networks become more and more popular, in the meantime, global players like Intel are pushing the technology. TinyOS is designed to meet the requirements of small devices with small resources. The major bottleneck is the power consumption of the radio module so better routing algorithms could improve the system significantly. Static resource allocation is a direct result of the small resources but with computation power of current motes, dynamic resource allocation could be afforded. Additionally, systems to change the node's code at runtime might broaden the room for applications as currently nodes have to be collected and manually reprogrammed to change their general behavior.

2.4 NesC Language

NesC is a programming language for networked embedded systems that represent a new design space for application developers. An example of a networked embedded system is a sensor network, which consists of (potentially) thousands of tiny, low power "motes," each of which execute concurrent, reactive programs that must operate with severe memory and power constraints. NesC's contribution is to support the special needs of this domain by exposing a programming model that incorporates event-driven execution, a flexible concurrency model, and component-oriented application design. Restrictions on the programming model allow the nesC compiler to perform whole-program analysis, including data-race detection (which improves reliability) and aggressive function inlining (which reduces resource consumption).
NesC has been used to implement TinyOS, a small operating system for sensor networks, as well as several significant sensor applications. NesC and TinyOS have been adopted by a large number of sensor network research groups, it is effective at supporting the complex, concurrent programming style demanded by this new class of deeply networked systems.[D].
The primary concepts in nesC's design are first, nesC applications are built out of *components* with well-defined, bidirectional *interfaces*. Second, nesC defines a concurrency model, based on tasks and events, and detects data races at compile time.
A few basic principles underlie nesC's design:

NesC is an extension of C: C produces efficient code for all the target microcontrollers that are likely to be used in sensor networks. C provides all the low-level features necessary for accessing hardware, and interaction with existing C code is simplified. Last but not least, many programmers are familiar with C.

C does have significant disadvantages: it provides little help in writing safe code or in structuring applications. NesC addresses safety through reduced expressive power and structure through components.

None of the new features in nesC are tied to C: the same ideas could be added to other imperative programming languages.

Whole-program analysis: nesC programs are subject to whole program analysis (for safety) and optimization (for performance). Therefore we do not consider separate compilation in nesC's design. The limited program size on motes makes this approach tractable.

NesC is a "static language": There is no dynamic memory allocation and the call-graph is fully known at compile-time. These restrictions make whole program analysis and optimization significantly simpler and more accurate. They sound more onerous than they are in practice: nesC's component model and parameterized interfaces eliminate many needs for dynamic memory allocation and dynamic dispatch. We have, so far, implemented one optimization and one analysis: a simple whole-program inliner and a data-race detector.

NesC supports and reflects TinyOS's design: nesC is based on the concept of components, and directly supports TinyOS's event based concurrency model. Additionally, nesC explicitly addresses the issue of concurrent access to shared data. In practice, nesC resolved many ambiguities in the TinyOS concepts of components and concurrency, and TinyOS evolved to the nesC versions.

2.5. TOSSIM [A]:

TOSSIM is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools.

TOSSIM's primary goal is to provide a high fidelity simulation of TinyOS applications. For this reason, it focuses on simulating TinyOS and its execution, rather than simulating the real world. While TOSSIM can be used to understand the causes of behavior observed in the real world, it does not capture all of them, and should not be used for absolute evaluations. TOSSIM is not always the right simulation solution; like any simulation, it makes several assumptions, focusing on making some behaviors accurate while simplifying others. One of the most common questions about TOSSIM is whether it can "simulate X" or whether it "has an accurate X model." In hope of answering most of such questions, here is a brief summary of its characteristics:

• **Fidelity:** By default, TOSSIM captures TinyOS' behavior at a very low level. It simulates the network at the bit level, simulates each individual ADC capture, and every interrupt in the system.

• **Time:** While TOSSIM precisely times interrupts (allowing things like bit-level radio simulation), it does not model execution time. From TOSSIM's perspective, a piece of code runs instantaneously. Time is kept at a 4MHz granularity (the CPU clock rate of the mica platforms). This also means that spin locks or task spin locks will never exit: as the code runs instantaneously, the event that would allow the spin to stop will not occur until the code completes (never).

• **Models:** TOSSIM itself does not model the real world. Instead, it provides abstractions of certain real-world phenomena (such as bit error). With tools outside the simulation itself, users can then manipulate these abstractions to implement whatever models they want to use. By making complex models exterior to the simulation, TOSSIM remains flexible to the needs of many users without trying to establish what is "correct." Additionally, it keeps the simulation simple and efficient.

– **Radio:** TOSSIM does not model radio propagation; instead, it provides a radio abstraction of directed independent bit errors between two nodes. An external program can provide a desired radio model and map it to these bit errors. Having directed bit error rates means that asymmetric links can be easily modeled. Independent bit errors mean longer packets have a higher probability of corruption, and each packet's loss probability is independent.

– **Power/Energy:** TOSSIM does not model power draw or energy consumption. However, it is very simple to add annotations to components that consume power to provide information on when their power states change (e.g., turned on

or off). After a simulation is run, a user can apply a energy or power model to these transitions, calculating overall energy consumption. Because TOSSIM does not model CPU execution time, it cannot easily provide accurate information for calculating CPU energy consumption.

• **Building:** TOSSIM builds directly from TinyOS code. To simulate a protocol or system, you must write a TinyOS implementation of it. On one hand, this is often more difficult than an abstract simulation; on the other, it means you can then take your implementation and run it on actual motes.

• **Imperfections:** Although TOSSIM captures TinyOS behavior at a very low level, it makes several simplifying assumptions. This means that it is very possible that code which runs in a simulation might not run on a real mote. For example, in TOSSIM interrupts are non-preemptive (a result of being a discrete event simulator). On a real mote, an interrupt can fire while other code is running. If pre-emption can put a mote in an unrecoverable state, then simulated motes will run without mishap while real-world motes may fail. Also, if interrupt handlers run too long, a real-world mote may crash; as code in TOSSIM runs instantaneously, no problems will appear in simulation.

• **Networking:** Currently, TOSSIM simulates the 40Kbit RFM mica networking stack, including the MAC, encoding, timing, and synchronous acknowledgements. It does not simulate the mica2 ChipCon CC1000 stack.

