UNIVERSITY OF CINCINNATI

Date: February 07, 2008

I, Premkumar Krishnan **,**

hereby submit this work as part of the requirements for the degree of:

Master of Science

in:

Computer Engineering

It is entitled:

"Performing Data Aggregation on Wireless Sensor Motes"

This work and its defense approved by:

Chair: Dr. Dharma P. Agrawal

Dr. Hal Carter

Dr. Wen-Ben Jone

Performing Data Aggregation on Wireless Sensor Motes

A thesis submitted to the

Division of Research and Advanced Studies of the University of Cincinnati

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In the Department of Electrical and Computer Engineering

Of the College of Engineering

February 07, 2008

by

Premkumar Krishnan

B. Tech. (Computer Science & Engineering)

Guru Gobind Singh Indraprastha University

Delhi, India, 2004

Thesis Advisor and Committee Chair: Dr. Dharma P. Agrawal

Abstract

A Wireless Sensor Network (WSN) consists of a large number of sensor nodes dispersed over a target area for monitoring [1] and these sensing devices are interconnected in the form of an ad-hoc network using wireless radios. A sensor node senses a physical parameter and transmits the same in a multi-hop fashion towards a single collector node known as the sink or Base Station (BS). At the sink, collected data from all the sensors is analyzed and appropriate action is taken. Again, most of the physical parameters obtained by the environmental sensors exhibit a property of spatial-correlation, i.e., smooth variation over space. This property of sensors can be exploited to remove redundant data sensed by neighboring sensors. This can be done by employing a fast and robust data aggregation protocol at intermediate sensors that combines its own sensed data with the data reported from its neighbors, before transmitting the final compressed packet to the BS. In this thesis, a tree-based aggregation scheme for WSNs, TREG (Tree-based Regression) [2] is implemented on wireless sensor motes and its effectiveness is evaluated. The experimental results obtained in the TinyOS [3] programming environment validate the results obtained previously from the software simulation. TREG builds a distributed spanning (aggregation) tree where each tree node performs multivariate polynomial regression and instead of raw data, transmits only the coefficients of a polynomial *P,* reducing the volume of data considerably. Since, computation consumes negligible energy as compared to data transmission [4], a primary objective of WSN programming is to conserve energy at each sensor through additional computation that could result in reduced radio communication. In TREG, some of the sensors only sense data and report them to their nearest tree nodes. Each of the tree nodes then implements an aggregation scheme in a distributed manner, thereby reducing the overall communication overhead as compared to a strictly centralized scheme. The sensor motes [5]

which are tree nodes, have been programmed to perform their own polynomial regression in nesC [6], a component-oriented variant of the C language, used by the TinyOS operating system. One of the main challenges in implementing a high level scheme in low level programs is the extremely resource constrained hardware of motes. The current generation of sensor mote typically has an 8-bit micro-controller, few KB of Program ROM and Data RAM respectively. Thus, any scheme must not overstretch the limited processing and memory storage of the mote hardware. Some of the main challenges encountered during the course of the implementation are detailed in this thesis.

To my dearest parents,

Mr. K.S. Krishnan

and

Mrs. Radha Krishnan

Acknowledgements

I am extremely grateful to my advisor Dr. Dharma P. Agrawal for his valuable guidance, constant motivation and support throughout my pursuit of the Masters Degree at the University of Cincinnati. I would also like to thank Dr. Hal Carter and Dr. Wen-Ben Jone for reviewing my thesis and being part of my thesis defense committee.

I would also like to thank my fellow researchers at the Center for Distributed and Mobile Computing (CDMC) for their support and helpful suggestions. I especially thank Torsha Banerjee for the help and guidance extended by her throughout my thesis research work.

Finally, I express my deepest gratitude to my parents and sister for their love and support, trust and encouragement. I would also like to thank all my friends, both at the University of Cincinnati and outside, for their help and motivation.

Table of Contents

List of Figures and Tables

Chapter 1

Introduction

1.1 Wireless Sensor Networks

A Wireless Sensor Network (WSN) consists of a large number of sensor nodes dispersed over a target area for monitoring some specific physical parameters. A sensor node senses a physical parameter and transmits the same in a multi-hop fashion towards a single collector node in the network known as the sink or Base Station (BS). At the sink, collected data from all the sensors is analyzed and appropriate action is taken. Each sensor node consists of a sensor-board having multiple transducers for sensing various physical and environmental attributes. It also has a micro-controller to perform simple computations, on board memory (RAM and ROM), a radio transceiver to receive and transmit data and a battery to power all the functional units. Figure 1.1 shows a typical wireless sensor network. WSNs are "data centric" in that data is requested from network nodes based on certain specific attributes such as "Give maximum acoustic data in the target area ($50 \leq x \leq 90$, $50 \leq y \leq 90$) every 15 seconds ".

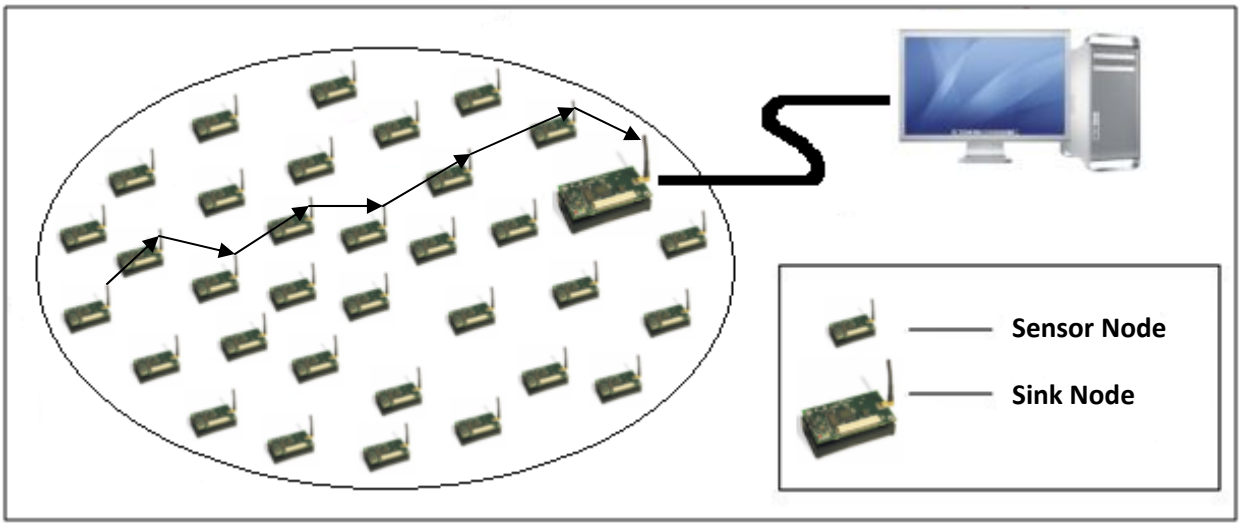

Figure 1.1: A Typical Wireless Sensor Network

1.2 Applications

WSNs are useful in areas ranging from environmental, military to scientific research and as such can have innumerable applications. Some of the most important applications of WSNs include:

1. Battlefield surveillance and tracking of enemy activities are the most common military applications of sensor networks, cited in the literature. Sensor nodes are likely to be deployed by a low-flying aircraft or an unmanned ground or aerial vehicle inside the enemy's territory to remotely monitor activities on the ground [1].
2. Environmental applications may include the use of sensors to monitor the land fill and air quality [7]. Solid waste from household and non hazardous industrial waste such as sewer sludge are typically dumped in landfills where the organic constituents in them undergo degradation by fermentation and oxidation-reduction. These result in the release of harmful gases such as methane, carbon dioxide, nitrogen, sulfide compounds and ammonia that have an adverse impact on the environment as well as living organisms. The traditional method of landfill monitoring involves drilling a set of collection wells across the landfill and collection of gas samples in airtight bags which are analyzed off-site, a time consuming process indeed. With WSNs, a large number of sensor nodes can be placed all over the landfill, each of which might house different types of sensors depending on the type of pollutant anticipated in a given area. Then, in a cost effective manner and in real-time, the entire area of interest can be monitored continuously for the emission of harmful gases from the degrading waste materials.

   The Environment Observation and Forecasting System (EOFS) is a distributed system spanning large geographical areas and used to monitor, model and forecast various physical

processes like flooding, environmental pollution and so on. It usually consists of sensor stations, a distribution network and a central area for processing a large volume of data generated in such a system. The CORIE [8] system for the Columbia River (Oregon, USA) is a prototype of the EOFS, comprising of a real-time sensor network, a data management system and advanced numerical models. About thirteen stationary sensor nodes fixed to a pier are deployed across the Columbia River estuary while one mobile node drifts off-shore. Sensor data are transmitted wirelessly toward on-shore stations and from there to a centralized server where a computationally intensive physical environment model analyzes the received data to guide vessel transportation.

The Automated Local Evaluation in Real-Time (ALERT) [9] is a WSN developed by the National Weather Service in the 1970s and deployed across most of the western United States to provide real-time rainfall and water level information. The sensor nodes in the ALERT system are equipped with water level sensors, temperature sensors and wind sensors. Data is transmitted using line-of-sight communication to a BS where a flood forecasting model processes the data and issues an automatic warning, if necessary.

3. WSNs can be extremely useful in detecting a major disaster such as earthquake, volcanic eruption, fire and terrorist attack. For instance, volcanic activity prediction has not been perfected to date. However, significant progress has been made in recent decades in their prediction. Typically, volcanoes (even dormant ones) are monitored continuously (a very expensive process) for gas emissions (SO_2) and thermal emissivity changes at the volcano's surface. WSNs can bring down the costs of monitoring significantly yet at the same time provide accurate data to enable effective study of volcanic activity. For example, using WSNs, it would be possible to obtain the temperature data at various locations of the

volcano's surface and build heat flux maps based on the temperature gradients. The problem of sensor deployment would also be eased considerably since wireless sensors can be randomly dispersed over the target area and later re-aligned if necessary.

4. The PODS [10] research project at the University of Hawaii has developed a WSN to investigate the reason behind the growing of endangered plant species only in certain parts of the wilderness. Sensor Nodes (SN) or Pods deployed in the Hawaii Volcanoes National Park, consist of a computer, radio transceiver, environmental sensors and occasionally a high resolution digital camera. The sensor data: (1) Weather data collected every ten minutes and (2) Image data collected once per hour, are relayed wirelessly to the Internet from where users can access the data remotely.

5. Habitat monitoring represents yet another important class of WSN applications. WSNs let life sciences researchers to monitor plants and animals in their natural habitat in a non-invasive manner. Severe human disturbance may, at the very least distort results by changing behavioral patterns or distributions, if not seriously reduce or even destroy sensitive populations by increasing stress, reducing breeding success, increasing predation, or causing a shift to unsuitable habitats [11]. A group of researchers at the University of California at Berkeley in collaboration with the Intel Research Laboratory developed and deployed a hierarchical WSN consisting of 32 Sensor Nodes (SNs) at Great Duck Island, Maine, in order to study the nesting habits of Leach's Storm Petrel. The SNs have been placed both in and around the burrows of the nesting Leach's Storm Petrel. The nodes inside the burrows are used to detect the presence / absence of petrel using infrared radiation, while the ones outside are used to measure general environmental data such as temperature, humidity, ambient light and so on. All these data are used in conjunction to

develop models to understand the preference for a particular nest site over others by the Storm Petrel.

All of the aforementioned WSN applications suggest that large amounts of data get transferred through the sensor network towards the sink. In the next section, we shall look at some of the important characteristics of WSNs.

1.3 Characteristics of Wireless Sensor Networks

Traditionally, large collections of sensors connected by wires to a central monitoring system have been used extensively to measure and analyze physical data remotely. Advances in the design of low cost, low power hardware, coupled with availability of small sized batteries have led to the development of sensor devices with wireless communication capabilities called wireless sensor motes [12]. Wireless sensors have significant advantages as compared to their wired counterparts in that they can easily deployed (in many cases just dropped off a low flying unmanned aircraft) and are easier to maintain.

Each wireless sensor node is powered by a set of batteries that have limited lifetimes. Depending on the application, however, rechargeable ones such as solar cells, may be used as well. But the vast majority of WSN applications would not have guaranteed exposure to the sun light. Therefore, SNs have to make use of the limited energy judiciously so as to maximize their lifetime, since in most cases, it would be extremely difficult to replace the batteries physically. Thus, besides the use of low power hardware, energy saving schemes must be an integral part of the WSN system architecture. We elaborate more on this aspect in the next chapter.

Another interesting aspect of WSN applications involving the sensing of environmental parameters such as temperature, pressure and humidity is the nature of the sensed data itself. Most environmental data have been found to vary gradually and continuously over space and

time. This phenomenon is referred to as the spatial-temporal correlation of the sensed data. Specifically, any WSN would have to have a high SN density in order to obtain good coverage of the area being sensed. This implies that sensors are likely to be relatively close to each other, thereby reporting data values with very high degree of correlation to the sink. This is called spatial correlation. The degree of correlation between reported values also depends on the proximity of the sensors to each other.

In a WSN, successive environmental data values reported by the same sensor also tend to exhibit a high degree of correlation. This property of the data values is referred to as Temporal Correlation, meaning correlation over time and is observed mostly with environmental data like temperature, humidity etc. The difference between the maximum and minimum temperatures on a typical day is fairly small and as such, the temperature varies between these two extremes gradually over a 24 hour period. Hence, each successive temperature value reported by an individual sensor is likely to be the same for a number of rounds of reporting, before any noticeable difference in values is observed.

Spatial-Temporal correlation in the sensed data values can be exploited in order to develop WSN solutions that not only reduce the volume of network traffic but also result in better power efficiency.

1.4 Overview of the thesis

In this thesis we have implemented TREG [2], a tree-based aggregation scheme, on wireless sensor motes and evaluated its effectiveness using experimental results. TREG builds a distributed spanning (aggregation) tree where each tree node performs multivariate polynomial regression and instead of raw data, transmits only the coefficients of a polynomial *P,* reducing

the volume of data considerably. We have implemented TREG in TinyOS 2.0 [3]. Since, computation requires much lower energy consumption as compared to wireless transmission [4], a primary objective of WSN programming is to save energy of each sensor through increased computation at the expense of reduced radio communication. In TREG, some of the sensors only sense data and report them to their nearest tree nodes. Each of the tree nodes then performs the aggregation scheme in a distributed manner, thereby reducing the overall communication overhead as compared to a strictly centralized scheme. We have programmed the sensor motes [5] which are tree nodes to perform their own polynomial regression in nesC [6], a component-oriented variant of the C language, used by the TinyOS operating system. One of the main challenges in implementing such high level schemes in low level programs is the extremely resource constrained hardware of motes. The current generation of sensor mote typically has an 8-bit micro-controller, few KB of Program ROM and Data RAM respectively. Thus, any scheme must not overstretch the limited resources of the mote hardware. Some of the main challenges encountered during the course of our implementation are discussed later in this thesis.