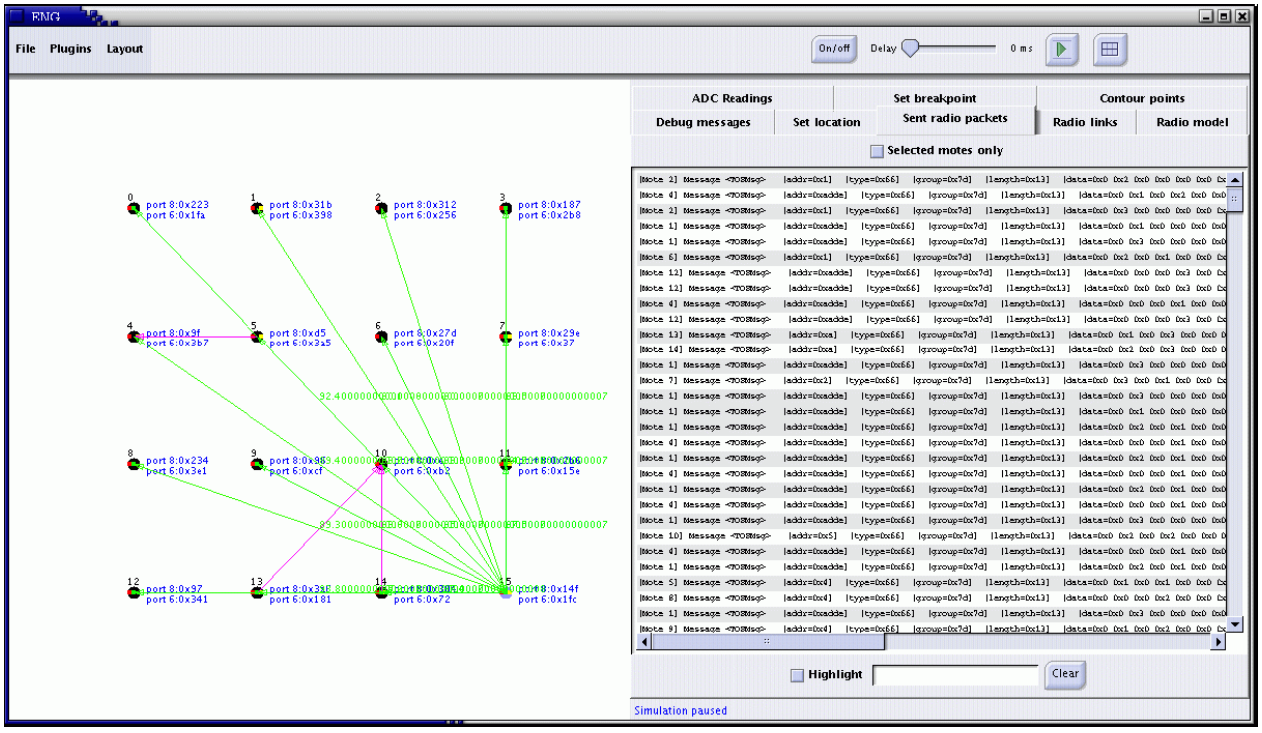
• **Authority:** Initial experience from real-world deployments has shown that TinyOS networks have very complex and highly variable behavior. While TOSSIM is useful to get a sense of how algorithms perform in comparison to one another, TOSSIM results shouldn't be considered authoritative. For example, TOSSIM can tell you that one algorithm behaves better than another under high loss, but the question remains as to whether the specified loss scenario has any basis in the real world. TOSSIM should not be considered an end-point of evaluation; instead, it is a system that allows the user to separate out environmental noise to better understand algorithms.

2.6. Tinyviz [A]:

TinyViz is a Java visualization and actuation environment for TOSSIM. The main TinyViz class is a jar file, tools/java/net/tinyos/sim/tinyviz.jar. TinyViz can be attached to a running simulation. Also, TOSSIM can be made to wait for TinyViz to connect before it starts up, with the -gui flag. This allows users to be sure that

TinyViz captures all of the events in a given simulation. TinyViz is not actually a visualizer; instead, it is a framework in which plugins can provide desired functionality. By itself, TinyViz does little besides draw motes and their LEDs. However, it comes with a few example plugins, such as one that visualizes network traffic.

Figure 2.6 shows a screenshot of the TinyViz tool. The left window contains the simulation visualization, showing 16 motes communicating in an ad-hoc network. The right window is the plug-in window; each plug-in is a tab pane, with configuration controls and data. The second element on the top bar is the Plug-in menu, for activating or de-activating individual plug-in. Inactive plug-in have their tab panes grayed out. The third element is the layout menu, which allows you to arrange motes in specific topologies, as well as save or restore topologies. TinyViz can use physical topologies to generate network topologies by sending messages to TOSSIM that configure network connectivity and the loss rate of individual links.



[Figure 2.6: TinyViz connected to TOSSIM running an object tracking application. The right panel shows sent radio packets, the left panel exhibits radio connectivity for mote 15 and network traffic. The green arrows and corresponding labels represent link probabilities for mote 15, and the magenta arrows indicate packet transmission.]

The right side of the top bar has three buttons and a slider. TinyViz can slow a simulation by introducing delays when it handles events from TOSSIM. The slider configures how long delays are. The On/Off button turns selected motes on and off; this can be used to reboot a network, or dynamically change its members.
The button to the right of the slider starts and stops a simulation; unlike the delays, which are for short, fixed periods, this button can be used to pause a simulation for arbitrary periods. The final button, on the far right, enables and disables a grid in the visualization area. The small text bar on the bottom of the right panel displays whether the simulation is running or paused.
The TinyViz engine uses an event-driven model, which allows easy mapping between TinyOS' event based execution and event-driven GUIs. By itself, the application does very little; drop-in plugins provide user functionality. TinyViz has an event bus, which reads events from a simulation and publishes them to all active plugins.

2.7. TinyDB [A]:

TinyDB GUI is a query processing system to gather, organize, display and analyze from TinyOS sensors. TinyDB provides simple SQL-like interface to specify data you want to extract. TinyDB is a centralized application which only resides on base station/PC, thus it's more like a GUI interface and written in Java.