1.5 Thesis Organization

This thesis is organized in the following manner. In chapter 2, we discuss the need for Data aggregation in WSNs and elaborate on different kinds of possible aggregations. We also discuss some existing work in the implementation of such schemes on sensor motes. Chapter 3 explains the TREG scheme of data aggregation in detail. In Chapter 4, the hardware and software environment used for the implementation of TREG are discussed. In Chapter 5, we explain the implementation of TREG on the MICAz motes and discuss the challenges faced during the course of the implementation. In Chapter 6, we present the results that we have obtained from the

experiments performed on the sensor motes. In Chapter 7, we conclude the thesis and discuss some of the work that we would like to take up in the future.

Chapter 2

Data Aggregation in Wireless Sensor Networks

Application of WSNs is being explored in a wide range of sectors. Amongst their most common applications is the sensing of some environmental parameter such as temperature, pressure, humidity and so on. Typically in such applications, a large number of SNs are strewn over the target area, resulting in the SNs being in close proximity to each other. In such deployments, it is observed that neighboring SNs report data values with a high degree of correlation. For instance, in the case of temperature, most of the time, SNs close to each other tend to report identical temperature readings. This is referred to as the spatial correlation of data. The other kind of correlation that is observed in sensed environmental data is the temporal correlation of data where the successive data values are found to be identical. With respect to WSNs, this implies that successive data values reported by the same SN are the same most of the time. This is mostly observed in environmental monitoring applications where the sensed parameter varies slowly and by a small amount between its minimum and maximum values.

The spatial and temporal correlations of the WSN data can be exploited favorably for the development of efficient communication protocols for the WSN paradigm. Sensor data correlations can be used effectively so as to avoid redundant data transmissions. This will also lead to the reduction of overall energy consumption in WSN. In-Network Processing or Data Aggregation schemes are the most popular ways of using the correlations in sensor data in order to develop schemes for WSNs that have significant advantages over traditional ways of data transfer in WSNs.

2.1 Network Routing Models

In general WSN routing protocols fall into two main categories [13]:

a) Address-centric Protocol (AC)

b) Data-centric Protocol (DC)

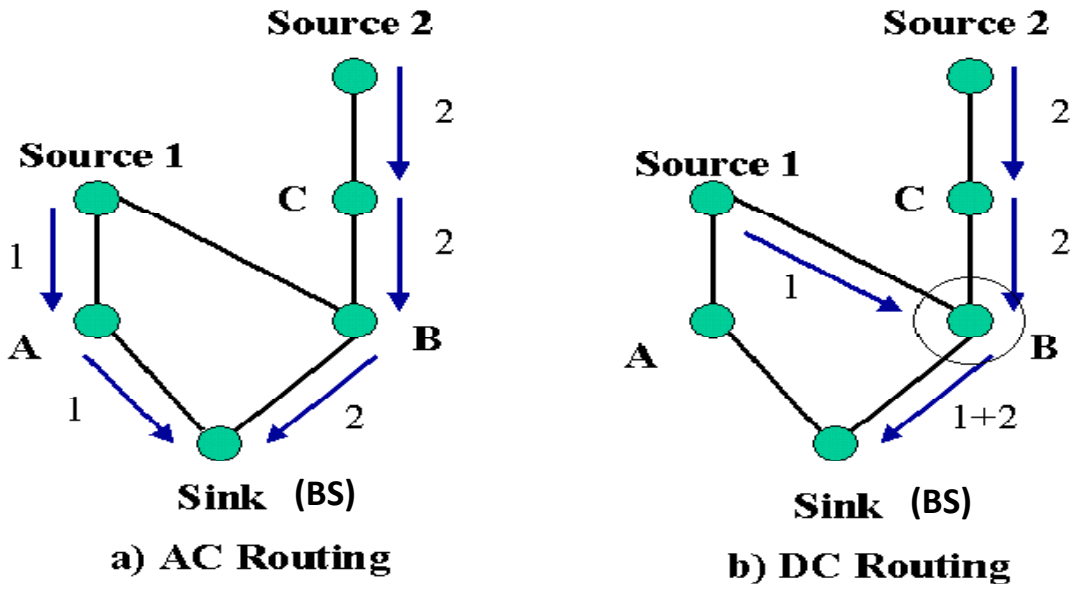

Figure 2.1: AC versus DC routing [13]

2.1.1 Address-centric (AC) Protocols

In AC protocols [13], the sink initially sends out a query through the network requesting some data. The SNs that have the desired data, respond by sending packets containing the same towards the BS. Each source sends its data independent of the other sources along the shortest path. This is also referred to as end-to-end routing. A sample scenario in a typical AC protocol is

illustrated in Figure 2.1 (a). As can be seen from the figure, both Sources 1 and 2 have data to be sent to the sink. However, each source sends the data independent of the other, resulting in two separate routing paths taken by the respective packets. The packet from Source 1 goes through node A while that from Source 2 goes through nodes C and B before reaching the sink.

2.1.2 Data-centric (DC) Protocols

In DC protocols [13], as in the case with the AC approach, the sink sends a query requesting some data initially and SNs having the same, respond with reply packets that are sent towards the sink. However, instead of simply forwarding the packet, the intermediate nodes along the path towards the sink access its contents and consolidate the data received from multiple sources.

Figure 2.1.2 (b) illustrates the DC approach to routing packets in WSNs. The packets from Sources 1 and 2 are aggregated at node B and a consolidated packet is sent to the sink. This aggregated packet consists of data resulting from the manipulation of the original data received from the two sources. It is evident from the figure that using the DC approach results in fewer packet transmissions than the AC approach, thereby resulting in significant energy savings in the SNs.

2.2 Data Aggregation

Data aggregation is a form of in-network processing that reduces the amount of data transferred over a WSN by increasing the amount of processing / computation performed in the network by the SNs. Simple functions such as MAX, MIN, AVG and so on are computed by the intermediate nodes based on the data received. Thus, at each intermediate node, the amount of outgoing data is considerably lower than the amount inputted. Several studies have concluded

that the energy consumed in doing computation is significantly lower than that consumed in wireless transmission. In [14], the authors state that transmitting 1 Kb of data to a distance of 100 m consumes the same amount of energy as executing 300 million instructions on a general purpose processor with a modest computing device rate of 100 million instructions per second (MIPS). Thus, although the computational overhead on the SN itself increases dramatically as a result of data aggregation, the amount of data transmitted decreases considerably and consequently substantial savings in the energy consumption by the SNs is observed.

2.3 The need for Data Aggregation

Typically, WSNs are used to remotely monitor a target area that's otherwise not easily accessible. In such situations, deployment necessitates randomly dropping sensors from low flying un-manned aerial vehicles. Thus, in order to cover the entire area uniformly, a large number of SNs are deployed. This implies that the SNs are in close proximity to each other. In environmental monitoring applications, it is usually observed that the physical parameter sensed by sensors close to each other are identical. This property, referred to as spatial correlation, results in a high volume of redundant traffic through the network. By reducing the amount of redundant data transferred across the WSN towards the sink, we can not only reduce the network traffic but also conserve energy of the SNs. This is crucial for the WSN to become a viable alternative to the wired ones. Generally, SNs derive their power from on-board non-rechargeable batteries and as such, cannot be resurrected once their batteries die out since they are typically deployed in inaccessible or hazardous environments. The only alternative then is to replace them with other SNs - an expensive exercise. Thus, it is imperative that the SN be operational as long as possible. This can be accomplished through the use of energy efficient schemes that maximize

WSN lifetime.

Spatial and Temporal correlations result in the generation of a lot of redundant network traffic, carrying identical data. Data aggregation schemes exploit these correlations in the sensed data from neighboring nodes and successive data values from the same node respectively in order to send only the essential information. At the sink, the original data can be regenerated using the received data.

2.4 Sensor Data Representation Using Regression

Sensor nodes in the WSN typically generate a large amount of data. Extracting all of this data from the network to the BS would incur a lot of communication that drains the limited resources available in the sensors. Thus, approaches that reduce network communication, while still retaining the structure in the data would substantially extend network lifetime [15].

Instead of extracting all of the data generated in the WSN, if a model of the data is created and only the model coefficients are transmitted, the communication overhead can be significantly reduced. Regression is one of the many ways of modeling sensor data. In regression, a sensor typically measures a physical attribute (which can be considered a function of time) f(t). Thus, it collects a set of data values $f(t_1), \dots, f(t_m)$ at times t_1, ..., t_m. Suppose, we now have a set of 'k' basis functions H = (h_1, h_2,…, h_k) and would like to fit these 'k' functions to the sensed data values. We thus have the following:

$$\hat{f}(t) = \sum_i w_i h_i(t) \approx f(t)$$

where $\hat{f}(t)$ is an approximation of $f(t)$ and $w = (w_1, \dots, w_k)$ are the coefficients of the basis functions.

If the number of coefficients is far smaller than the number of sensed data values i.e. (k « m) then the coefficient vector becomes a compressed representation of the sensed values [15]. Then, rather than sending the data values explicitly, we could instead send the coefficients and thereby reduce the amount of data that is to be transmitted across the network.

2.5 Related Work

Although several schemes for programming and data aggregation in WSNs have been proposed in literature, few actually provide experimental validation and performance evaluation.

In the work proposed by Welsh et al. [16], WSNs are programmed using abstract regions – "a family of spatial operators that capture local communication within regions of the network which may be defined in terms of radio connectivity, geographic location, or other properties of nodes." Abstract Regions provide a programming model that supports various operators that allow nodes to query the state of neighboring nodes and perform efficient aggregation, compression and summarization of local data. Our work, instead, focuses on programming sensor nodes for aggregating spatially correlated attribute values. We assume that underlying mechanisms are present that create a Query Tree (QT) of nodes that perform in-network processing of data [2]. Each node in the QT performs polynomial regression based on its own reported readings and the coefficients obtained from its children.

iHEED [17] integrates node clustering with tree based aggregation and routing. Cluster heads (CHs) form an overlay network for data forwarding while other nodes only report data to the cluster heads. However, periodic re-clustering of the network is performed to select nodes with higher residual energy. This incurs significant overhead in cluster maintenance and routing. Simple operators such as AVG, MAX, SUM and COUNT are considered to study the impact of

data aggregation in conjunction with the node clustering in increasing the network lifetime. However, such a process of data aggregation fails to retain the inherent property of sensor data such as spatial-temporal correlations present in environmental and climactic data. Using TREG, we have implemented a data compression method with sensor motes without significant loss of accuracy. It can be integrated with any existing tree based query processing scheme for WSNs.

Distributed Kernel Regression [15] uses a kernel function to fit temporally correlated sensor measurements at different time intervals. Since the readings are also spatially correlated, a node also approximates its readings with that of its neighbors. But, to accomplish this, every pair of neighboring nodes needs to exchange messages which can involve substantial communication overhead. On the other hand, each tree node in TREG only sends a constant number of coefficients to their parent. In this way, the root in our scheme has a final set of coefficients and the information about the entire approximated region. Thus, the message overhead involved when packet exchange occurs on the motes based on TREG is significantly lower.

Chapter 3

Tree Based Polynomial Regression to Represent Sensor Data

3.1 Introduction

In this chapter, we discuss the Tree Based Polynomial Regression (TREG) [2] scheme that has been implemented on the wireless sensor motes and evaluated in this dissertation. The TREG scheme is a way of representing the sensor data in transit across the WSN towards the sink so as to reduce the amount of data transferred. In doing so, many significant advantages over the simple multi-hop data forwarding scheme are also realized, making it an attractive alternative to the same. We elaborate on these issues later in this chapter.

3.2 The TREG scheme

Sensed parameters which are typically physical attributes, exhibit a gradual and continuous variation over 2D Euclidean space [2]. In the real world, it is observed that physical attribute values and location are highly correlated and therefore sensors in close proximity [18], sensing physical attributes too exhibit similar property. The correlation in the sensed data is exploited by the TREG to aggregate the attribute values obtained from 'nearby' sensors. As a result, the underlying redundancy in the sensor data values reported by multiple sensors close to each other is eliminated thereby reducing the number and / or the size of the packet transmissions in the network. This leads to significant savings in the energy consumption by the sensor motes and consequently increased mote lifetimes.

The TREG data representation scheme begins with the creation of multiple attribute-based trees

[2] also referred to as query trees or QTs over the WSN. The creation of binary trees leaves the WSN with two types of sensor nodes with specific roles, as follows:

a) Sensing Nodes

The sensing or non-tree nodes essentially sense the physical attribute and report the value to the nearest tree node. These nodes do not perform any computation. The data may be transmitted to the tree nodes in a multi-hop fashion, following the shortest path.

b) Tree Nodes

The tree nodes do not sense any data and are used for storage and performing the regression computation. Every tree node has a number of non-tree sensing nodes reporting to it. Once it receives data from each of the non-tree nodes, the tree node performs multi-variate polynomial regression to calculate the regression coefficients.

Once the tree construction phase is over, the non-tree sensing nodes periodically sense attribute values and report the same to the nearest tree nodes. Each tree node waits to hear from all of the nodes reporting to it and then performs the polynomial regression operation to compute the regression coefficients that are passed on to the parent node in the tree instead of the actual attribute values reported by sensing nodes. The tree nodes at next higher level combine the coefficients obtained from their children tree nodes as well as the attribute values received from the sensing nodes reporting to it for computing their own set of regression coefficients. This process is repeated at each tree node and stops at the root of the QT. The root has a final set of coefficients that approximates the entire region. The root responds to any query received for attribute value at any point in the approximated region using the final set of coefficients computed.

3.3 Background

This section gives an overview of our network model. For completeness of the text a description of the aggregation tree used in the TREG scheme [2] and a background of the polynomial function is provided.

3.3.1 The TREG Network Model

A data aggregation tree (also called Query Tree or QT) is constructed over the WSN [2]. Let there be *n* nodes in the network. The QT consists of *t* nodes called tree nodes. The other nodes are referred to as non-tree (NT) nodes. Each NT node senses its environmental parameter(s) and reports it to its nearest tree node. The QT is well spread over the entire WSN so that *t* tree nodes are uniformly distributed over the network. This ensures that the attribute readings sent by NT nodes to the upward tree node incur a smaller hop count, thereby indirectly increasing the overall lifetime of the NT nodes. Recall that NT nodes only sense attributes while the QT is for storage only. The tree nodes solely perform data aggregation and do not do any sensing of data.

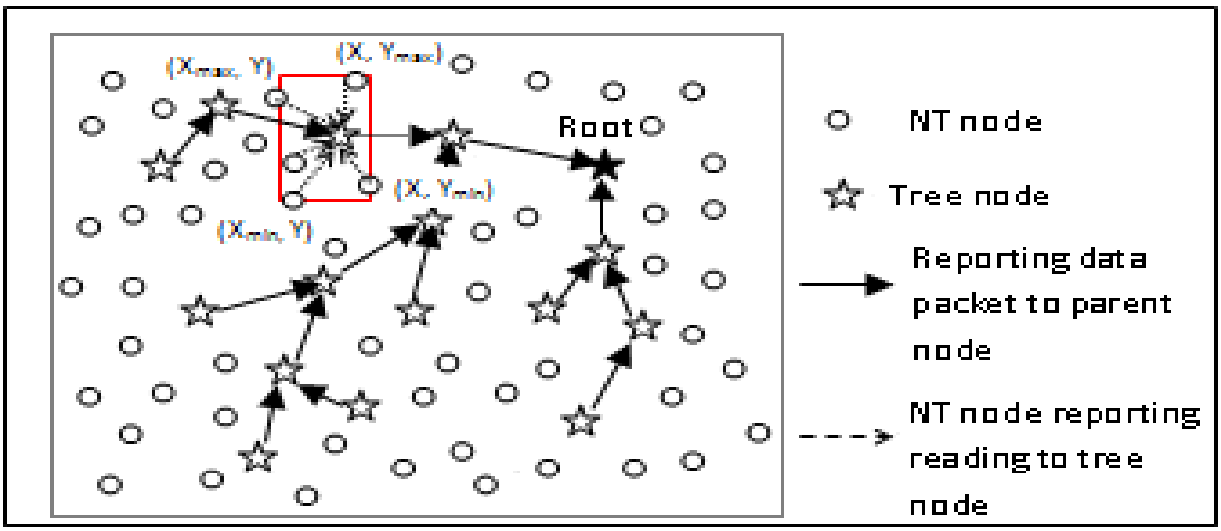

Figure 3.1: Network Model of a Query Tree (QT)

3.3.2 Polynomial Function

A QT is constructed over the network for data aggregation. The QT is denoted by the graph $G_{QT} = (V_{QT}, E_{QT})$ where V_{QT} is the set of tree nodes and E_{QT} is the set of edges joining the tree nodes. Each NT node periodically senses data and reports its reading to its nearest tree node. Since, attribute (say *z*) in a spatially correlated region is a function of sensor location denoted by the (x, y) coordinates, z is dependent on both the independent variables x and y, i.e., $z = f(x, y)$. Thus, $z_1 = \left(\left(f_1(x_1, y_1)\right), x_1, y_1\right)$ is one such attribute tuple where z_1 is the attribute value sensed by a node at location (x_1, y_1). Node *i* of a QT creates a function approximation $f_i(x_i, y_i)$ from the data reported to it by its nearest NT nodes. Each tree node then performs polynomial regression of the dataset $\left(x, y, f(x, y)\right)$(constructed from multiple such NT nodes reporting to it).

The polynomial function (i.e., *P*) has the following general form:

$$P = f(loc_1, loc_2, \ldots, loc_{ns}) = \beta_0 + \sum_{k=1}^{m} \beta_k loc_k$$

where, β_0, $\beta_1 \ldots \beta_k$ are the coefficients of the polynomial of degree *m* and $f(loc_1, loc_2, \ldots, loc_{ns})$ gives the set of attributes sensed at locations $loc_1 = (X_1, Y_1), \ldots, loc_{ns} = (X_{ns}, Y_{ns})$ (where n*s* is the number of NT nodes reporting to each tree node) and *ns* >> *m*.

Each child node sends two components to its parent: the information of polynomial (P), and the corresponding approximated area information as $\{X_{min}, Y_{min}, X_{max}, Y_{max}\}$ shown in Fig. 3.1. In order to specify the area represented by a polynomial, we use these four parameters X_{min}, Y_{min}, X_{max}, and Y_{max} to represent a rectangular area as denoted by the rectangle in Fig. 3.1. The four

parameters denote the four extreme ends of the two-dimensional area, spanning all the NT nodes reporting to one of the tree nodes. Of all these NT nodes, the node having minimum x-coordinate defines x_{min} and the one with maximum x-coordinate defines x_{max} and similarly for y_{min} and y_{max}.

Parent nodes at each level regenerate values of the sensed attribute by using the received *P* and the range. These data values are then combined with a node's own reported readings to calculate the new set of coefficients that will be passed to the next higher level. Thus, nodes at each level use *P* (obtained from their children) to improve the approximation function, *f(x,y)* and this procedure continues till the root is reached. After the data aggregation process is complete, the root has the final polynomial, *P*, approximating the data sensed over the entire network region, with a percentage error less than 6% [2]. Again, by sending only coefficients and the area approximated, the software simulation shows a Compression Ratio of 0.5 [2], over the case of each tree node simply transmitting T_{ns} packets. Apart from achieving high accuracy and Compression Ratio, it is possible to know the attribute value at any location in the aggregated network, even if there may not be any sensor at that location thereby proving the usefulness of the scheme even in a sparse network.

Let us consider an example where approximately 22 sensors report data to its tree node, i.e., n_s = 22. We consider the case of quadratic polynomial, i.e., $m = 2$. The previous work [2] shows that these coefficients of a polynomial of degree 2 can accurately capture the raw data from up to 40 sensors. With drastic increase in n_s, *m* can be increased, still saving a lot of data from being transmitted. Let tree node *B* in Fig. 3.1 receive the sensed data, $(X_i, Y_i, f(X_i, Y_i))$ from each of its neighboring NT node, *i* (denoted by the squares in Fig. 3.1). *B* performs regression on this data-set and creates polynomial with coefficients (β_0, $\beta_{1...}\beta_8$). It also computes the four parameters X_{min}, Y_{min}, X_{max}, and Y_{max}, i.e. the rectangular sensed area that is approximated by this polynomial.

It then sends the coefficients and the four parameters to its parent node *A*. *A* regenerates random locations (x,y) using the four parameters and generates the corresponding *f(x,y)* using *B*'s coefficients. It then appends this dataset of the form (x,y,f(x,y)) to the dataset it forms from the readings sent by its neighboring non-tree nodes. *A* performs polynomial regression on this combined data-set to generate a new set of coefficients. It then computes the four parameters from the x-y locations of non-tree nodes reporting to it. It combines this area with the area approximated by *B,* creating a new set of X_{min}, Y_{min}, X_{max}, and Y_{max}. Then *A* sends its calculated (β_0, $\beta_{1,\ldots}$ β_8) and X_{min}, Y_{min}, X_{max}, and Y_{max} to its parent. This process continues until the root computes the final polynomial. Using this polynomial, the root can get the attribute value at any point in the region spanned by the tree, as long as the location information is known to the root. These values can be obtained by choosing suitable x and y coordinates and substituting them back in *P*.

Reporting data through the aggregation process should be much more efficient in terms of the bandwidth and the latency, than sending individual data bits corresponding to a specific geographical co-ordinate.

3.4 TREG Advantages

TREG is a sensor data representation scheme for WSNs that builds multiple attribute-based trees using multi-variate polynomial regression and function approximation to eliminate redundant data transmissions. Every tree node only sends the regression coefficients and the boundary of the region approximated by it to its parent. Thus, the size of the packet that the tree node sends is always fixed, irrespective of the number of input data. Also, queries regarding the attribute value at any point in the approximated region (even those areas devoid of sensor nodes) can be replied

by the root itself, using the final set of computed coefficients and the coordinate values, instead of having to flood the entire region with a query packet. The scheme is scalable since the size of the data is constant and accuracy increases with increasing density. In summary, the primary advantages of this scheme are as follows:

• Attribute value can be obtained even in regions devoid of sensor nodes with a fair degree of accuracy.

• Value at any part of the approximated region can be obtained simply by querying the root node instead of flooding the regions, thereby reducing the number of packet transmission leading to significant savings in mote's energy consumption and increased lifetime of the WSN.

• Packet size transmitted by any tree node is fixed making the scheme extremely scalable.

In the following section, we summarize an earlier work [2], involving the software simulation of the TREG scheme.

3.5 Software Simulation Results

Below is a summary of the results that have been presented in [2]:

• A steady fall in the mean error and percentage error for synthetic data is observed as the depth of the query tree increases is observed.

• The percentage error varies up to a maximum value of 5.64 for a tree depth of 4.

• The authors present the percentage error for a total of 381 readings in the form of a graph that shows the number of readings falling in various error levels. This error distribution is found to be of a steadily decreasing nature.

• For a tree of depth 4, a majority of the error values fall in the range 0.0 % to 1.68 %.and the maximum error 5.64% is observed.

• The compression ratio, as defined by the authors, is about 0.02. The authors further suggest that the decreasing nature of the ratio with increase in the tree depth is due to further reduction in the output data at greater depths.

• For different depths of the QT, the size of the data packet transmitted by the root to the sink is constant when data compression is performed by all the nodes of the QT. The data packet size is independent of the number of nodes in the approximated region.

Chapter 4

Mote Programming

This chapter describes the hardware and software environment used for the implementation of TREG. We also discuss some of the challenges faced in working with such an environment.

4.1 Mote Hardware

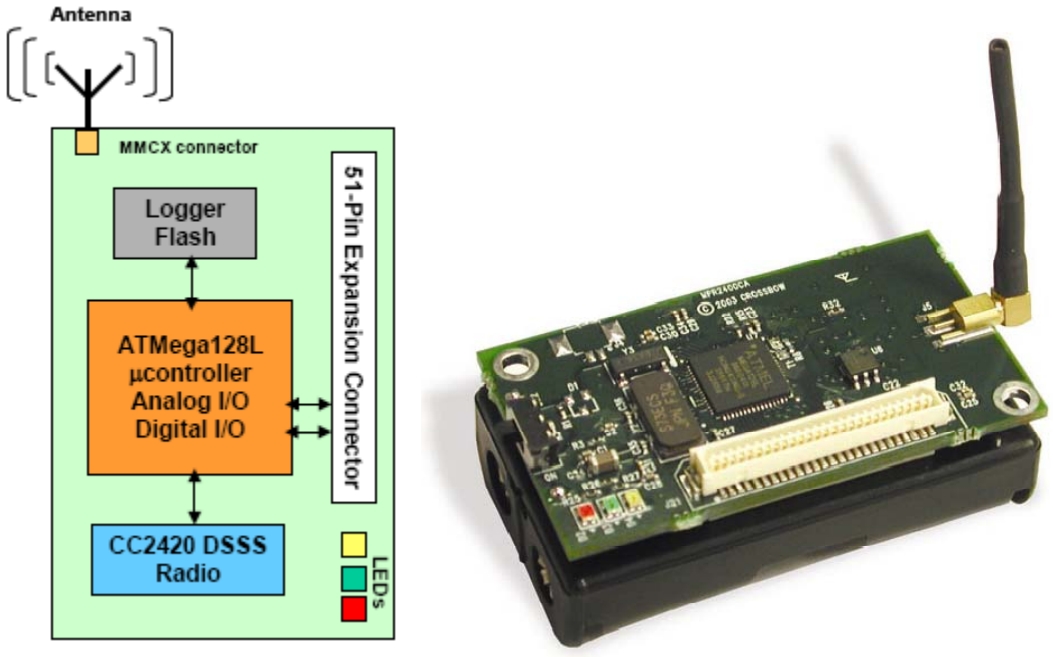

Figure 4.1: MPR2400 (MICAz) block diagram and Photo with standard antenna. [19]

4.1.1 The MICAz Sensor Mote

In our deployments, we have used UC Berkeley motes as sensor nodes. Specifically, we have made use of the crossbow MICAz mote (Fig. 4.1), which has a 2.4GHz, IEEE 802.15.4 (ZigBee) compliant radio [5]. Like the other members of the MICA family, the MICAz has an 8 bit Atmel

ATmega 128L (~8 MHz) processor. It has 128 KB program memory, 4 KB data memory and 512 KB non-volatile memory. The radio is a Chipcon CC2420 wideband transceiver employing

OQPSK ("offset quadrature phase shift keying") with half sine pulse shaping [19]. The 802.15.4 radio includes a DSSS (digital direct sequence spread spectrum) baseband modem providing a spreading gain of 9 dB [19], an effective data rate up to 250 Kbps [19] and AES-128 based hardware security. The RF transmission power is programmable from 0 dBm (1 mW) to –25dBm [19]. Lower transmission power reduces interference and decreases the radio power consumption from 17.5 mA at the full power to 8.5 mA at the lowest power. The radio transmission range varies from 75m to 100m outdoors and 20m to 30m indoors [5]. A pair of AA batteries and a DC boost converter provide a stable voltage source, though other renewable energy sources can be used as well.

The MICAz mote also comes with a 51-pin expansion connector that can connect to various sensor boards, containing temperature and photo sensors, sensors to measure humidity and pressure, along with accelerometers, magnetometers and microphones. Table 4.1 provides a summary of the essential specifications supplied by crossbow in the MICAz data sheet.

4.1.2 MIB510 Serial Interface Board

The MIB510 [19] (Fig. 4.2) is a multi-purpose interface board that is used with the MICA family of sensor motes. It provides an interface for a RS-232 Mote serial port that is used to program the MICAz. It can also be used to supply power to the devices through an external power adapter option. The MIB510 has an on-board in-system processor (ISP)—an Atmega16L located at U14—that is used to program the Motes. Fig. 4.2 shows the top view of the MIB510CA series of mote interface boards and identifies the main components.