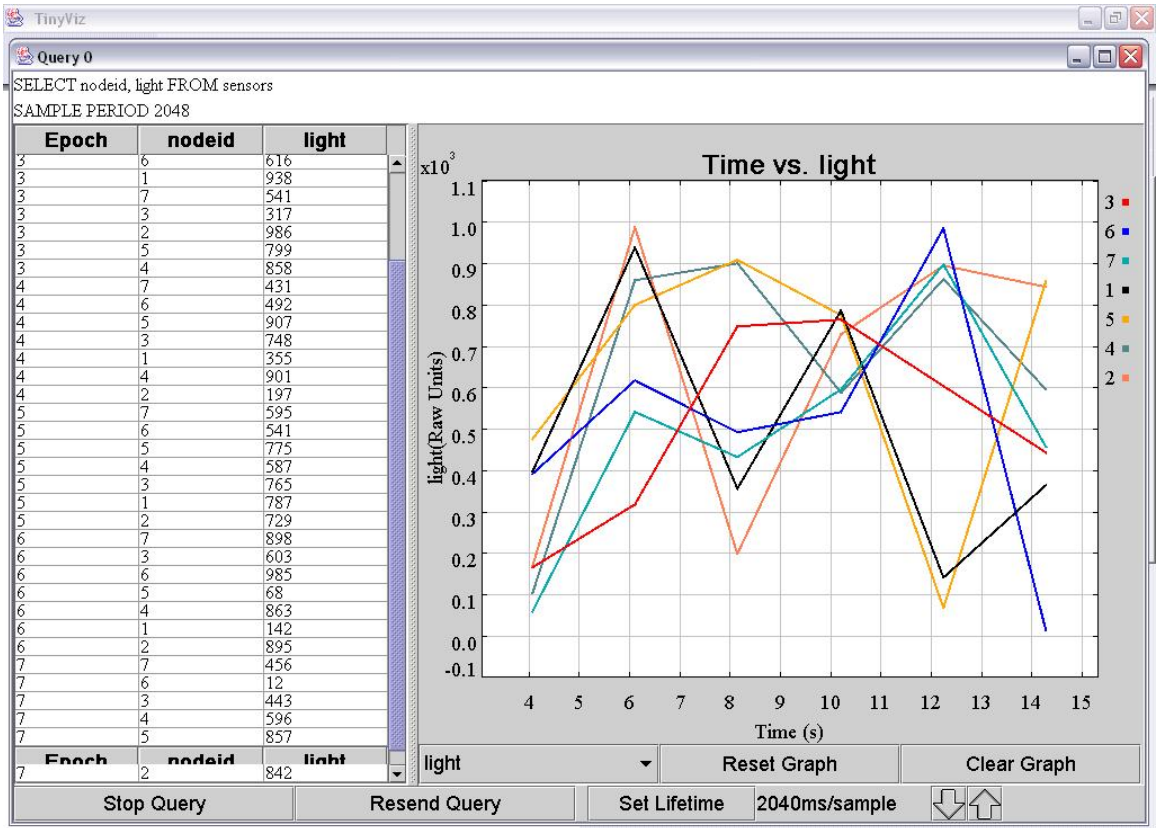


Figure 2.7: sample look of TinyDB interface

TinyDB application is nesC coded and programmed on top of TinyOS system for each node.

3. PROJECT SCOPE AND DESCRIPTION

TinyOS is an open source embedded operating system designed specially for sensor networks. This compact operating system includes memory management, device management (radio), and task scheduling and protocol management for communication layers. Every mote has TinyOS as small as 200K programmed into its ROM and runs the application on top of it. TinyOS implements network routing on application, data link and physical layers and can be ported to different platforms and sensor boards, such as mica2, mica and PC.

The project will modify and enhance routing on application layer. The communication stack is like:

Application->Data Link (MAC) ->Physical

3.1 TinyOS routing algorithm analysis

As the nature of wireless sensor networks, in which the sensor node can only hear and communicate with other peers, called "neighbors", within its signal broadcasting distance, the routing tree algorithm is decentralized. Every node selects or is selected to forward messages between neighbors. In this fashion messages get propagated over the network.

At each node, X:

Initialization

Start timer for TimerTask

Start timer for SendRouteTask

If TimerTask fired

  If neighbor table does not exist

    Create a new one now

  Else update neighbor table

Choose a parent

If SendRouteTask fired

  If neighbor table does not exist

  Return

Sending broadcasting routing messages (index, parent, cost, hop- count)

If Receive routing messages

  Update the neighbor table

If Intercept Snoop messages

  Update neighbor table

Forever

Figure 3.1 The TinyOS Routing Algorithm

- **Message Overhead**

  As it can be seen from the algorithm described in Figure 3.1, the timer for each node goes off after a time period and a broadcast message is sent anyway. Suppose timers for all nodes are not synchronized and there are N nodes. There will be cases in which the child node's timer goes off first and sends out useless messages till their parents notify all the other children. The best case scenario is when the parent always sends routing message ahead of its children resulting in O (N) broadcast messages being sent out to create a neighbor table for each node and another N broadcast message sent out to fill up the neighbor information. In this case it's hard to say the balanced tree structure has less overhead than skewed ones since the messages overhead is determined by the sequence of timers to be fired.

- **Time Complexity**

Every time it creates or updates a neighbor table, it has to scan the entire table once, and every node has to scan its own neighbor table at least twice (create and update). If the neighbor table size is M (upper limit of TinyOS table size is 16), then the total time spent on the routing tree algorithm is O ($2M^2$).

- **Space Complexity**

The neighbor table structure looks like the following:

```
typedef struct TableEntry {
uint16_t id; // Node Address
uint16_t parent;
uint16_t cost;
uint8_t childLiveliness;
uint16_t missed;
uint16_t received;
int16_t lastSeqno;
uint8_t flags;
uint8_t liveliness;
uint8_t hop;
uint8_t receiveEst;
uint8_t sendEst;
} TableEntry
```

Figure 3.2 Neighbor Table

The multihop routing packet structure looks like:

```
typedef struct MultihopMsg {
uint16_t sourceaddr;
uint16_t originaddr;
int16_t seqno;
uint8_t hopcount;
/* offset changes */
uint8_t data [(TOSH_DATA_LENGTH - 7)];
} __attribute__ ((packed)) TOS_MHopMsg;
```

Figure 3.3 Routing Message

3.2 TinyOS: WMEWMA-link quality estimator

TinyOS introduces a new link quality estimator- window means with EWMA(exponentially weighted moving average), also termed as WMEWMA.