Table 4.1: Summary of the MPR2400 (MICAz) Specifications [5]

Processor/Radio Board	**MPR2400CA**	**Remarks**
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 µA	Sleep mode
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals
RF Transceiver		
Frequency band1	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna, LOS
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 µA	Idle mode, voltage regular on
	1 µA	Sleep mode, voltage regulator off

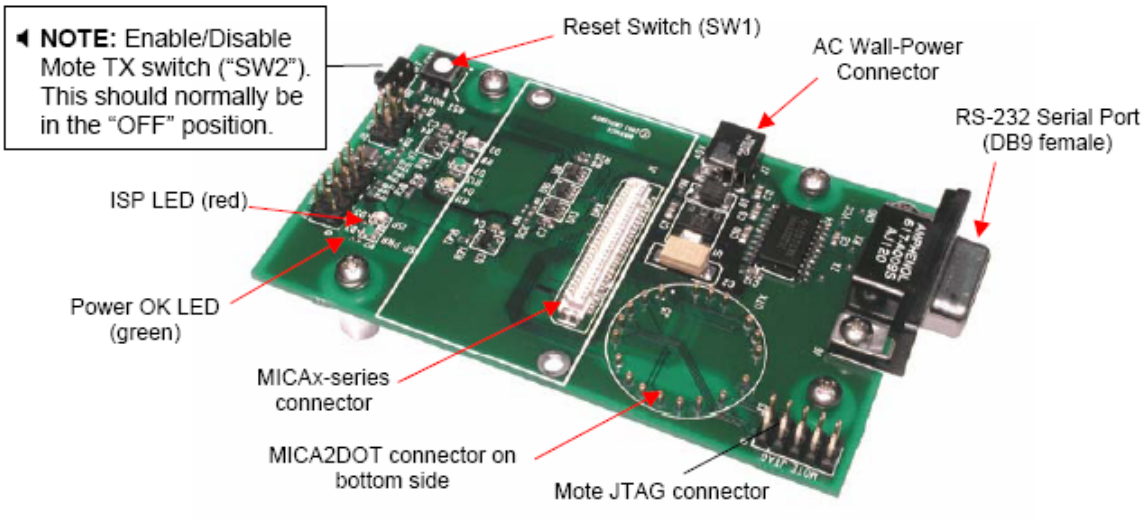

Figure 4.2: MIB510CA board. [19]

The Code is downloaded to the ISP through the RS-232 serial port, which then programs the code into the mote. The ISP and Mote share the same serial port. The ISP runs at a fixed baud rate of 115.2 kbaud [19]. The ISP continually monitors incoming serial packets for a special multi-byte pattern. Once this pattern is detected, it disables the Mote's serial RX and TX, and then takes control of the serial port.

The ISP processor is connected to two LEDs, a green LED labeled "SP PWR" (at D3) [19] and a red LED labeled "ISP" (at D5) [19]. The SP PWR (green) is used to indicate the power state of the MIB510 [19]. If the ISP (red) LED is on, the MIB510 has control of the serial port [19]. It also blinks once when the RESET (SW1) button is pushed and released.

4.2 Mote Software

The MICAz motes used in the experiments are powered by TinyOS [3], an open source operating system designed specifically for resource constrained WSNs. TinyOS is a component based, event driven OS written in nesC. Its various system components provide services such as basic

communication, power management, task scheduling, clock and timers, I/O and on-board EEPROM access to user applications. TinyOS does not have any blocking operations. Operations with long latency are split-phase, i.e., operation request and completion are distinct. Command completion is signaled using events. The following is a summary of the essential features of TinyOS:

- It has a very small footprint (just 400bytes). This is crucial since on-board memory available in the motes is typically just a few kilo bytes.
- TinyOS supports an event-driven architecture – the lower layer sends events to the higher layer. Thus there is very low overhead due to the absence of busy-wait cycles
- There is no kernel or process management in TinyOS. Virtual memory (VM) is another important OS concept that has been left out of current implementations, though few projects are trying to implement a mechanism for VM.
- It follows a component based model. System consists of components of an application and TinyOS base components. Components consist of Commands, Events, and Tasks.
- Commands are requests for actions from lower level components. Events are generated due to some hardware interrupt and provide a means for lower components to signal higher level ones. Tasks are operations that may take several clock cycles to complete. Once invoked, they run to completion and cannot be preempted by other tasks or interrupts. The Task scheduling follows a FIFO sequence.
- All memory is statically allocated to the program at compile time itself. This is critical to ensure that the program memory requirement does not exceed the available system memory.

We program the motes using the nesC language. nesC is an extension to the C language that supports the structure and event based concurrency model of TinyOS. The nesC compiler combines components from the TinyOS system libraries and user applications into a single executable file that runs on the mote hardware.

4.3 Challenges

Although TREG is based on a very simple concept of polynomial regression, its implementation posed many challenges due to an extreme resource limitation of the mote platform. A comparison between a typical present day desktop personal computer and the MICAz sensor mote reveals how constrained, the resources on the mote platform are as compared to a PC. A typical modern day PC consists of a dual-core 64 bit processor with speeds in the range of 2.0 – 3.0 GHz, few Gigabytes (8.0 GB MAX) of RAM and several Gigabytes (~250 GB) of secondary storage. On the other hand, the current generation of sensor mote typically has an 8-bit micro-controller, few kilobytes of Program ROM (~128 Kb) and Data RAM (~4 Kb) respectively. This leads to many practical difficulties when working with sensor motes. We enlist some of the major problems that we faced in our work using the wireless sensor motes' hardware and software platform.

Firstly, the TREG algorithm involves a number of successive Matrix Operations (some involving floating point operations) and a final Gaussian Elimination step to compute the set of coefficients. Since floating-point operations are not supported by the 8 bit Atmel processor, they have to be done entirely in software. This definitely impacted the performance since such operations have been shown to be faster in the hardware than in the software. As the mote hardware becomes more powerful day by day, we expect them to support floating point opera-

tions sometime in the near future.

Secondly, the TREG implementation requires the existence of an aggregation tree in which nodes periodically compute a set of coefficients, based on data received from their children and pass them on to their parent which repeats the process. This means that the parent-child links thus established, must be fairly stable to ensure that successive packets from a given child are passed to the same parent most of the time. TinyOS 1.x does not have any explicit tree based routing support though it does support a multi-hop routing engine to forward packets from leaf nodes to the sink. In multi-hop routing, packets from source to the same destination may travel along different routes during any given session. This renders such a scheme ineffective for our purpose. On the other hand, TinyOS 2.x implements a default tree routing algorithm using the collection component to build collection trees. The collection component of TinyOS 2.x provides interfaces that enable an intermediate node to intercept a packet that it is supposed to forward and process the contents. For our work, we have assumed a two phase operation where sensing nodes first discover the closest tree nodes to forward data packets. Also, the aggregating nodes organize themselves into a tree rooted at a node attached to the workstation acting as the BS.

Thirdly, TinyOS does not support dynamic memory allocation. This means that all data structures need to be allocated a fixed amount of memory beforehand. The allocated size should be such that the performance of the algorithm remains satisfactory, yet at the same time not to exceed the 4 KB data memory available to the application. For our experiments, we required 2-D arrays to be defined to represent matrices. The maximum dimensions of any matrix that we used are (12x4) where 12 is the number of samples and 4 is the number of coefficients computed by TREG. Our attempt at implementing a quadratic polynomial with 9 coefficients failed since the program data memory exceeded currently available data memory (4KB) on the MICAz motes.

Since TinyOS currently does not support any form of virtual memory, it is impossible to make use of the on-board flash memory for storing the program data not fitting in the RAM.

Finally, no drivers are currently available in TinyOS 2.x for the MTS/MDA series of sensor boards from crossbow. Because of this reason, we have been unable to physically record temperature data. Instead, we have programmed the sensor motes with carefully selected data set that adheres to the spatial correlation requirement. Since the focus of our study is on the computation rather the nature of the sensed data itself, we feel that this would have no bearing on the outcome of our scheme. The study of the effectiveness of the algorithm viz. time correlation among the data samples has been left as future work since that would be impossible without the availability of sensor board drivers. It may be noted that only minor changes, involving the inclusion of the Read interface and wiring it up to the corresponding interface provided by a system component that gets the thermistor readings, would be necessary. Also, since the values returned by the ADC are 16 bit integers, an appropriate conversion routine such as crossbow's "*xconvert.c*" will be required to convert the values returned by the ADC into floating point temperature values.

Chapter 5

TREG Scheme Implementation

In this chapter, we discuss a detailed the implementation of the TREG algorithm on the MICAz sensor motes using the TinyOS / nesC programming environment. We start with a mathematical discussion of the TREG algorithm provided in [2] and then proceed to describe the nesC implementation.

5.1 Mathematical Model [2]

Each tree node stores the attribute value sent by each of the nearest NT sensor nodes. It may be recalled that the NT nodes only sense attributes while the QT is for storage only. Since the measured attribute varies with respect to the space in a continuous manner, the values stored at the tree node can be considered as function values having two inputs, x and y. Since attribute (say z) in a spatially correlated region, is a function of sensor location which can be denoted by the (x, y) co-ordinates, therefore z is dependent on both the independent variables, x and y, i.e., z = f(x, y). Thus, $z_1 = ((f_1(x_1, y_1), x_1, y_1)$ is one such attribute tuple where z_l is the attribute value sensed by a node at location (x_l, y_l). Node 'i' of the QT creates a function approximation $f_i(x, y)$ from the data reported to it by its nearest NT nodes.

A polynomial equation is generated with three input variables (z = f(x, y), x, y) for all the data points in one particular node of QT using multivariate polynomial regression [20]. A general multi-linear regression model is as follows [20]:

$$z = f(x_1, x_2, \cdots, x_m) = a_0 + \sum_{k=1}^{m} a_k x_k \,, \tag{5.1}$$

where x_1, x_2,…, x_m are the independent variables called predictors of the model and z is the dependent variable. The observations are sampled periodically and the observed values of the variable 'z' are used at the particular levels of x_k to estimate 'a'. 'z' is the n-element vector of observed values and 'a' is the (m+1)×1 vector estimate of 'a'.

Applying least-square criterion, it is necessary to minimize the squared error, i.e.

$$F(a) = (Xa - z)^T . (Xa - z) \tag{5.2}$$

where

$$\mathrm{X} = \begin{pmatrix} 1 & \mathrm{x}_{11} & \mathrm{x}_{21} & \cdots & \mathrm{x}_{\mathrm{m}1} \\ 1 & \mathrm{x}_{12} & \mathrm{x}_{22} & \cdots & \mathrm{x}_{\mathrm{m}2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathrm{x}_{1\mathrm{n}} & \mathrm{x}_{2\mathrm{n}} & \cdots & \mathrm{x}_{\mathrm{mn}} \end{pmatrix} \tag{5.3}$$

$$z = \begin{pmatrix} z_1 \\ \vdots \\ \vdots \\ z_n \end{pmatrix} . K \ , \tag{5.4}$$

and,

$$a = \begin{pmatrix} a_1 \\ \vdots \\ \vdots \\ a_m \end{pmatrix} . K \ . \tag{5.5}$$

For minimization, partial differentiation of F(a) with respect to 'a' must be equated to zero.

$$\text{that is, } \nabla_a . F(a) = \nabla_a . (Xa - z)^T . (Xa - z) = 0 \ .$$

Again,

$$(Xa - z)^T . (Xa - z) = (\nabla_a . (Xa - z)^T) . (Xa - z) + (\nabla_a . (Xa - z)^T) . (Xa - z) \ ,$$

$$= 2X^T (Xa - z) \ .$$

$$= 2X^T Xa - 2X^T z = 0$$

Thus, the normal equation obtained is as:

$$X^T Xa = X^T z \,. \tag{5.6}$$

The system has a solution if X^TX is not singular, i.e. it has an inverse. Therefore, multiplying both sides of Equation (5.6) by $(X^TX)^{-1}$, we get $a = (X^T X)^{-1}.X^T z$, where $(X^T X)^{-1}.X^T$ is called the (Moore–Penrose) pseudo-inverse of the matrix X and is a generalization of the inverse X^{-1}.

Using polynomial regression of this model, we get the following equations analogous to Equations (5.3) – (5.5). Here, 'x' and 'y' are the independent variables.

$$X = \begin{pmatrix} 1 & y_1 & y_1^2 & x_1 & x_1y_1 & x_1y_1^2 & x_1^2 & x_1^2y_1 & x_1^2y_1^2 \\ 1 & y_2 & y_2^2 & x_2 & x_2y_2 & x_2y_2^2 & x_2^2 & x_2^2y_2 & x_2^2y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & y_n & y_n^2 & x_n & x_ny_n & x_ny_n^2 & x_n^2 & x_n^2y_n & x_n^2y_n^2 \end{pmatrix}, z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \text{ and } \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix}. \tag{5.7}$$

For estimating β, a unique inverse of X should exist, i.e., X^TX must be of full rank (m+1) [23], given that β is a (m+1) × 1 vector. In other words, n » (m + 1) and no column of X can be expressed as weighted linear combination of any set of other columns are necessary conditions.

Here, 'n' is the number sensor locations whose readings are fitted to a polynomial. For each tree node performing regression, $n = n_s$:

$$f(x,y) = \beta_0 + \beta_1 y + \beta_2 y^2 + \beta_3 x + \beta_4 xy + \beta_5 xy^2 + \beta_6 x^2 + \beta_7 x^2 y + \beta_8 x^2 y^2 \,. \tag{5.8}$$

Equation (5.8) can also be written as

$$f(x,y) = \sum_{i=0,j=1}^{i=8,j=9} \beta_i \,.\, \emptyset_j \,, \tag{5.9}$$

where $\emptyset_1 = 1, \cdots, \emptyset_9 = x^2.y^2$.

Again, each tree node creates the matrix, A_m (from the readings reported by the 'n' NT nodes) for calculating the polynomial. A_m is a square matrix of size 9×9.

The (i-j)th element of

$$A_m = \sum_{l=1}^{n_s} \emptyset_i(x_l, y_l).\emptyset_j(x_l, y_l) \text{ ,} \tag{5.10}$$

where i = j .

At any instant, each tree node has the tuples, $(x_1, y_1, z_1) \ldots (x_{n_s}, y_{n_s}, z_{n_s})$ corresponding to the 'n_s' sensing nodes. Each tree node creates the matrix F with these tuples. F is a matrix of size, 9×9 where the ith element,

$$F_i = \sum_{l=1}^{n_s} z_l.\emptyset_i(x_l, y_l), \text{ where } \emptyset_1 = 1, \ldots, \emptyset_9 = x^2.y^2 \tag{5.11}$$

At each tree node, $\beta_0, \beta_1, \ldots, \beta_8$ are calculated from Equation (11):

$$\beta = A_m^{-1} \times F \tag{5.12}$$

Equations (5.7), (5.10)–(5.12) together constitute the method of Gaussian elimination [21] for solving equation and computing inverse of a matrix. We explain the Gaussian elimination method of solving a system of linear equations in the next section.

When 'β', obtained from Equation (5.12), is used with a given location (x, y), we solve z = f(x, y) to retrieve the attribute value at a node location (x, y). f(x, y) is the final function available at the root of the QT.

Each tree node uses Equation (5.12) to generate the coefficients and sends this set to its parent.

Nodes at each level regenerate values of the sensed attribute by using these coefficients obtained from their children. These data values are then combined with a node's own reported readings to calculate the new set of coefficients that will be passed to the next higher level.

5.2 Gaussian Elimination Step in Polynomial Regression

In linear algebra, the Gaussian elimination is an algorithm for determining the solutions of a system of linear equations [22]. It consists of two parts, (a) forward elimination and (b) backward elimination.

(a) Forward Elimination (FE): reduces a given system to either triangular or echelon form. If the result of FE is a degenerate equation with no solution, then the system has no solution.

(b) Backward Elimination (BE): uses back-substitution to find the solution of the linear system.

Example: Consider the following system of equations [22]

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ x_1 + 3x_2 - 2x_3 = -1 & L_2 \\ 3x_1 + 5x_2 + 8x_3 = 8 & L_3 \end{cases}.$$

In Matrix form we can write the above system as follows:

$$A = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 3 & -2 \\ 3 & 5 & 8 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, b = \begin{pmatrix} 2 \\ -1 \\ 8 \end{pmatrix},$$

Step 1: Eliminate x_1 in the lines L_2 and L_3:

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ \quad x_2 - 4x_3 = -3 & L_2 \leftarrow L_2 - L_1 \\ \quad -x_2 + 2x_3 = 2 & L_3 \leftarrow L_3 - 3L_1 \end{cases},$$

Step 2: Eliminate x_2 in lineL_3:

$$\begin{cases} x_1 + 2x_2 + 2x_3 = 2 & L_1 \\ \quad x_2 - 4x_3 = -3 & L_2 \\ \quad\quad - 2x_3 = -1 & L_3 \leftarrow L_3 + L_2 \end{cases},$$

Step 3: By BE we solve the linear system to get the solution vector x:

$$x = \begin{pmatrix} 3 \\ -1 \\ 1/2 \end{pmatrix}.$$

Gaussian elimination on an $n \times n$ matrix requires approximately $2n^3 / 3$ operations.

5.3 nesC Regression Algorithm

Our implementation divides the sensor network into two types of nodes namely Sensing nodes and TREG nodes. The Sensing nodes periodically sample ADC readings and pass them to their closest tree (TREG) node. The TREG nodes are for storage only and perform the polynomial regression over the region approximated by them.

Each of the sensing nodes runs the "*SensorTregAppC.nc*" application to periodically send its X and Y coordinates and a floating point value from a carefully chosen temperature data set that satisfies the spatial correlation requirement. As and when drivers for the MDA 300 sensorboard become available, the motes can be programmed to sample the ADC thermister reading and send it to the closest tree (TREG) node. Each of the tree nodes runs the "*TregAppC.nc*" application

which implements the TREG polynomial regression algorithm.

In our implementation, we assume that the tree nodes can have both sensing nodes as well as children tree (TREG) nodes. The basic program logic flow is as follows:

1. Each tree node waits until it hears from either one of the sensing nodes or its children nodes.

2. Upon receiving such a packet, it starts a one-shot timer for 30 – 45 seconds to allow all the nodes reporting to it to send their packets. It also maintains a counter to count the number of packets received from the sensing nodes (spktcount) and the tree nodes (tpktcount) respectively. The counter is incremented each time a packet is received successfully.

3. As soon as the timer fires, the node checks the "tpktcount" variable to see if any TREG packets have been received. If no TREG packets have been received, then it proceeds to compute coefficients directly by posting the "*computeCoefficientsTask()*".

Table 5.1: *computeCoefficientsTask()*

```
scount ← Number of Sensor Packets received.
X^TX ← Product Matrix of X^T and X
X^TZ ← Product Matrix of X^T and Z
01:  computeCoefficientsTask() {
02:      MatrixInit(X^TX);          //Initialize the matrix X^TX[4][4] to 0.0
03:      MatrixInit(X^TZ);          //Initialize the matrix X^TZ[4][1] to 0.0
04:      TransposeCalc(X, X^T, scount);          //Compute the Transpose X^T of X
05:      Matrix_Mult(X^T, X, X^TX, 4, scount);   //Compute the product of X^T & X
06:      Matrix_Mult(X^T, Z, X^TZ, 1, scount);   //Compute the product of X^T & Z
07:      Gaussian (4, XTX, XTZ, B);   //Compute the Beta Vector using Gaussian Elimination.
08:  }
```

The pseudo code for the "*computeCoefficientsTask()*" reveals a sequence of Matrix operations and a final Gaussian Elimination step to compute the coefficients. The Matrix X is an (*n* x 4) matrix, each of whose rows has the elements 1, x, y and (x.y). Matrix Z is an (*n* x 1) matrix comprising of attribute values corresponding to the x, y values of Matrix X. '*n*' is the number of samples that are available for the computation. In our experiments, we have taken $n = 20$. We first find the Transpose X^T of matrix X followed by the matrix products X^TX and X^TZ. X^TX has dimensions (4x4) while X^TZ is a (4x1) matrix. The two matrices thus obtained are input to the Gaussian Elimination Step (*Gaussian*) which outputs the Beta vector of coefficients. The generic Gaussian Elimination algorithm is of the order of $O(n^3)$. However, since the TREG algorithm inputs matrices with '*n*' at most equal to 9, this should not be a problem. In our experiments we have set $n = 4$ as described above.

Each TREG node then passes to its parent, this beta vector of coefficients along with the (x_{min}, y_{min}) and (x_{max}, y_{max}) values taken over all the sensing nodes in the sub-tree under the current parent that collects reports from tree nodes.

However, if any TREG packet have been received (*tpktcount > 0*) then the "*regenerateAttributeValuesTask()*" is first posted to regenerate attribute values for each TREG packet received based on the (*xmin*, *ymin*), (*xmax*, *ymax*) and the Beta vector of coefficients. The pseudo code for the "*regenerateAttributeValuesTask()*" is shown in Fig. 5.4. We define *N* as the number of values to be regenerated for each received TREG packet. It is obtained by dividing the difference of 20 and *spktcount* by *tpktcount* (line 06).

Table 5.2: *regenerateAttributeValuesTask()*

01: 20<- max no. of rows of matrix X
02: spktcount <- No. of SensorPkts received

```
03:  tpktcount <- No. of TregPkts received
04:  N <- No. of values to be regenerated for each received TregPkt
05:
06:  N = (int)((20 - spktcount) / tpktcount);
07:  regenerateAttributeValuesTask()  {
08:     Repeat for each received TregPkt {
09:       tempX = xmin_coord;
10:       tempY = ymin_coord;
11:       xinc = (int)(( xmax_coord - xmin_coord) / N);  //Compute the X coordinate increment
12:      yinc = (int)((ymax_coord - ymin_coord) / N);    //Compute the Y coordinate increment
13:      rcount = 1;       //Initialize the regeneration counter.
14:      Repeat {
15:       //Create a new SensorPkt record
16:       //Append it to the existing list of SensorPkt records
17:         newsensorpkt.x_coord = tempX;
18:         newsensorpkt.y_coord = tempY;
19:         newsensorpkt.data=  (beta[0]  *  1)  +  (beta[1]  *  tempX)  +  (beta[2]  *  tempY)  +
(beta[3]*tempX*tempY);       //Regenerate the Z values
20:         spktcount = spktcount + 1;   //Increment spktcount for each sensor record regenerated.
21:         tempX = tempX + xinc;       //Get the next X value
22:         tempY = tempY + yinc;       //Get the next Y value
23:         rcount = rcount + 1;
24:     } while ((tempX < xmax_coord) and (tempY < ymax_coord)) and (rcount <= N))
25:     }
26:       //Once we've regenerated Z values we can read sensor log to regenerate coefficients.
27:     post computeCoefficientsTask();
28: } //End task regenerateAttributeValuesTask()
```

For each TREG packet we regenerate N values in the following manner:

We calculate the fixed increments (*xinc* and *yinc*) in X and Y values as shown in lines 14 and 17 respectively. We then start regenerating attribute values starting at (*xmin, ymin*) using the formula $Z = B[0].1 + B[1].X + B[2].Y + B[3].X.Y$. New X and Y values are obtained by adding the increments calculated previously to the current X and Y values. Each pair of X and Y values thus obtained is used to calculate the corresponding attribute value (line 19). As shown by the lines 14 thru 24 of the pseudo code, we do this while ensuring that we do not exceed the number *N* of regenerations per TREG packet. We also ensure that the boundary conditions are included so as to cover the entire region and improve the accuracy of the scheme. The worst case occurs when no SENSOR Packets have been received and hence all values need to be regenerated before computation.

The worst case complexity depends on the number of TREG packets received and the number of values to be regenerated. Since any given TREG node has a fixed number of children and the number of values to be regenerated is fixed, the complexity is of the order O(*n*) where *n* is equal to the number of TREG packets received.

The regenerated X, Y and attribute values are appended to the data structure that holds the values obtained from the received SENSOR Packets. At each tree node, this procedure is repeated in a distributed manner for each TREG packet that has been received. Once we have generated values for each TREG packet, we proceed to compute the coefficients as explained above till the root is reached. The root computes the final set of coefficients for the entire region in a similar fashion as other tree nodes and passes them over the serial port to the workstation at the other end. With the final set of coefficients and the (X_{min}, Y_{min}) and (X_{max}, Y_{max}) values we can obtain approx-

-imated temperature value at any point in the region.

Chapter 6

Performance Evaluation

In this chapter, we present experimental results of TREG implementation on MICAz motes. Our results essentially corroborate the simulation results presented in the earlier work [2]. Additionally, we also show that the energy consumption of the entire system is considerably lower as compared to a simple tree based data forwarding scheme. In evaluating TREG we consider:

(1) The accuracy of approximation of the sensed parameter,

(2) The comparison of computational delay to the transmission delay,

(3) The overall system energy consumption, and

(4) Compression ratio.

Fig. 6.1 shows the network topology used for performing the experiments using a tree of depth 2. The sensors deployed over a 400m x 400m region are assumed to be our target area for monitoring. A temperature variation in the range 29° - 34° C is assumed over the region as shown in the figure. Such a large temperature variation can be considered to be realistic for a highly unstable region. The accuracy of our scheme would increase if a smaller variation in the value of the sensed parameter is present. As can be seen in the diagram, we had some tree (TREG) nodes and sensing nodes – a minimum of 5 nodes reporting to each of the leaf nodes and others reporting their values to the non-leaf nodes including the root. The root itself is connected to a workstation using the serial port. After performing the polynomial regression over the entire

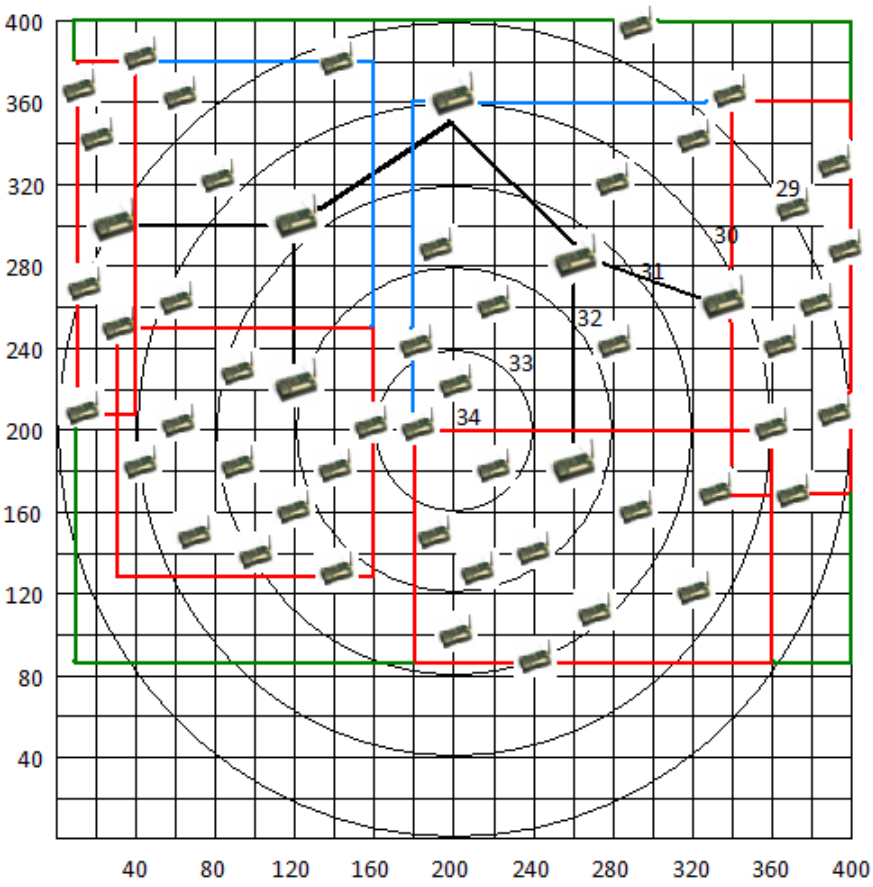

Figure 6.1: Network topology for tree depth = 2 and 48 nodes

region, the root sends the final set of coefficients along with the boundary of the region over the serial port. The experiments have been conducted at the University of Cincinnati using 56 MICAz motes. Although, a larger number of motes would have enabled us to cover more cases, we believe that we have nevertheless been able to show the effectiveness of the scheme on a reasonable size sensor network test bed. The locations of the sensors have been pre-determined based on a topology map similar to the one shown in Fig. 6.1 and the values have been hardcoded in the program. Again, the sensors currently do not sense the physical attribute because of the lack of sensor board drivers. Instead, they have been programmed to return a constant value selected from a dataset consisting of spatially correlated data samples.