WMEWMA(t, a) = (Packets Received in t)/max(Packets Expected in t, Packets Received in t)

Where *t* is the time window represented in number of message opportunities and *a* ∈ [0, 1] controls the history of the estimator. WMEWMA computes an average success rate over a time period and smoothens the average with a EWMA.[9] Although the literature gives out the complex formula, at the application layer, each node snoops on its neighbor messages (messages which are not addressed to it but it can hear/receive them, since all the messages are broadcast into networks) and keeps counting on received and send messages. By observing messages of success and loss, each node has the numerical estimation of its link quality and passes it on to its neighbors.

3.3 Attribute Based Routing for non-constant attributes Implementation

Madden's proposed idea for semantic routing for constant attributes such as location or sensor ID is a simple power-conservation scheme. It's easy to implement with low maintenance costs. Considering the real deployment environment, in most sample periods, sensor readings such as temperature and sound are not likely to widely fluctuate. Instead they mostly deviate within a fixed or constant range. As an example, the situation exists for the light value in the daytime. So, if a user query requests for nodes with attribute data far below range, the sensor nodes not followed in that range ideally would not need to participate in/respond to such a query. Here the proposed scheme implements this idea (let's call it Semantic Routing Tree (SRT)) for non-constant attributes. Current TinyOS application layer routing is primarily based on signal strength. SRT reuses the TinyOS tree structure and appends the historical reading-range

to each node in the routing layer so that query dissemination efficiently routes requests to only appropriate regions. SRT needs periodic maintenance since the sensory attribute's range has to be updated periodically.

3.4 Dynamic Semantic Routing Implementation

Let's now discuss the other contribution proposed in this project. Unlike SRT, which routes the message to most stable nodes, Dynamic Semantic Routing Tree (DSRT) extends the idea of routing primarily based on the historical readings. The purpose of DSRT is to cluster motes area into semantic-close groups so that the query only reaches to affected groups; and maximally save communication and computation cost for non-affected nodes. Intuitively DSRT is not as stable as SRT since it excludes the signal factor.

3.5 Project Task Summary

- TOSSIM modification
  Source file name: adc_model.c
- TinyDB GUI modification
  Source file name: ResultFrame.java
- SRT / DSRT implementation on TinyOS application layer
  Source file names: MultiHopWMEWMA.nc
  MultiHop.h
  RouteControl.nc
- Query dissemination modification in TinyDB application
  Source file names: TupleRouterM.nc
  MultiHopEngineM.nc

4.SYSTEM DESIGN,ARCHITECTURE AND IMPLEMENTATON

4.1 Complete Picture of implementation

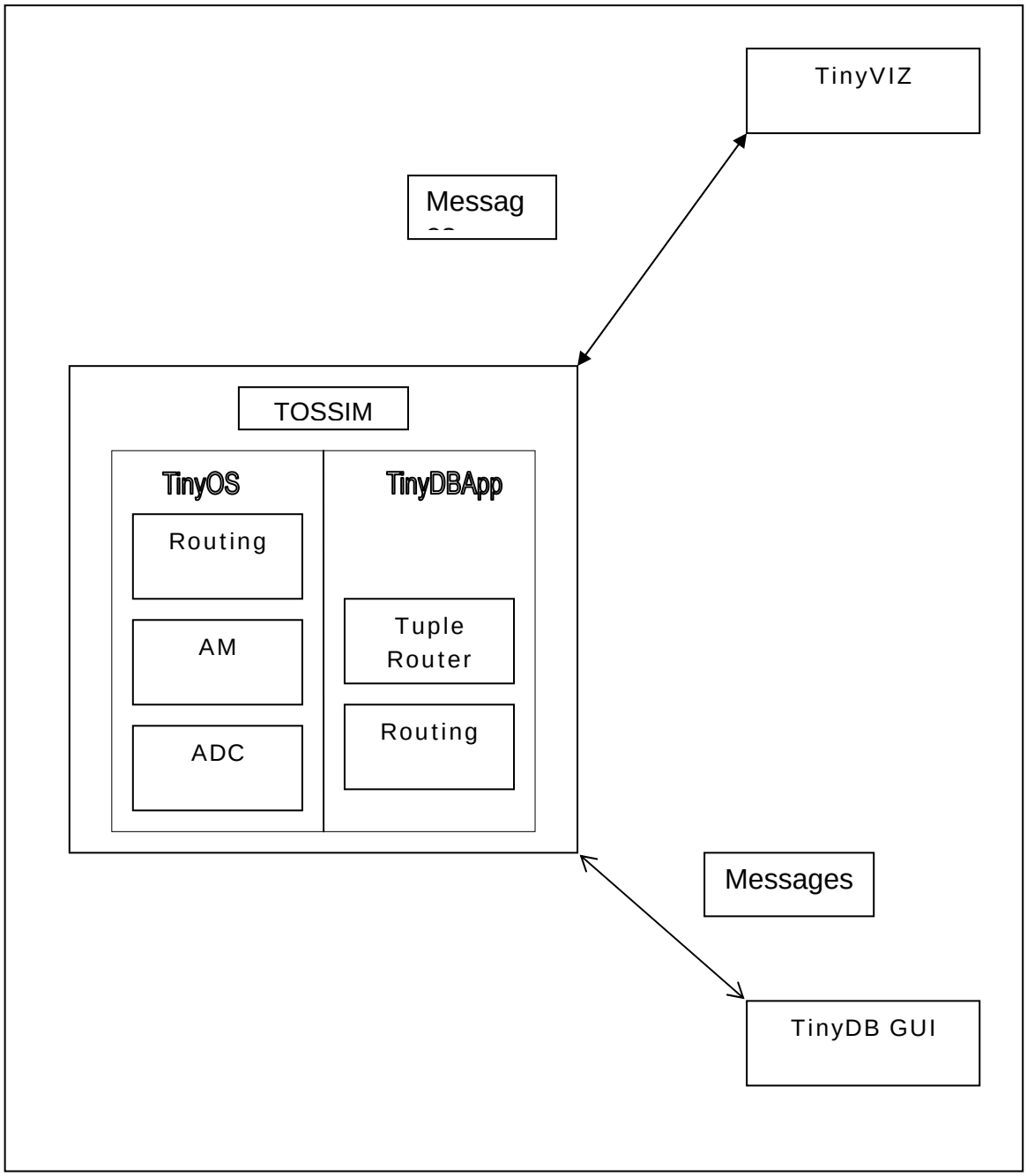


Figure 4.1 System Architecture [A]

Component Descriptions

- **TinyOS Routing:** it's a major module in charge of routings at all communication layers

- **TinyOS AM:** the module managing radio transmission

- **TinyOS ADC:** analog-to-digital reading module.