6.1 Accuracy

In order to evaluate the accuracy of the scheme, we first conducted experiments on the sensor test bed using trees of depth 1, 2 and 3. Then, we chose a random set of 50 data points from our network topology map. Using the coefficients obtained at the end of the experiments for each, case we generate the temperature values using the polynomial explained earlier in this document. These are then compared with the actual values as obtained from the contour map to determine the accuracy. Fig. 6.2 shows the results for a tree of depth 1. We considered 3 cases – when the

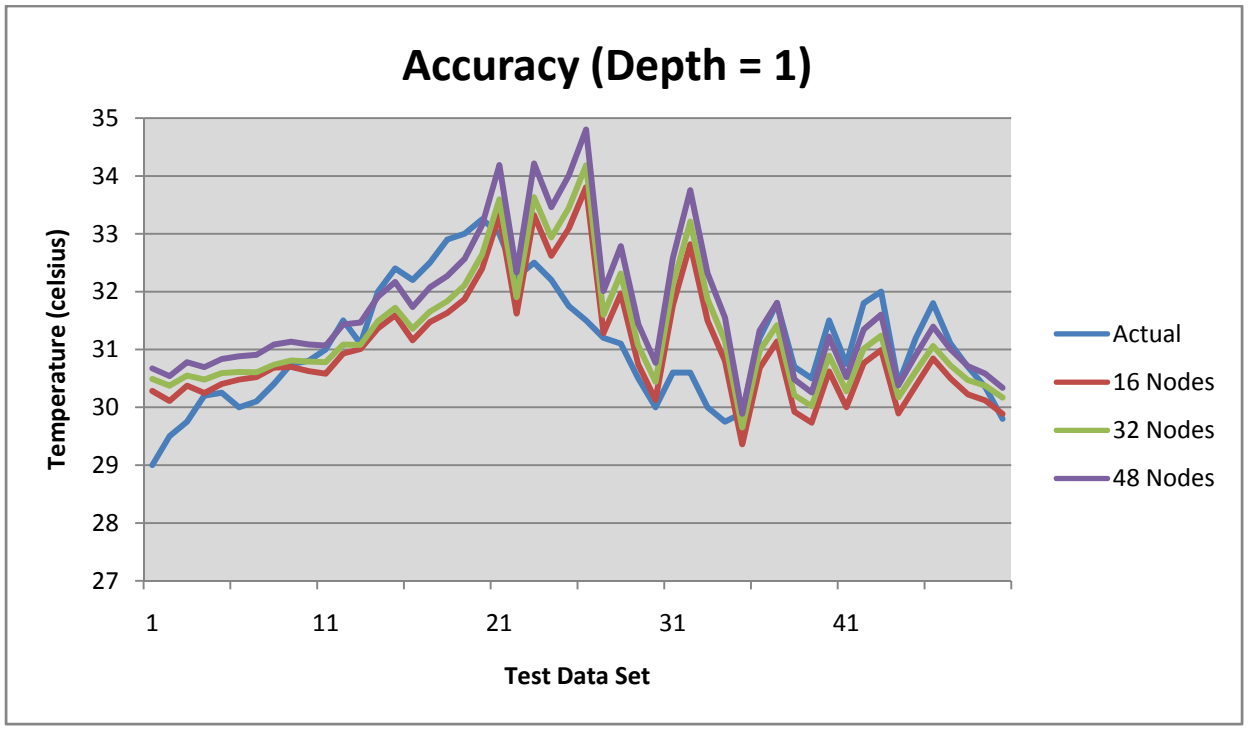

Figure 6.2: Accuracy for tree depth = 1

number of nodes in the network is 16, 32 and 48 respectively. As can be seen from the figure, all three curves closely follow the actual curve most of the time. Thus, we can see that to a large extent, the polynomial has indeed managed to capture the temperature distribution in the network successfully. We also find that the 3 curves also follow fairly closely to each other.

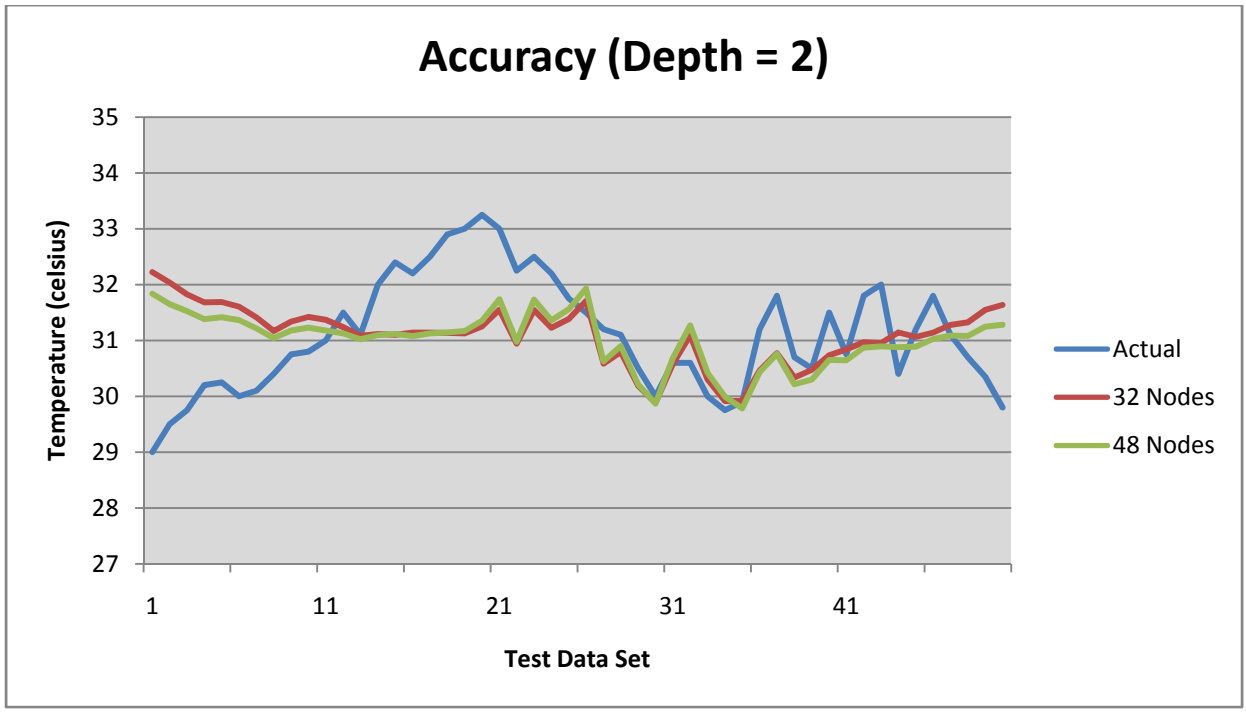

Figure 6.3: Accuracy for tree depth = 2

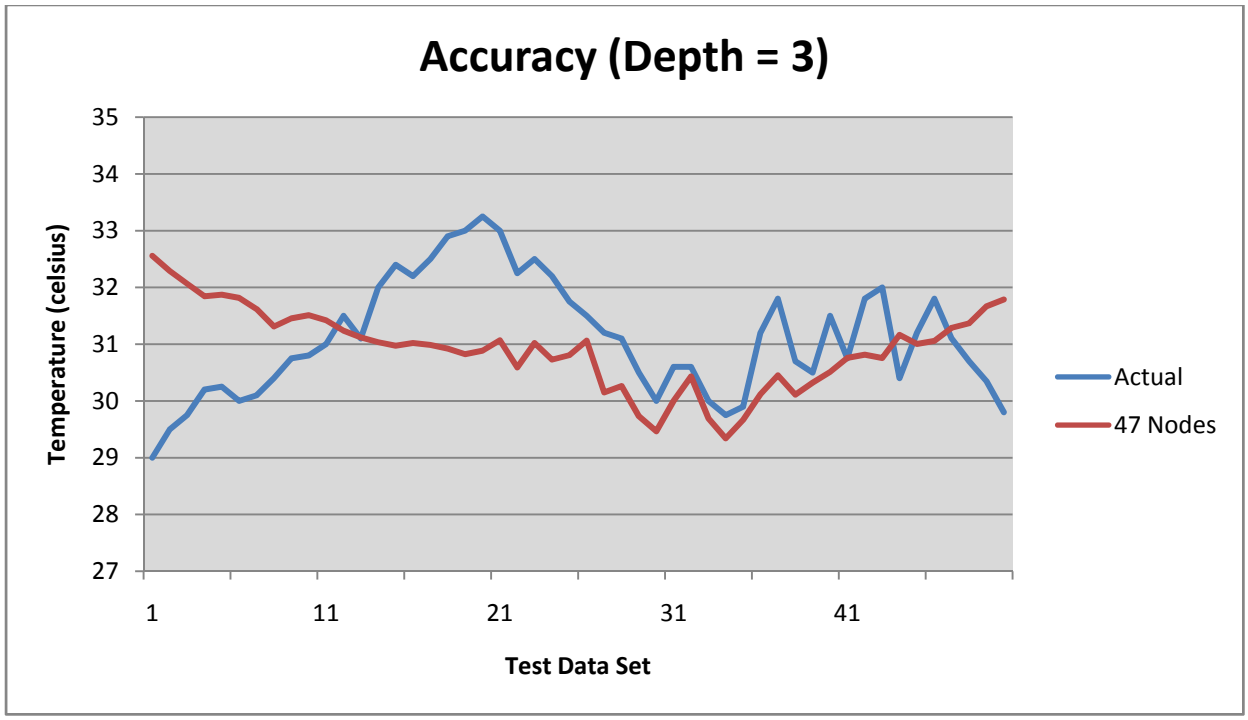

Figure 6.4: Accuracy for tree depth = 3

This is because the increase in the number of motes in the network from 16 to 48 does not increase the network density significantly. Since, accuracy in our case is directly proportional to network density, we find no major improvement in the accuracy by a small increase in the number of motes.

Figs. 6.3 and 6.4 show the same curves for trees of depths 2 and 3 respectively. In both, the cases, we find that the curves of regenerated temperature values diverge from the actual curve to some extent. This is as expected since the network size increases with increasing the tree depth. However, the nodes of nodes remain the same in our experiments (32 or 48 nodes). Therefore, the network density decreases in these two cases, thereby resulting with poor accuracy.

6.2 Percentage Error

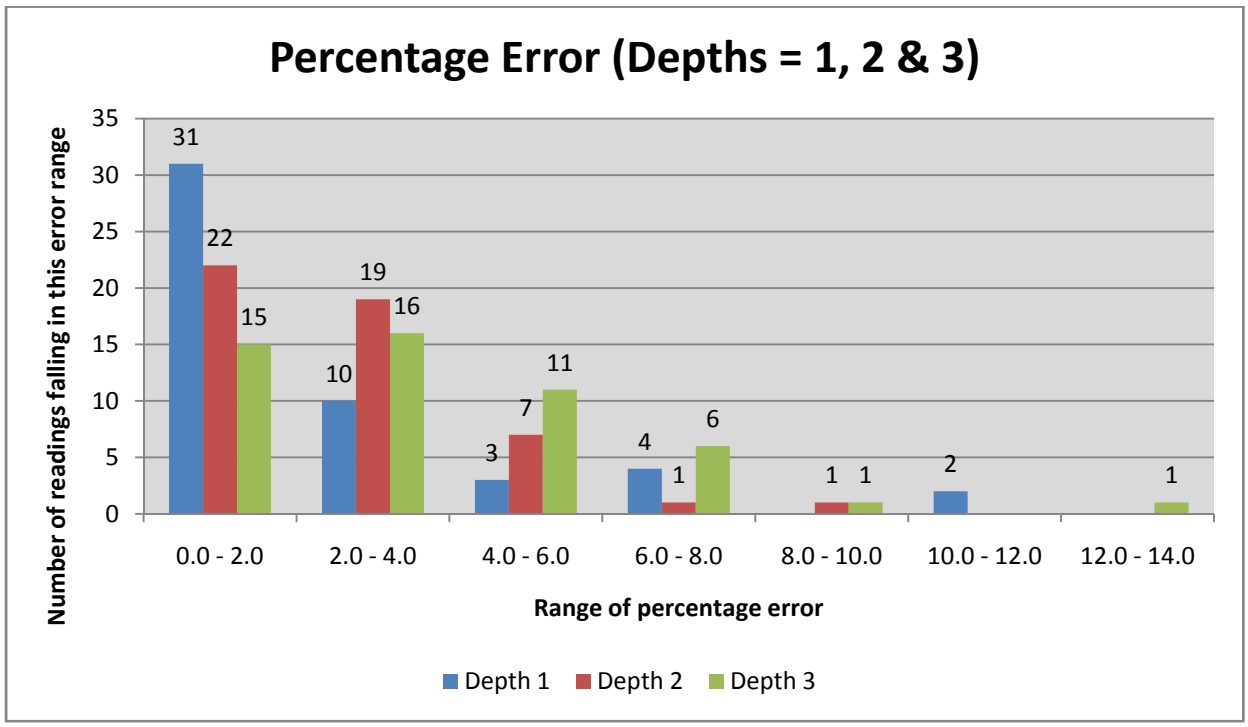

Figure 6.5: Percentage errors for tree depths 1, 2 & 3

The Percentage Error is calculated ass,

$$E = \left(\frac{|z-\bar{z}|}{z} \times 100\right),$$

where 'z' is the actual attribute value and '$\bar{z}$' is the computed value of the attribute.

Fig. 6.5 shows the variation of percentage error for various tree depths. Similar to the simulation results presented in [2], we find that the majority of the points where the approximate is calculated fall in the lower half of the graph. A steady decrease in the number of readings falling in higher error ranges is also evident from the graph. The maximum percentage error was 10.49% for depth 1, 9.78% for depth = 2 and 12.27% for depth = 3.

6.3 Computational Vs Transmission Overhead

We measure the computational overhead in terms of the time taken to perform the computation in the motes. Since there is no in built mechanism to obtain the computation time, we start a program timer as soon as the task to compute the coefficients or to regenerate attribute values is posted depending on the reception of TREG Packets and stop the timer once the coefficients have been computed.

To calculate the time it takes for a packet to be transmitted from a sensing node to the root of the tree, we measure the time it takes for a packet sent by a sensing node to reach the root of the tree. This time is measured by simply making the intermediate nodes to forward the packet received from a sensing node, instead of performing any computation.

Fig. 6.6 shows the variation of both computation time and transmission time with increase in tree

depth. It's evident from the figure that the transmission time per packet increases much more than the increase in the computation time with increasing tree depth.

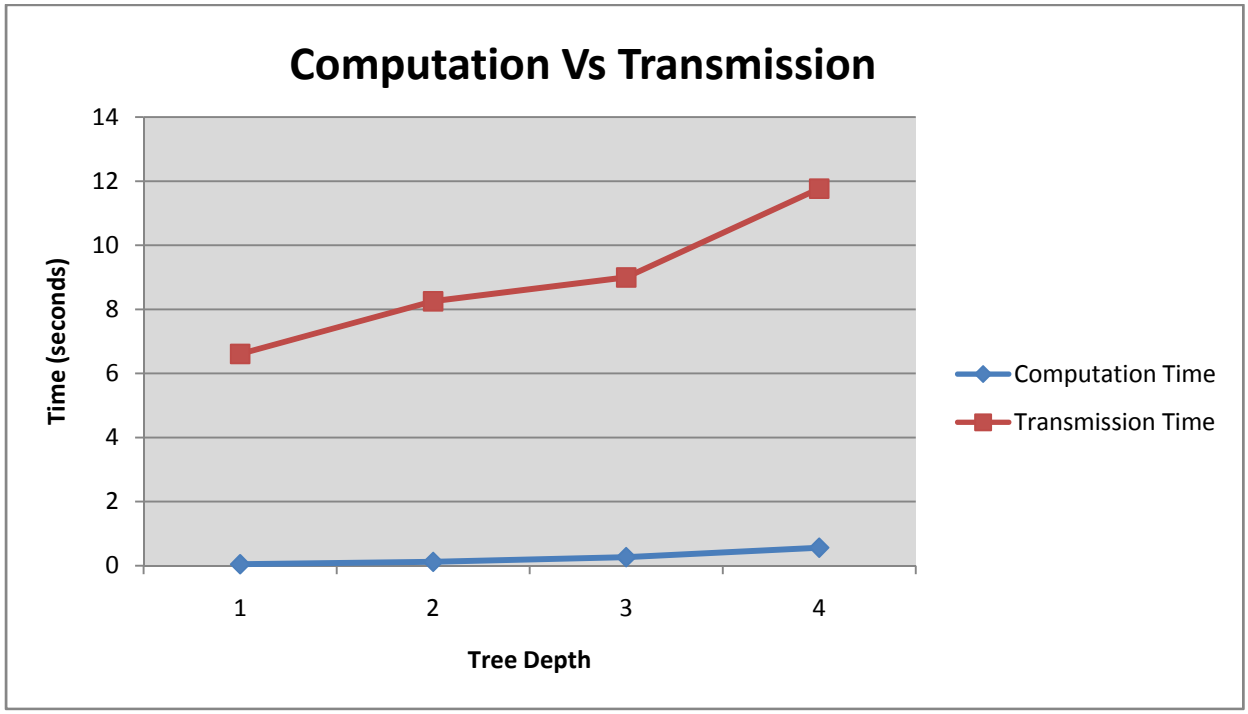

Figure 6.6: Variation of Computation/Transmission times with tree depth

The computation time using TREG is independent of the number of sensing nodes. Thus, for the same tree height, the system computation time remains constant, even if the number of sensing nodes increases. However, with increasing tree depth, the number of tree nodes increases. Consequently, the system performs more computation as shown by the small increase in the computation time. On the other hand, the total system time for transmission increases significantly with an increase in the number of sensing nodes. This is shown in Fig. 6.7 for a tree of depth = 1, where it can be seen that the system time for computation remains constant when the number of sensing nodes increases from 16 to 48 while the net system time for transmission increases significantly.

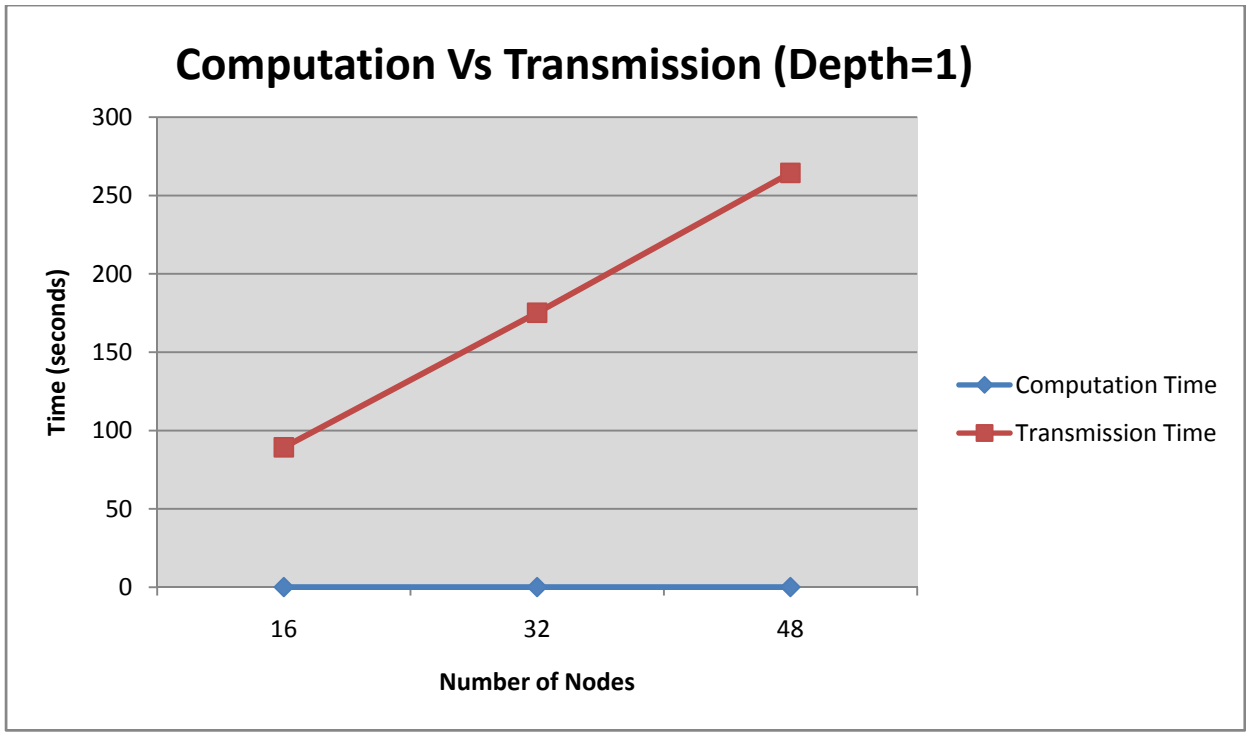

Figure 6.7: Variation of Computation/Transmission times with number of sensing nodes

6.4 Total Energy Consumption

The total system energy consumption is defined as the sum of all the energies consumed by the nodes of the network in one round of experimentation. This includes the energy used up by the sensing nodes in transmitting the SENSOR Packets (16 bytes), the energy consumed in the tree nodes to receive the SENSOR Packets, perform the computation as well as the energy consumed in transmitting the TREG Packets (32 bytes) to their parents.

We compare the total energy consumed in the network using TREG with that for a simple data packet forwarding scheme. As can be seen from Fig. 6.8, the energy consumed in the network with data aggregation using TREG is less than that consumed without any data aggregation. It is also evident from the graph that with increase in the number of nodes in the network, the energy

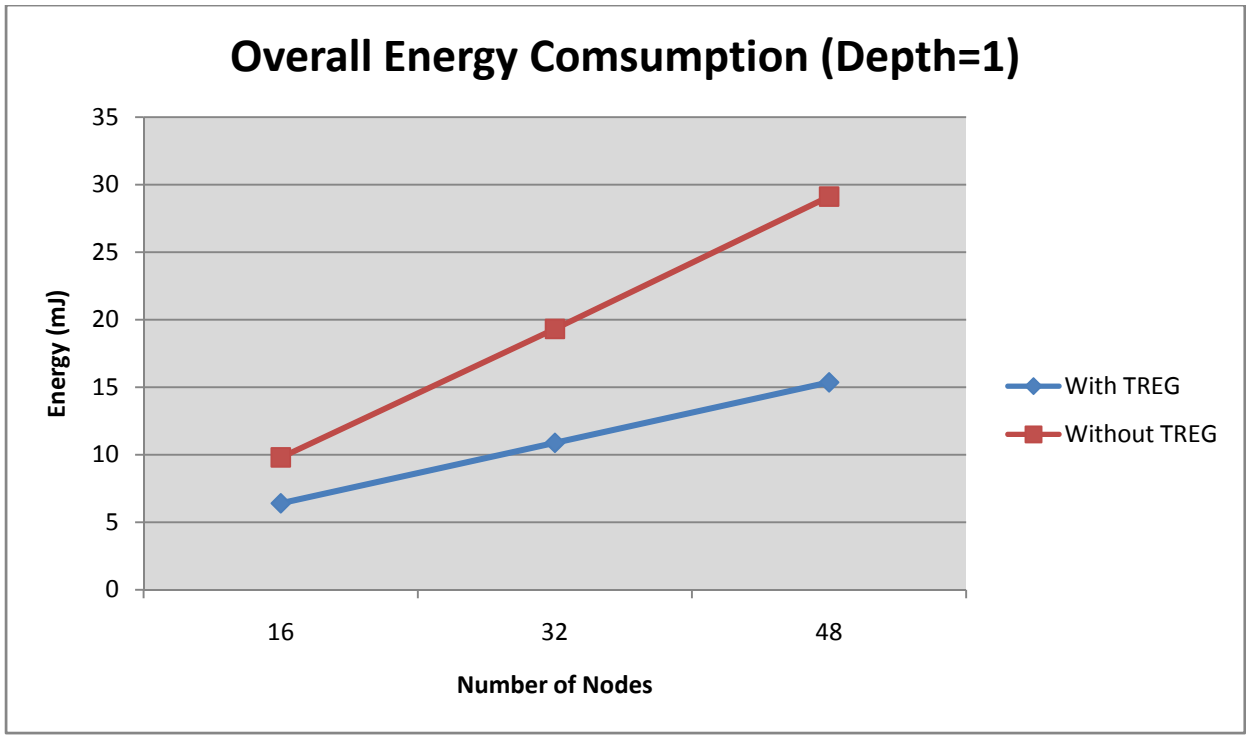

Figure 6.8: Variation of energy consumption with increasing number of nodes

consumed grows much more in the simple packet forwarding case than in the one using TREG. It can thus be seen that using a data aggregation scheme such as TREG, can indeed lead to increased lifetime of sensors, which is a fundamental challenge in WSNs.

6.5 Compression Ratio (CR)

The CR in our scheme is defined as the ratio of the sum over all edges of number of bytes transmitted on an edge after compression over the number of input bytes on an edge. Fig. 6.9 shows the variation of compression ratio with depth of the aggregation tree. The graph, as expected, suggests that the CR decreases with depth, since the degree of reduction is better at a larger depth. The average CR obtained in our experiments is 0.29.

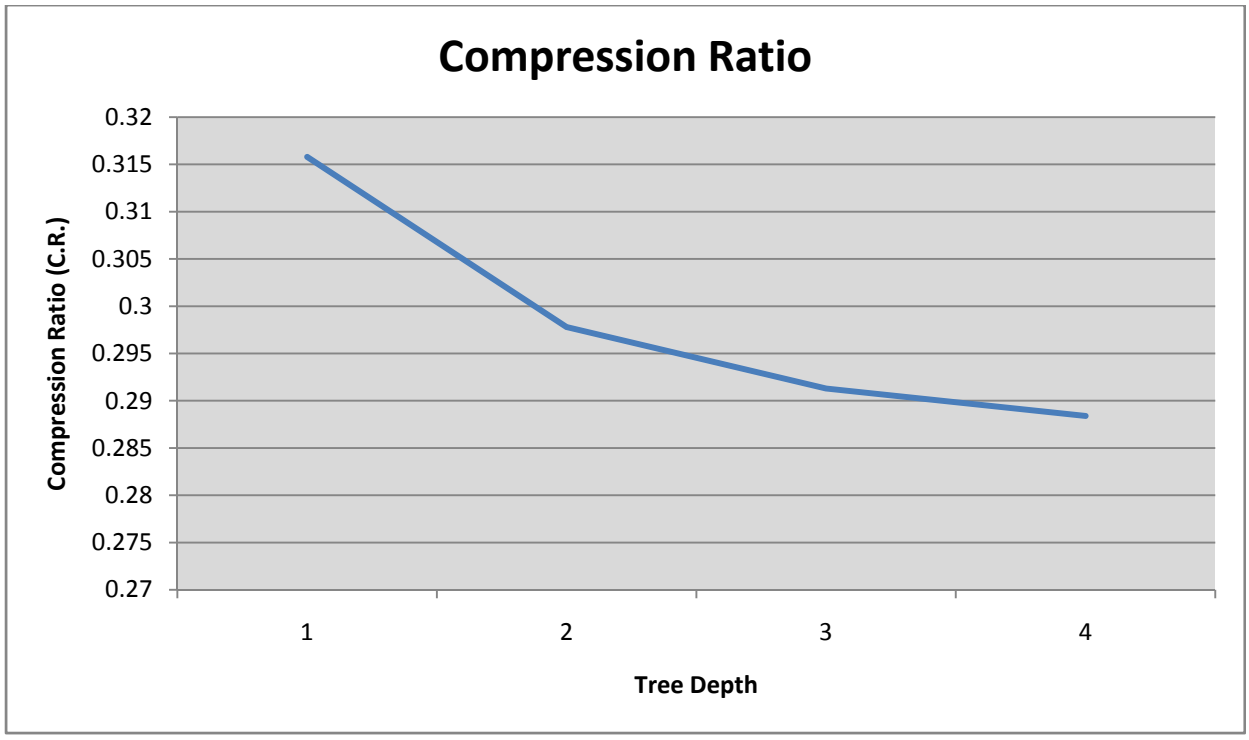

Figure 6.9: Compression ratio Vs Tree depth

6.6 Linear versus Quadratic Regression Polynomial

The linear polynomial used thus far in the experiments consists of four coefficients. We also wanted to study the effect of the use of a higher degree polynomial on the motes with respect to energy consumption as well as the performance of TREG. However, since the memory requirements of the program using a quadratic polynomial with nine coefficients exceeded the available memory on the motes itself, we could not proceed with the experiments. Instead we ran our program on a desktop personal computer in order to compare the results of regression obtained using a quadratic polynomial viz. those obtained using a linear one. In this section we compare the accuracy of the regression polynomials and their effect on the percentage error of the approximated values.