- **TinyDB Tuple Router:** the module translating incoming query messages.

- **TinyDB Routing:** this module contains query dissemination and application layer routings.

4.2. Project Design

The implementation of DSRT is based on TinyOS/TinyDB code by TinyOS group. The basic process is broken down into three components:

• **History Reading Collector.** History reading collector gathers up all reading information such as temperature, light and sound for a certain period of time and extract the range value (highest and lowest) for each attribute. This range value will be passed on to the routing component for clustering. As for real implementation in the TOSSIM, history reading range is manually put into a configuration file, MoteSet. Every time TOSSIM boots up, it reads the file first and generates random ADC readings within given ranges. TinyOS routing module also reads the range information into memory.

• **New Routing Algorithms.** The original TinyOS routing algorithm (network layer) is an ad-hoc routing based on link quality, the most reliable nodes are chosen to be the parents to relay messages for other peers. Two new routing algorithms (SRT and DSRT) are implemented. The later chapter will elaborate on both algorithms.

• **Query Dissemination based on new routing tree.** In TinyOS/TinyDB, a query request issued from base station gets propagated to all sensor nodes by broadcasting: every node hears the request simply relays query messages to its neighbors. The new query dissemination will selectively relay query requests to only related peers based on the routing tree.

At each node, X:
Initialization
Start timer for TimerTask
Start timer for SendRouteTask
If TimerTask fired
If neighbor table does not exist
Create a new one now
Else update neighbor table
Choose a parent (**update node range**)
If SendRouteTask fired
If neighbor table does not exist
Return
Sending broadcasting routing messages (index, parent, cost, hop- count, **range**)
If Receive routing messages
Update the neighbor table (**include range information**)
If Intercept Snoop messages
Update neighbor table
Forever

Figure 4.2: SRT Algorithm

4.3 SRT Algorithm Analysis

As shown in the diagram above (Figure 4.2), SRT differs with TinyOS Routing in that it adds in the range information into multihop messages. As a result, space complexity is increased by 2 more bytes. Neighbor table size is increased by 2 bytes as well. Other than that SRT has the same message overhead and time complexity as TinyOS Routing.

4.4 Range Information Management

In order to efficiently disseminate query requests, each node dynamically merges its range with its children's ranges. Thus the range of a sensor node overlaps information of sub-tree under it. A special case is BS's range reflects the history range of the whole sensor network, which blocks the unrelated query even before the query is sent out to the network. When tree structure changes, all range information is adjusted - either being removed or merged.

At each node, X:
Initialization
Start timer for TimerTask
Start timer for SendRouteTask
If TimerTask fired
If neighbor table does not exist
Create a new one now
Else update neighbor table
Calculate Manhattan distance for every neighbor
Choose a parent (**update node range**)
Path compression
If SendRouteTask fired
If neighbor table does not exist
Return
Sending broadcasting routing messages (index, parent, cost, hop- count, **range, layer**)
If Receive routing messages
Identify which ripple layer I'm sitting on
Update the neighbor table (**include range information**)
If Intercept Snoop messages
Update neighbor table
Forever

Figure 4.3: DSRT Algorithm

4.5 DSRT Algorithm Complexity Analysis

Same as TinyOS and SRT, it uses two timers to send out routing messages and choose parents. So message overhead and time complexity is same as TinyOS and SRT. However, DSRT (Figure 4.3) puts range and layer information in the routing message structure:

```
typedef struct MultihopMsg {
uint16_t sourceaddr;
uint16_t originaddr;
int16_t seqno;
uint8_t hopcount;
uint8_t rnglow;
uint8_t rnghigh;
uint8_t layer;
/* offset changes */
uint8_t data [(TOSH_DATA_LENGTH - 10)];
} __attribute__ ((packed)) TOS_MHopMsg;
```

Figure 4.4 DSRT Routing Message

So that the total message size is increase by 3 bytes comparing to TinyOS.

4.6 Cycle Prevention Techniques: Hop count and Layer

In the distributed area, each node has the only knowledge of its neighbors instead of the whole picture; it's very easy to produce cycles and stranded nodes inside the tree structure. When a cycle exists in the tree, a message is infinitely passed back and forth in the network, which wastes large amount of communication and should be prevented at all cost.

TinyOS routing algorithm prevents cycles by controlling the *hop count*. A node selects a parent under the condition that its future parent must have the smaller hop count than itself. Thus, the tree is built top down. However there is still slim chance to get into a cycle when two nodes happen to choose the opposite as its parent at the same time.

DSRT introduces a new concept, *layer*, we picture sensor network is made of multiple layers of nodes, starting from the base station, all nodes which can hear BS are the first layer, 0 and nodes sitting on layer 1 can hear layer 0, so on and

so forth. When a query request is issued from BS, it propagates from layer 0 to the farthest layer, which is similar to the ripple effect. Each node has to identify its layer before it comes into parent selection. Layer identification is built top down from BS as well. Once a node knows its layer, it starts parent selection under the rule that it can only choose its parent from its ancestor layers or inside the same layer. And inside the same layer, each node checks if the current parent candidates' have smaller hop counts. Similarly as TinyOS, there is still chance of producing cycles. In the later discussion, we can see a complete cycle-free goal is hard-to-achieve in the distributed environment.

4.7 Layer Acquisition Algorithm

Before building up the routing tree, DSRT assigns layer identities to every sensor node in the network, with the innermost layer (closest to BS) to be the *layer 0*. The top down layer building algorithm is like Figure 4.5:

```
While (true) {
If "I can hear from the Base Station"
then myLayer = 0
Else {
Wait (three timer cycles);
If "I waited for enough time" and "I can hear some one with a layer id I"
Then
myLayer = I +1;
} }
```

Figure 4.5: Layer Acquisition Algorithm

4.8. The Parent Selection Process

In the deployment of sensor networks, sensors can be spread out to cover a large area and the distance from sensors to the base station could be out the reach of sensors' radio signals. For example, the largest hearing distance of two mica motes is 50 feet and radio signal strength decreases as the distance increases. When a sensor node can't communicate with the base station directly, it depends on other sensor nodes closer to the base station to relay the communication. Thus, parent selection enables a message from any given sensor node to eventually reach the base station through one or many hops. The TinyOS

routing implements a multi-hop tree structure. A typical TinyOS routing tree is shown in Figure 4.6.

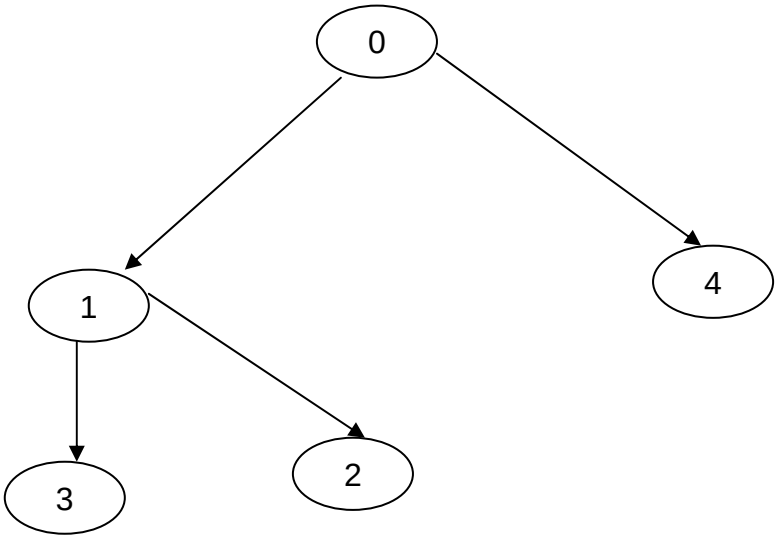


Figure 4.6: A TinyOS Routing Tree

As shown in Figure 4.6, by default, the base station ID is 0. Here let's assume that nodes 2 and 3 are beyond the communication range of the base station and rely on node 1 to relay the communication. We term node 1 to be the parent of nodes 2 and 3. Each node is free to choose a parent on its own. The process of selecting a node from its neighbor is called parent selection. Parent selection is crucial in multi-hop environment and has the direct impact on efficiency of communication. Next two sections will talk about different parent selection criteria of TinyOS, SRT and DSRT.

4.9. Manhattan Distance based parent selection in DSRT:

DSRT selects parents based on range information rather than physical proximity, that is, each node would choose a node that has the most 'similar' range as its parent. For example, suppose a sensor node historically has the temperature between 70F and 90F. Now, let's say it has to choose a parent between two nodes, A and B, each with a range 50-60F and 70-90F. Which one will it select? In this case, B is selected since it has the same range even if A might be closer

to the node in terms of physical distance. What if A and B have the range of 20-30F and 30-40F? In this case, it's hard to evaluate the approximation. Manhattan distance provides the quantitative evaluation of length between two points. Manhattan distance [15], also known as city block distance is defined as:

$$D(i, j) = |xi1-xj1| + |xi2-xj2| + \ldots + |xip - xjp|$$

Where $i = (xi1, xi2, xi3 \ldots xip)$, $j = (xj1, xj2, xj3 \ldots xjp)$ and are two p-dimensional data. Specifically, here the range is defined as a two-dimensional numerical data. For every sensor node attribute two ranges are defined: lowest range and highest range, I = (lowest, highest) and M-distance between two sensor nodes, i and j can be described as

$$D(i, j) = |lowest(i) - lowest(j)| + |highest(i) - highest(j)|$$

Example:
Node A has the range of [10, 20], B has it of [30, 40]
$D(A,B) = |10 - 30| + |20-40| = 40$
Before calculating the distance, a serial of data standardization is performed since standardization gives all variables an equal weight. The process goes like the following:
Given a group of range information, lowest range array LA (LA1, LA2…LAn) and highest as HA (HA1, HA2…HAn)

Calculate the mean value M for LA and HA

$MLA = 1/m * (LA1 + LA2 + \ldots + LAn)$
$MHA = 1/m * (HA1 + HA2 + \ldots + HAn)$

Calculate the standard deviation S for LA and HA

$SLA = sqrt(1/n * ((| LA1 - MLA |2 + \ldots + | LAn - MLA|2))$
$SHA = sqrt(1/n * ((| HA1 - MHA |2 + \ldots + | HAn - MHA |2))$

Calculate the z-score based on S and M

$ZLAi = (LAi - MLA)/ SLA$, $ZHAi = (HAi - MHA)/ SHA$

The M-distance between two nodes 1 and 2 is finally calculate as

$D(1, 2) = |ZLA1 - ZLA2| + |ZHA1 - ZHA2|$

And two nodes are considered as closest when they have *min* (M-distance) of all neighbors.

4.10. DSRT Parent Selection Process

DSRT defers parent selections between 0 and other layers. A sensor node in layer 0 tries to directly connect to BS except its range falls into others. The purpose is as the layer 0 is closest to the BS; it should avoid multi-hops as possible. Starting from layer 1, parent selection is determined by M-distance.

4.11. Path Compression

DSRT tries to cluster sensor nodes and it allows one or more hops inside one layer. Therefore, DSRT routing tree has more depth than TinyOS. It's a tradeoff between increasing tree depth and segmenting clusters. For example, a cluster of two or three nodes and a huge hop count doesn't save much communication and processing cost for conditional queries, instead, it scarifies a lot of network performance. This case should be avoided and here is the path compression coming into play. Path compression in the graph theory refers to that in order to eliminate long paths, the child saves the pointers to its ancestor parents (e.g. Parent's parent) and skip the current parent to compress the path. DSRT introduces a constant, HOP_LIMIT to control the length of path. If the current hop counts exceed the HOP_LIMIT, DSRT does path compression.

4.12. Code Optimization

Embedded system programming requires optimal code to save computation power. The following code optimization schemes are applied in SRT and DSRT coding.

keeping *struct* size to be power of two

Minimize the local variables and declare them in inner scope

Prefer *int* over char

Minimize the function arguments

5. TESTING RESULTS AND ANALYSIS

5.1. Test Environment Setup

All tests are executed in TOSSIM and TinyDB GUI. Three types of radio model models- single hop, lossy model and empirical model are simulated.

In TOSSIM, a network signal is either a one or zero. All signals are of equal strength, and collision is modeled as a logical or; there is no cancellation. The single hop model assumes the perfect transmission of radio single. Each node can hear from its neighbors without any data loss.

The "lossy" radio model places the nodes in a directed graph. Each edge (a, b) in the graph means a's signal can be heard by b. Every edge has a value in the range (0, 1), representing the probability a bit sent by a will be corrupted (flipped) when b hears it. For example, a value of 0.01 means each bit transmitted has a 1% chance of being flipped, while 1.0 means every bit will be flipped and 0.0 means bits will be transmitted without error. Each bit is considered independently.[A]

Empirical model assumes each mote has a transmission radius of 50 feet. Combined with the bit error rate, this means each mote transmits its signal in a disc of radius 50 feet, with the bit error rate increasing with distance from the center. [A]

5.2 Evaluation Metrics

There are several network metrics to evaluate the performance of sensor networks.

- **Messages to build up the tree(BTM):** routing messages it takes to built up a consistent tree and it happens when sensor networks boot up
- **Query messages disseminated to the whole network (QM):** number of query messages are sent to the network so that all sensor nodes get the query request
- **Round trip time for a certain query (RTT):** interval in seconds from issuing the query to data coming back and displayed.
- **First response time (FRTT):** the round trip time of the first responded node.
- **Last response time (LRTT):** the round trip time of the last responded node.
- **Response percentage (RP):** number of nodes responded with data results/ number of nodes expected to return results
- **Participation Rate (PR):** number of nodes involved into queries
- **Network Topology (NT):** there are three NT being tested in this project.

  Single Hop: each node can reach BS with no data loss

  Multihop: Complement of single hop. Nodes are spread out as grid with equal distance. Data is lossy and erroneous. The TinyOS empirical model (refer to manual) is used to generate test data.

  Skewed (straight line): sensor nodes form a straight line.

5.3 Simple Comparison & Analysis

Let's see an example of 5 sensor nodes, labeled as 1, 2, 3, 4, 5 in single hop. And namely each has the light value (lumens) as Figure 5.1.

NodeID	Light Ranges
1	[20,60]
2	[10,100]
3	[20,60]
4	[10,100]
5	[10,60]

Figure 5.1 light ranges

TinyOS and SRT routing structures turn out like this (Figure 5.2):

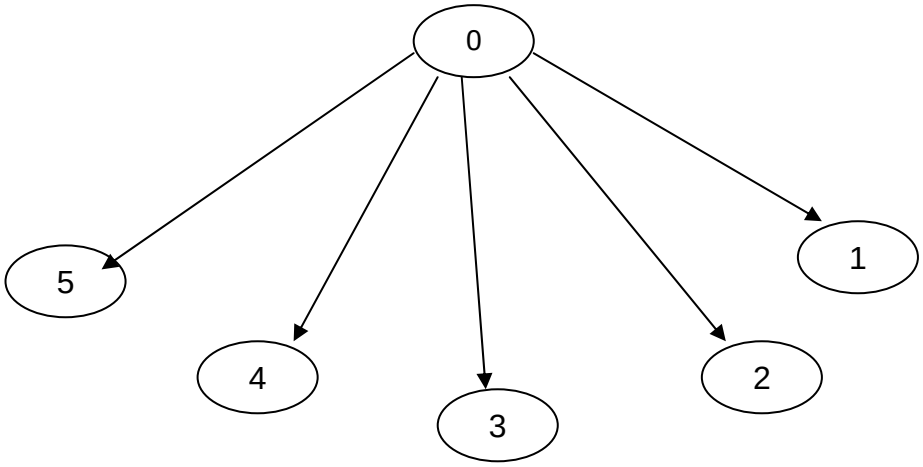


Figure 5.2 TinyOS / SRT Routing tree

Here TinyOS and SRT structures are the same (Figure 5.2). That's because SRT doesn't not change the way routing tree is built up and it only adds range information on each node. And DSRT structure is a bit different.

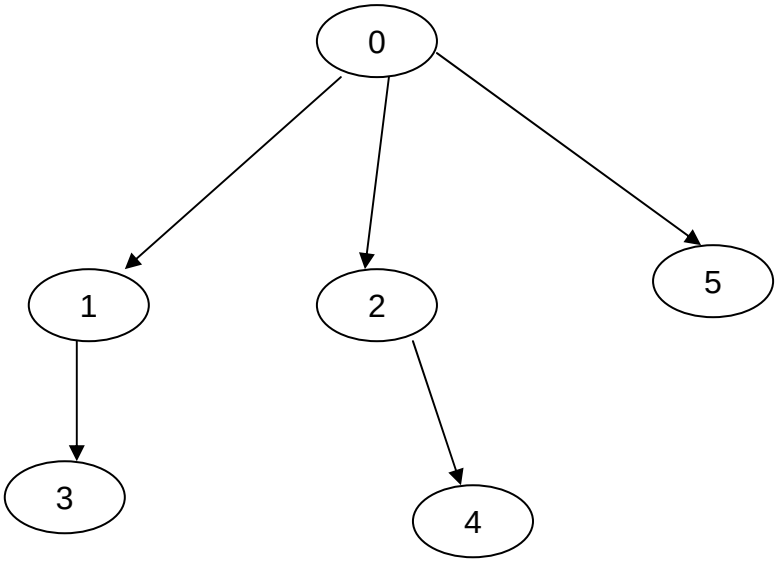


Figure 5.3 DSRT Routing tree

As shown in Figure 5.3, DSRT tries to arrange nodes with similar light range together. Here since nodes 1 and 3 are within the same range, node 3 becomes the parent of node 1. It's worth mentioning that nodes 1 and 3 is interchangeable, that is, it's possible node 1 becomes the parent of 3 and directly connects to BS. It all depends on whose timer is fired first and selects the BS as parent.

Supposing there is a query, format as

Select nodeid, light

This extracts all the light readings from all sensors. In this case, BS sends out the query request and every node works on its own, sending back the data. DSRT has more latency since its largest hop count is 2. However in the case of conditional query, format as

Select nodeid, light where light> 60

Node 1 will not participate in the query process. What it saves? Communication cost between node 1 and 3, processing cost of node 1. Looks like not save much. However, in large scale networks, communication and processing cost of ten or more nodes means a lot.

5.4. Test Results

- Test case scenario 1:

  Single Hop, 10 nodes, light ranges are within [50, 100].

  For following Non-conditional query,

  Select nodeid, light from sensors

	BTM	RP	FRTT / LRTT	NT
TinyOS/SRT	11	100%	15~16/16~17	single hop
DSRT	3	100%	16~19/16~20	single hop

Figure 5.4 Test Case 1

There is not much performance discrepancy between three, only that TinyOS/SRT cost more BTM, since TinyOS/SRT need sending more routing messages to establish WMEWMA metrics.

- Test case scenario 2:

  Single Hop, 30 nodes, light ranges are within [10, 30], [40, 70], [80, 120].

  For following Non-conditional query,

  Select nodeid, light from sensors

	BTM	RP	FRTT / LRTT	NT
TinyOS/SRT	27	86%	56/157	single hop
DSRT	4	76%~83%	55~57/76	single hop

Figure 5.5 Test Case 2

DSRT has poorer RP and RTT. It can be explained by the DSRT tree structure tends to have bigger tree depth and enlarge the network latency.

- Test case scenario 3:

  Single Hop, 30 nodes, light ranges are within [10, 30], [40, 70], [80, 120], conditional query:
  Select nodeid, light where light in T

	QR		
	T= [10,30]	**T= [40,70]**	**T=[80,120]**
TinyOS	22	22	22
SRT	20	22	19
DSRT	14	11	11

Figure 5.6 Test Case 3(QR)

	PR		
	T= [10,30]	**T= [40,70]**	**T=[80,120]**
TinyOS	29	29	29
SRT	19	27	26
DSRT	16	14	16

Figure 5.7 Test Case 3(PR)

TinyOS routing does not filter any query messages and it consumes more messages and processing power than SRT and DSRT. DSRT outperforms a little than SRT in terms of QM and PR.

- Test case scenario 4:

Multi-Hop empirical model, 4x4 grid with equal distance of 10 feet, 16 nodes, light ranges are within [10, 60], [60,110], [120, 170].

For following non-conditional query:

Select nodeid, light from sensors

	BTM	RP	FRTT/LRTT	NT
TinyOS/SRT	26	100%	33 / 96	Multi hop
DSRT	7	53-60%	55-57 / 95~519	Multi hop

Figure 5.8 Test Case 4

- Test case scenario 5:

  Multi-Hop empirical model, 4x4 grid with equal distance of 10 feet, 16 nodes, light ranges are within [10, 60], [60,110], [120, 170].
  For following conditional query:

  Select nodeid, light where light in T

	QR		
	T= [10,60]	**T= [60,110]**	**T=[120,170]**
TinyOS	11	11	11
SRT	16	17	18
DSRT	4	8	5

Figure 5.9 Test Case 5(QR)

	PR		
	T= [10,60]	**T= [60,110]**	**T=[120,170]**
TinyOS	15	15	15
SRT	4	12	14
DSRT	5	10	11

Figure 5.10Test Case 5(PR)

- Test case scenario 6:

  Multi-Hop lossy model, 25 nodes, light ranges are within [10, 50], [60,100], [125, 190], and [200, 250].

  For following Non conditional query:

  Select nodeid, light From sensors

	BTM	RP	FRTT/LRTT	NT
TinyOS/SRT	12	100%	34 / 175	Multi hop
DSRT	7	100%	109 / 320	Multi hop

Figure 5.11Test Case 6

- Test case scenario 7:

  Multi-Hop lossy model, 25 nodes, light ranges are within [10, 50], [60,100], [125, 190], and [200, 250].
  For following conditional query:

  Select nodeid, light where light in T

	QR			
	T= [10,50]	**T= [60,100]**	**T=[125,190]**	**T=[200,250]**
TinyOS	5	5	5	5
SRT	5	5	5	5
DSRT	4	5	4	5

Figure 5.12 Test Case 7(QR)

	PR			
	T= [10,50]	**T= [60,100]**	**T=[125,190]**	**T=[200,250]**
TinyOS	24	24	24	24
SRT	8	8	8	8
DSRT	9	9	8	8

Figure 5.13Test Case 7(PR)

5.5. Result Analysis

TinyOS routing does not filter any query messages and it consumes more messages and processing power than SRT and DSRT.

In the single hop environment, DSRT outperforms SRT and TinyOS in that it takes fewer messages overhead to establish the routing and disseminate the messages.

While in the lossy environment, DSRT has poor network response time. (As the value increases for FRTT/LRTT)

6. FUTURE WORK

DSRT builds up routing structure based on single non-constant attributes and supports conditional query of single predicate. Same as SRT, It keeps track of range information for single attribute. However, user queries can be multi-dimensional, for example query for temperature and sound at the same time. In the case, there will be multiple routing trees, one for each attribute. How to disseminate dimensional queries will involve finding common optimal paths between multiple tree structures. As for space, with more and more attributes adding in, efficient storage of range information has to be taken into account. So future work will include attribute based routing implementing over multi attribute based queries and multi query support for this proposed routing scheme.

REFERENCES

Collection of literature

Papers

1. S. Madden. The design and evaluation of a query processing architecture for sensor networks ph.d. thesis. UC berkeley.

2. R. Govindam. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCOM ’00), 2002.

3. J. L. Feng Zhao. Information-directed routing in ad hoc sensor networks. In International Conference on Mobile Computing and Networking, 2003.

4. D. D.Estrin. Rumor routing algorithm for sensor netowrks. In Proceedings of the First Workshop on Sensor Networks and Applications(WSNA), 2002.

5. Yao Y., Gehrke J.: Query Processing for Sensor Networks. Proceedings of the CIDR Conference, 2003.

6. Trigoni N., Yao Y., Demers A., Gehrke J.: Multi-query Optimization for Sensor Networks. Technical Report TR2005-1989, Cornell University, 2005.
7. Yao Yong, Johannes Gehrke, “Query Processing In Sensor Networks", CIDR Conference,2003.
8. Geoffrey Werner-Allen, Swieskowski Patrick and Welsh Matt , "MoteLab: A Wireless Sensor Network Testbed” ,IEEE Conference, 2005.

9. Chong Chee-Yee, Srikanta P. Kumar, "Sensor Networks: Evolution, Opportunities, and Challenges",IEEE Volume 91,No. 8 , August 2003.
10. Gummadi Ramkrishna , Li Xin, Govindan Ramesh , Hong Wei, "Energy-Efficient Data Organization and Query Processing in Sensor Networks" 21st international Conference on Data Engineering (ICDE 2005), IEEE
11. Bharathidasan Archana, Sai Ponduru Vijay Anand "Sensor Network: An Overview".

Websites

A. TinyOS TOSSIM http://www.tinyos.net/
B. Sensor Nodes http://www.xbow.com/
C. TinyDB: http://www.telegraph.cs.berkeley.edu/**tinydb**/
D. NesC: http://www.nesc.sourceforge.net/

Books

"Wireless Sensor Networks: An Information Processing Approach".
By Feng Zhao and Leonidas Guibas.