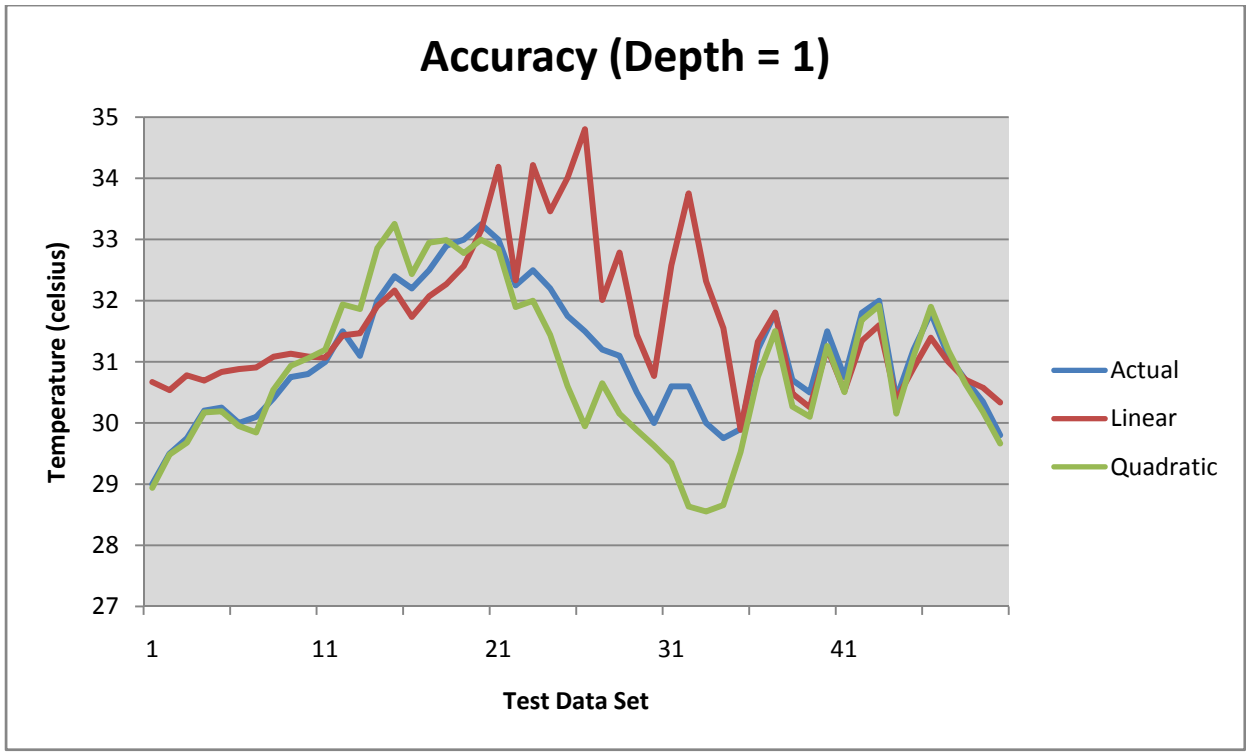

Figure 6.10: Accuracy using Linear (4 coefficients) and Quadratic (9 coefficients) polynomials.

Fig. 6.10 above shows the actual and approximated curves obtained by regenerating attribute values for the 50 random points selected as described earlier. It is clearly evident from the graph that the curve of approximated values using a quadratic polynomial follows the actual curve more closely than the one obtained using a linear polynomial. This implies that more coefficients are required in order to capture the correlation in the sample data set chosen. In general, higher the degree of correlation of the data set is easier it is to capture it with a polynomial of lower degree. The authors in [2] suggest that a polynomial equation of second degree (quadratic) is sufficient to capture the non-linearity of the measured data set accurately. Increasing the degree further results in no improvement in the accuracy (percentage error of 6.0% is reduced to 5.9%) but increases the computational complexity of the regression at each node and the overhead in sending the increased number of coefficients (27 for a cubic polynomial).

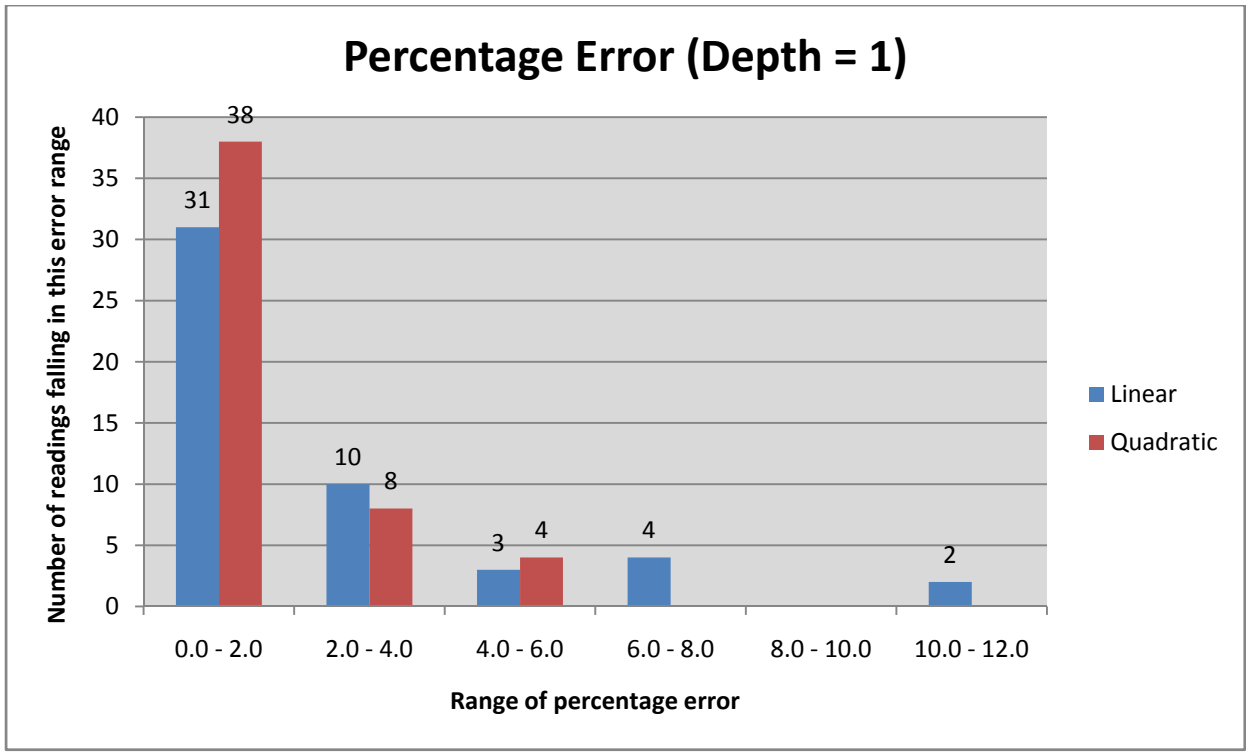

Figure 6.11: Percentage error using Linear (4 coefficients) and Quadratic (9 coefficients) polynomials.

The increase in the accuracy is also evident from the percentage error of the regenerated attribute values as shown in the Fig. 6.11. We can see that a greater number of samples now fall in the 0.0 to 2.0 % error range and the maximum percentage error is reduced as well. The maximum percentage error in the quadratic polynomial case was found to be 6.42% as compared to 10.49% in the linear case implying that the correlation in the data set was captured better by the higher degree polynomial.

A comparison of the energy consumption in the motes due to the use of the two polynomials should provide a better indication of the trade-off involved between increased accuracy and energy consumption. This would be feasible once the on-board memory available is large enough to hold our application and is thus left as future work.

The results presented thus far in this thesis have been based on experiments conducted on the more popular MICAz mote hardware platform from Crossbow. However, it has been observed that the program when ported to the Moteiv tmote-sky mote platform overcomes many of the limitations of the MICAz implementation.

The tmote-sky [24] is an ultra low power IEEE 802.15.4 compliant wireless sensor module manufactured by Moteiv (now Sentilla). Some of the key features of the platform are as follows:

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Interoperability with other IEEE 802.15.4 devices
- 8MHz 16 bit Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep (< 6μs)
- Hardware link-layer encryption and authentication
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector

The preliminary results obtained by running the TREG application on the tmote-sky motes indicate that the on board sensors are fully supported by TinyOS 2.x and the memory available is adequate to hold all the data structures required to perform the polynomial regression using a degree 2 polynomial. These two factors severely constrained the implementation on the MICAz as mentioned earlier in this thesis.

At the time of writing this thesis, the number of tmote-sky motes available was not sufficient to perform any experiments. Since, the tmote-sky platform opens up a plethora of possible experiments including experiments to evaluate the performance of the algorithm with respect to temporal correlation in addition to spatial correlation in a real world environment, it is best left as future work to ensure cogency and completeness.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

WSNs have recently become one of the most actively researched topics in both academia and industry. Researchers all over the world are conceiving new applications almost every day, from battle field surveillance to environmental monitoring. Most of these applications involve the transfer of large amounts data across the network towards the sink. SNs are small battery powered devices and typically deployed at places that are not easily accessible. Thus, once their battery runs out, these SNs have to be discarded and replaced with new ones – a very undesirable and expensive procedure. It's therefore imperative to maximize SN lifetimes. In-Network Processing or Data Aggregation is amongst the most popular techniques for improving SN lifetime. In this thesis, we have implemented TREG, one of many promising data aggregation schemes, on wireless sensor motes and evaluated its effectiveness using experimental results. TREG exploits spatial correlation in the sensed environmental data to reduce the amount of data transmitted up the tree towards the root. The approximating polynomial regression function also enables the root to compute the value of the physical parameter at any point in the network, without actually querying the network. This, in addition to the in-network processing, leads to much lower network traffic by way of reduced transmissions and re-transmissions, resulting in significant savings in the energy consumed by the SNs. System energy savings result in increased SN lifetimes, thereby bringing down the overall WSN maintenance costs. Our results as presented in the previous chapter essentially validate the simulation results provided in the previous work [2]. In addition, we have shown that the total energy consumption in a system

running the TREG application is much lower than that based on a simple multi hop data forwarding scheme.

7.2 Future Work

Our attempt at an implementation using a quadratic polynomial (9 coefficients) failed since the memory requirements of the program exceeded the available memory (4 KB) on the MICAz motes. We would like to leave this as future work when on board memory is sufficiently large enough to store our program. We would also like to test TREG with real environmental data for validating the effectiveness with respect to temporally correlated samples as well. This would be feasible once drivers for the MDA 300 sensorboard are available.

Also, as mentioned at the end of the previous chapter, the tmote-sky mote platform overcomes many of the limitations of the currently more popular MICAz mote hardware and hence opens up a plethora of possible future experiments to exhaustively evaluate the performance of the TREG application in a real world deployment.

Bibliography

[1] D.P.Agrawal and Q.A. Zeng, Introduction to Wireless and Mobile Systems, 436 pages, Brooks/Cole Publication, 2003

[2] T. Banerjee, K. Choudhury, and D. P. Agrawal, "*Tree based data aggregation in sensor networks using polynomial regression*," in *Proceedings of the 8th Ann. Conf. on Information Fusion*, July 2005.

[3] TinyOS [Online] www.tinyos.net

[4] J. Hill, R. Szewczyk, A.Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. "System architecture directions for networked sensors" in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

[5] MICAz Data Sheet [Online]
http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf

[6] NesC [Online] http://nescc.sourceforge.net/

[7] D.P. Agrawal, M. Lu, T.C. Keener, M. Dong, and V. Kumar, "Exploiting the use of wireless sensor networks for environmental monitoring," *Journal of the environmental management,* August 2004, pp 35-41

[8] CORIE [Online] http://www.ccalmer.ogi.edu/CORIE/

[9] ALERT [Online] http://www.alertsystems.org/

[10] E. Biagioni and K. Bridges, "The application of remote sensor technology to assist the recovery of rare and endangered species," in *Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications,* Vol 16, No. 3, August 2002, pp 315-324

[11] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler and J. Anderson, “Wireless Sensor Networks for Habitat Monitoring,” *Intel Research Laboratory,* Berkeley, Intel Corporation, {amm,dculler}@intel-research.net, EECS Department, University of California at Berkeley, {polastre, szewczyk, culler}@cs.berkeley.edu, College of the Atlantic, Bar Harbor, Maine jga@ecology.coa.edu

[12] C.D.M. Cordeiro and D.P. Agrawal, *Ad Hoc & Sensor Networks: Theory and Applications*, World Scientific Publishing Co. Pte. Ltd, 2006.

[13] B. Krishnamachari, D. Estrin and S. Wicker, "Modelling Data-Centric Routing in Wireless Sensor Networks," in *Proceedings of IEEE INFOCOM 2002*

[14] K. Sohrabi, J. Gao, V. Ailawadhi and G. J. Pottie, “Protocols for self-organization of a wireless sensor network,” in *Proceedings of IEEE Personal Communications,* 7(5), 16–27 (October 2000)

[15] C. Guestrin, P. Bodik, R. Thibaux, M. Paskin, and S. Madden. “Distributed regression: an efficient framework for modeling sensor network data”, in *Proc. IPSN, 2004.*

[16] M. Welsh and G. Mainland, “Programming sensor networks using abstract regions”, in *Proc. the First USENIX/ACM Symposium on Networked Systems Design and Implementation* (NSDI 2004), March 2004.

[17] O. Younis and S. Fahmy, “An Experimental Study of Routing and Data Aggregation in Sensor Networks”, in *IEEE Int.Conference on Mobile Ad Hoc and Sensor Systems (MASS),* 2005.

[18] E. Kuh, "An essay on Aggregation Theory and Practice, No. 43," *Working papers from Massachusetts Institute of Technology (MIT)*, Department of Economics

[19] MPR-MIB Series Users Manual [Online]

http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf

[20] C. Borgelt, "Intelligent Data Analysis," School of Computer Science Otto-von-Guericke-University of Magdeburg, Universit¨atsplatz 2, D-39106 Magdeburg, Germany.

[21] L. Fausett, *Numerical Methods: Algorithms and Applications*, 0130314005 (Hardback). Pearson Education: NJ, October 2002.

[22] Gaussian Elimination Example [Online] http://www.math-linux.com

[23] J.D. Finn, *A General Model for Multivariate Analysis,* Holt, Rinehart & Winston: NY, 1974. ISBN: 0-03-083239

[24] Tmote-Sky Data Sheet [Online] http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf