University of New Mexico
UNM Digital Repository

Electrical and Computer Engineering ETDs

Engineering ETDs

9-9-2007

Hierarchical aggregation and intelligent monitoring and control in fault-tolerant wireless sensor networks

Prasanna Sridhar

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Recommended Citation
Sridhar, Prasanna. "Hierarchical aggregation and intelligent monitoring and control in fault-tolerant wireless sensor networks." (2007). https://digitalrepository.unm.edu/ece_etds/243

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Prasanna Sridhar

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality
and form for publication on microfilm:

Approved by the Dissertation Committee:

, Chairperson

Dr. Mo Jamshidi

Dr. Asad M. Madni

Dr. Chouki Abdallah

Dr. Mahmoud Reda-Taha

Dr. Nader Vadiee

Accepted:

Dean, Graduate School

Date

Hierarchical Aggregation and Intelligent Monitoring and Control in Fault-Tolerant Wireless Sensor Networks

BY

PRASANNA SRIDHAR

B.E., Computer Science, Bangalore University, 2000
M. S., Computer Science, University of New Mexico, 2003

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico
Albuquerque, New Mexico

July, 2007

©2007, Prasanna Sridhar

DEDICATION

Dedicated to Lord Ganesha, my family, friends and well-wishers

ACKNOWLEDGMENTS

I heartily acknowledge Dr. Mo Jamshidi, my advisor and committee chair, for continuing to encourage, support, guide and mentor me in my research work and career. I am indebted to him for his help both in my professional and personal life.

I would like to extend my utmost and deepest gratitude to Dr. Asad M. Madni, my co-advisor and external expert, without him this thesis would not have been possible. The wealth of knowledge, guidance, and technical ideas that he has brought into this thesis is invaluable. His professional approach, mentoring and style will remain with me forever as I continue my career.

I would like to thank my other committee members Dr. Chaouki Abdallah, Dr. Mahmoud Reda-Taha, and Dr. Nader Vadiee for their valuable inputs for this thesis. I would also like to thank Dr. Thrishantha Nanayakkara for his suggestions and inputs. Gratitude is extended to the Electrical and Computer Engineering Department at the University of New Mexico and at the University of Texas San Antonio for their support.

Lastly, but most importantly, I thank my parents, family, friends, and colleagues, for their immense love and support.

Hierarchical Aggregation and Intelligent Monitoring and Control in Fault-Tolerant Wireless Sensor Networks

BY

PRASANNA SRIDHAR

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico
Albuquerque, New Mexico

July, 2007

Hierarchical Aggregation and Intelligent Monitoring and Control in Fault-Tolerant Wireless Sensor Networks

by

PRASANNA SRIDHAR

B.E., Computer Science and Engineering, Bangalore University, 2000

M.S., Computer Science, University of New Mexico, 2003

Ph.D., Engineering, University of New Mexico, 2007

ABSTRACT

The primary idea behind deploying sensor networks is to utilize the distributed sensing capability provided by tiny, low powered and low cost devices. Multiple sensing devices can be used cooperatively and collaboratively to capture events or monitor space more effectively than a single sensing device. The realm of applications envisioned for sensor networks is diverse including military, aerospace, industrial, commercial, environmental and health monitoring. Typical examples include: traffic monitoring of vehicles, cross-border infiltration-detection and assessment, military reconnaissance and surveillance, target tracking, habitat monitoring and structure monitoring, to name a few.

Most of the applications envisioned with sensor networks demand highly reliable, accurate and fault-tolerant data acquisition process. The integrity of data alone can have tremendous effects on the performance of any data acquisition system. Due to the low manufacturing cost, the sensors lend themselves to be deployed in large numbers with a high spatial distribution. Such a large deployment scheme often generates enormous amount of data that needs to be efficiently summarized and delivered for analysis and processing. In-network data compression, data aggregation/fusion, and decision propagation are some of the processes that deal with huge data issues. A hierarchical data aggregation scheme developed in this thesis is a highly effective and energy efficient means (by reducing communication packets) to deliver decision milestones to the end-user. The sensing devices are also prone to failure due to the inherent characteristics such as construction and deployment. It is thus necessary to devise a fault-tolerant mechanism with a low computation overhead to validate the integrity of the data obtained from the sensors. Moreover, a robust diagnostics and decision making process should aid in monitoring and control of critical parameters to efficiently manage the operational behavior of a deployed sensor network. Specifically, this research will focus on innovative approaches to deal with multi-variable multi-space problem domains (data integrity, energy-efficiency and fault-tolerant framework) in wireless sensor networks. We present three information-based methods for improving the performance (fault-tolerance and efficiency) of wireless sensor networks (WSNs). The first is a method for time varying weight adaptation in a mixture model for sensor data aggregation. The second technique applies fuzzy inference methods to solve a multi-criteria decision problem, specifically the efficient management of data collection in a WSN. The third method presented proposes the use of spatially variant weights to reduce the significance of sensor readings taken near the boundary of the sensor range, in order to minimize potential corruption of aggregated data. The solutions proposed in this thesis have practical implementation in developing power-aware software components for designing robust networks of sensing devices.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background – Sensors and Sensor Networks

The concept of sensing physical phenomena has been inspired from biological living creatures. Sensors have been in existence for few decades now and are being used in everyday life. Applications of sensors include automobiles, machines, aerospace, medicine, robotics, etc. With emerging technologies such as Microelectromechanical Systems (MEMS), sensors are being manufactured at low cost and at microscopic level. In most cases, these micro-sensors reach significantly higher speeds and higher sensitivity compared to macro-sensors. Such emerging manufacturing technologies, along with improvements in wireless communication and computation processes, sensor research has undergone a revolution. The traditional single sensor system is replaced by large array of tiny, self-powered sensors that can wirelessly communicate to the "outside" world. Large numbers of sensors may be integrated into systems to improve performance and lifetime, and decrease life-cycle costs. As with many technologies, defense and military applications have driven the research and development of sensor networks. The solution to use one expensive sensor to cover the whole area of interest is too risky and expensive, with the related false alarms and single point of failure. Having multiple sensors clearly alleviates the problem of single point of failure. For example, swarm of Unmanned Aerial Vehicles (UAVs) can provide better situation awareness than a single UAV used for reconnaissance and surveillance. Thus, traditional single sensing system is replaced by multiple spatially distributed sensors that not only sense but also process and communicate critical information and decision milestones. This has been facilitated by the integration of multiple sensors, processors, memory and RF communication devices onto a single board.

1.1.1 Glossary of Terms

a. *Sensor node*: A multi-sensor platform that houses variety of MEMS based sensors (such as temperature, humidity, accelerometer, light, pressure, etc) along with low power microprocessor and radio. Generally, these nodes are powered by battery.

b. *Ubiquitous computing*: Also known as pervasive computing or calm computing. It is a model of computing in which computer functions are integrated into everyday life, often in an invisible way.

c. *Data Aggregation:* Meaningful summary of the given data.

d. *Middleware:* Software that connects other software applications often to support complex distributed systems.

1.2 Problem Statement and Motivation

Sensor network applications often require minimal human intervention, thereby exhibiting autonomous behavior. Once deployed, these sensor nodes often form an ad-hoc network. The role of each sensor node is to acquire data samples from various sensors on-board and communicate the acquired data and/or summary statistics rather than raw data for further processing. For an autonomous system to operate normally, it is necessary to monitor it continuously or at predetermined time intervals. To monitor the integrity and performance degradation in a cost-effective way, it is first necessary to probe the integrity of *data* acquired from the sensors. Such probing methods are often termed as *system diagnostics*. Thus the capability of a data acquisition system must have a high degree of accuracy and efficiency in acquiring and interpreting data from multiple information sources.

This research work is motivated by several problems that are still persistent in the real-world network of sensors. Therefore, the primary aim of this research thesis is to provide innovative solutions to each problem domain – efficient data acquisition, large data processing, decision making and monitoring that all help in optimizing the performance of the given sensor network in real-world deployment scenarios. Most of the solution proposed herein is first of its kind in the area of sensor networks. In doing so, we hope that this research work will provide an entry point to a larger body of literature in sensor networks management.

1.3 Research Challenges and Contributions of this Thesis

1.3.1 Research Challenges

The research challenges identified in sensor networks fall into three broad categories - Sensing, computing and communication as shown figure 1.1. Sensor networks generally pose considerable technical problems in data processing, communication and sensor management [1-2]. We have identified *data processing* and *sensor network management* as the key challenges within this thesis and provide solution along with merits and demerits of existing solutions for practical implementation.

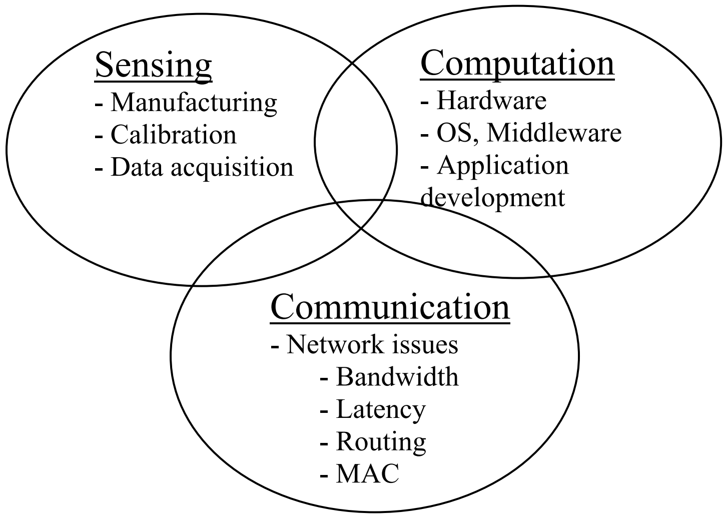


Figure 1.1 Research Areas Identified In Sensor Networks

Sensor networks must deal with resources – energy, bandwidth, processing power, etc. – that are changing dynamically. Given such dynamic situation, the sensor nodes need to operate autonomously. This requires research into issues such as number of nodes to keep network alive, proper size of network with redundant nodes, effective means to optimize on resources, increase network lifetime, etc. Such issues often fall into sensor network management.

Processing data from large number of sensors requires more resources (bandwidth, transmission power at intermediate nodes). More nodes on the other hand, results in better performance. Therefore, it is necessary to communicate as much data as possible but at the same time reduce the resource consumption. Since large data sets might also get corrupted during communication or acquisition, there needs to be ways to handle faulty data. Other data processing issues that are dependent on application are latency, reliability and completeness of the data.

1.3.2 Research Objectives

This subsection discusses the objectives of this research work. Following are the major objectives identified along with a brief problem description. These objectives are based on the data processing and sensor network management challenges.

1. Fault-Tolerant Data Acquisition and Fusion

While there are several automated data fusion approaches, handling faulty information contributing to fusion process still remains a challenging task. If the data from faulty sensors are accommodated in the fusion process, the resultant fused decision/data still remains faulty. This can pose a serious problem in situation awareness scenarios, such as false information about a target. With multiple sensors, there is an increasing risk involved in faulty information associated with the data.

2. System Monitoring

For systems such as networks of sensing devices, it is necessary to monitor continuously or at pre-determined time interval. Such monitoring process becomes complicated with the increase in system parameters. For example, generating a state-space model for a sensor node or network would be a hard task. In such situations, identifying, probing and tuning critical parameters are effective means to monitor and diagnose the network. Such diagnostics will often help to maintain or improve the integrity of the deployed sensor network.

3. Interpretation, Decision and Learning from Acquired Data

Data is everything. Data from the network provides important information on the state of the network as well as the environment being sensed. Interpreting data will help to diagnose the network for reliability and integrity. For example, one of the objectives of a sensor network with on-board batteries is to survive as long as possible and derive meaningful feature level information from the environment. Therefore, such networks are generally faced with the management of conflicting objectives such as conserving limited on-board energy and keeping the sensor awake to pick up stimuli from the environment. We have identified automated decision making with multiple objectives/criteria and learning to take such actions based on the varying situations.

1.3.3 Specific Contributions

The contributions of this research thesis are as follows:

1. Development of theory for meaningful aggregation of acquired data from sensors with tolerance to faults.
2. Development of middleware for implementing the theory proposed for data aggregation.
3. Theory and application of multi-criteria decision making in sensor networks.
4. Theory and application of approximate reasoning and rule-based approaches for sensor node integrity.
5. Network protocol design and implementation for message passing among sensor nodes, cluster-head and base-station.
6. Development of an extensive graphical user-interface (GUI) for interpreting and analyzing the acquired data from sensor network.
7. Implementation of reinforcement function or critic on a sensor node.

The presentation of material from conceptual ideas to focused research has resulted in following technical papers:

1. Sridhar, Prasanna, Madni, Asad M., Jamshidi, Mo, "Hierarchical Aggregation and

Intelligent Monitoring and Control in Fault-Tolerant Wireless Sensor Networks", *Inaugural Issue of the IEEE Systems Journal, to be published in 2007*

2. Sridhar, Prasanna, Madni, Asad M., Jamshidi, Mo, "Intelligent Multicriteria Decision Making in Robot Path Planning using Sensor Networks", *First IEEE Systems Conference, Hawaii, 2007*

3. Madni, Asad M., Sridhar, Prasanna, Jamshidi, Mo, "Fault-Tolerant Data Acquisition in Sensor Networks", IEEE *System of Systems Engineering Conference, San Antonio, 2007.*

4. Sridhar, Prasanna, Madni, Asad M., Jamshidi, Mo, "Intelligent Object-Tracking using Sensor Networks", *IEEE Co-sponsored Sensor Applications Symposium (SAS), San Diego, 2007.*

5. Sridhar, Prasanna, Madni, Asad M., Jamshidi, Mo, "Hierarchical Data Aggregation in Spatially Correlated Distributed Sensor Network", *IEEE Co-sponsored World Automation Congress, 2006*

6. Sridhar, Prasanna, Madni, Asad M., Jamshidi, Mo, "Intelligent Monitoring of Sensor Networks using Fuzzy Logic Based Control", *IEEE Conf. on Systems, Man and Cybernetics, 2006*

7. Azarnoush, Hamed, Horan, Ben, Sridhar Prasanna, Madni, Asad M., Jamshidi, Mo, "Towards Optimization of a Real-World Robotic-Sensor System of Systems", *IEEE Co-sponsored World Automation Congress, 2006*

8. Sridhar, Prasanna, "Optimal Node Density Estimation in Sensor Networks", *Workshop on Systems and Intelligent Control, October 2005*

9. Sridhar, Prasanna, Jamshidi, Mo, "Discrete-event modeling and simulation: Application to wireless sensor networks", *IEEE Conf. on Systems, Man and Cybernetics, 2004.*

1.3.4 Related Research Areas

The research area in sensor networks is relatively broad and interdisciplinary; predominantly dealing with computation and communication. Most of the challenges and bottlenecks in sensor network research deal with energy efficient design and development of software and/or hardware components [3]. The focus of this research work is to present novel ideas that have practical implementation in developing power-aware software components for designing robust networks of sensing devices.

We identify at least four major research areas – data acquisition and information processing, fault-tolerant algorithms, decision making, and diagnostics that directly relate to the proposed approach presented in this paper. Each chapter will discuss in detail the existing literature and how our method deviates or improves on the existing work.

1.3.5 Preliminary Assumptions

We consider multiple heterogeneous sensors (such as temperature, pressure, humidity, etc.) on a single sensor board. We consider static sensors for our experimentation assuming that the drift is very negligible. Such a multi-sensor platform, often referred to as a sensor node, has limited computation and communication capabilities. These nodes when densely deployed in a region of interest, offer a spatially distributed sensing capability. The resultant network of these nodes is often clustered in order to efficiently manage and implement information routing, data aggregation, event localization, etc., – a divide and conquer strategy. Several different clustering algorithms for sensor networks have been proposed [4-6]. An important feature of clustering is that it enables a hierarchical organization of sensor nodes, with different functional capabilities at each level. Sensing can be done at the lowest level of hierarchy and decision making at higher levels, such as at a cluster-head level, gateway level or base-station level. A base-station is assumed to have higher processing and communication capabilities compared to sensor nodes.

A cluster-head can represent the information and operational characteristic for a cluster of sensor nodes. By information characteristic we mean that, information generated from several sensor nodes can be fused at the cluster-head to obtain an aggregated data or a decision milestone. By operational characteristic we mean that the operation of each sensor in that particular cluster is validated by examining the quality or integrity of data. We assume that the sensor nodes are grouped into different clusters.

1.4 Summary of Chapters

This section will present the preview of all the chapters that follow. In total there are seven chapters, the first being the introduction. The second chapter deals with efficient data acquisition and processing in sensor networks. The problem of handling large data sets, simple yet effective fault-tolerant data acquisition methods and decision propagation techniques in sensor networks are introduced in this chapter. Whereas chapter 2 discusses data integrity in sensor networks, chapter 3 outlines innovative methods to ensure network integrity by intelligently monitoring and diagnosing the given network of sensors. Chapter 4 introduces the concept of multi-criteria decision making to further enhance the integrity of the network. Learning and optimization techniques specifically related to real-world sensor networks are introduced in chapter 5. Chapter 6 provides the design, implementation and results of the concepts portrayed in previous chapters on commercially available hardware platforms. Chapter 7 concludes by opening the stage to possible future work and extensions to the ideas and algorithms proposed in this research work. Each chapter concludes with summary with any related work being discussed within the context of the chapter. Moreover, any simulation results to validate the approach illustrated are given in the same chapter.

CHAPTER 2

DATA ACQUISITION AND PROCESSING

Data acquisition is simply the process of acquiring raw data from different sources of interest for data analysis and processing. In large distributed systems, there is an enormous amount of data delivered to the central processing unit. Particularly, in sensor networks, due to the low manufacturing cost, the sensors lend themselves to be deployed in large numbers with a high spatial distribution, generating huge data that needs to be *efficiently* delivered for data processing. Efficient data acquisition process for information processing is the foremost priority for analyzing or diagnosing the status of a deployed sensor network. Efficiency is the key issue in terms of energy consumption, fault-tolerance and relevancy of the sensor data. In this chapter, we propose an innovative way to acquire fault-tolerant data from distributed embedded sensors and then summarize the acquired data hierarchically in an energy efficient manner to optimize the performance of the data acquisition and delivery process.

2.1 Fault-Tolerant Techniques in Sensor Networks

Fault tolerant techniques have been studied in the field of computers for over a half century now. Such techniques are either specific to a given application or have a desired reference for comparing the output. The principle idea behind fault tolerance is the system's ability to perform or operate correctly even in the presence of faults. The problem of fault identification and isolation is generally a hard task in sensor networks due to the very nature of their construction and deployment. In this chapter, we will relate our proposed approach of fault-tolerance in sensor networks to some of the already existing methods sensors and embedded systems. Built-in Self Test (BIST) or Continuous Built-In Test (CBIT) has been studied extensively for combinational and sequential logic, memories and other embedded logic blocks. BIST technique involves embedding additional hardware logic which can be used to test the operation of the primary circuit logic [7]. Few researches have addressed the concept of software based built-in test [8]. In [9], a fault-tolerant approach for sensor network is proposed by using back-up sensors for faulty ones. Elnahrawy et al. [10] proposed a Bayesian approach to reduce noise and uncertainty in sensor networks. This, however, assumes prior knowledge of true sensor reading. Hereford [11], proposed an Evolvable Hardware (EHW) design to reprogram the circuit in case of any faults occurring in the sensors. In order to detect faults in the sensors, the paper proposed to use spatial correlation and Kalman Filter (to estimate actual output).

Thus, most of the fault-tolerant techniques in sensor networks [8-11] either assume prior knowledge of the sensor reading or have a desired reference to compute the error of the sensor reading. Our proposed approach in this chapter is similar to the one proposed in [11], however, we use weighted function to reduce the “contribution” of faulty sensors instead of reprogramming the circuit. The advantage of our proposed fault tolerant mechanism is that, in general, it does not rely on the sensors to be geographically deployed close to each other. The geographic proximity of sensors can be used to complement the proposed fault identification process.

2.2 Fault-Tolerant Data Acquisition

The integrity of data has tremendous effects on the performance of any data acquisition system. Noise and other disturbances can often degrade the information or data acquired from these systems. Devising a fault-tolerant mechanism in wireless sensor networks is very important due to the construction and deployment characteristics of these low powered sensing devices. Moreover, due to the low computation and communication capabilities of the sensor nodes, the fault-tolerant mechanism should have a very low computation overhead.

2.2.1 Built-In Test Method

Each sensor within a node is assumed to work within a ***usable threshold window [min,max]***. A built-in test is said to have passed if the sensor reading r is within this window. That is, reading *r* should follow the equation $min<r<max$, where *min* and *max* are chosen appropriately for a given sensor and given application, which constitutes the operational behavior of the sensor. Every sensor is guaranteed to work "correctly" within a given operating range specified by the manufacturer. For example, a manufacturer could specify an operating range for a temperature sensor as -25 to 125^{o}C. Similarly, a chemical sensor (such as carbon monoxide sensor) can have an operating range from 0 to 500 ppm (parts per million). We call this operating range as ***guaranteed window***. This window is usually obtained from the sensor manufacturer. The usable threshold window [*min*,*max*] will incorporate the guaranteed window for a given sensor. That is, *min* of the usable threshold window will be lesser than the minimum range of operation defined by

the guaranteed window and *max* will be greater than the maximum operating range of the sensor. For example, we can define -40 to +150^{o}C as usable threshold window for a temperature sensor that is guaranteed to sense temperature within the range -25 to +125^{o}C with a specified accuracy. The sensor might still work outside this guaranteed window, however, with a much lesser accuracy. By using the concept of an added usable window versus a guaranteed window alone, we are trading off the performance optimization of the individual sensor versus optimizing (maximizing) the performance of the sensor network. In order to uniquely identify the sensor, each cluster head assigns even numbered binary code for each of the sensors within its cluster. A binary addition of the unique binary code at the cluster-head is used to exactly determine the faulty sensor(s). For example, consider four sensors in one of the cluster. If they are assigned binary code of 0001, 0010, 0100, 1000, then cluster-head simply computes a bit parity check. In case one of the sensors has failed the built-in test, it resets the binary code before sending it. By checking the parity of the binary sum, we can uniquely identify the sensor which has failed the built-in test. A possible message structure for implementing such protocol is as shown in figure 2.1.

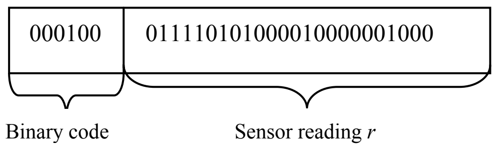


Figure 2.1. Message Structure

The major limitation of the above mentioned approach is the way the cluster head differentiates a common node that belongs to two or more clusters. If we have a closed overlapping clustering, few nodes belong to two or more clusters. Consider a scenario (as show in figure 2.2) where a sensor node has same node identification 0010 and belongs to two clusters. The checksum for parity check will be wrong in this case, since there are two nodes which transmit 0010 as a unique binary code.

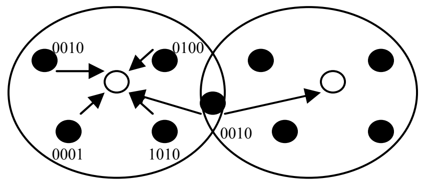


Figure 2.2 Overlapping Clustering Zone

Also, the number of bits used increases with the increase in the number of nodes in the cluster. The problem of time synchronization at the summing node could also pose a problem. In order to resolve these problems, we use node identification as integers. Operating system such as TinyOS [12] running on sensor nodes often allows users to program the nodes with identification numbers (node-id's). The modified message structure is as show in figure 2.3.

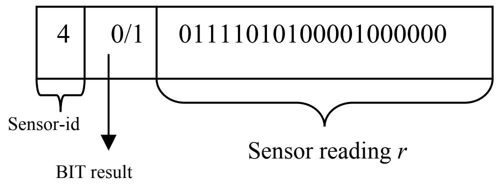


Figure 2.3 Modified Message Structure

We consider two different test methods to utilize the windowing effect to detect faulty behavior of a sensor.

1. ***Boolean Test*:** The built-in test can be a simple *pass* or *fail* test for each sensor on a

sensor node. Such tests can often be run on a sensor node since each node offers limited computation. The base station sends out a beacon message, possibly carrying a query to the cluster heads. The cluster heads in turn send out their respective beacon messages to all the nodes within their cluster. The node, when interrogated, transmits an encoded message that contains sensed information r for each of the sensors on-board. This hierarchical structure distributes the computation burden of decoding the messages from the sensors on to local cluster heads.

Two possible scenarios can be considered for the sensor to respond to such message. One method is that the sensor does not transmit any data if it fails the built-in test, thereby conserving communication cost and bandwidth of the network. However, this does not guarantee that the sensor has actually received a query. Packets might have been dropped due to network congestion. In order to alleviate this problem, our second approach is to send the result of the built-in test along with the sensor-id (node-id) in the header of the message, as shown in figure 2.3.

Although the above mentioned approach provides a simple solution to track faulty sensors, it does not capture the graceful degradation of a given sensor or a sensor node. For example, consider a scenario where a sensor reading r for a given sensor node, is *closer* to either *min* or *max* of the usable threshold window. For this situation, the sensor passes the built-test, even though its performance is degrading. One might argue that the sensor reading is still correct. For example, a temperature sensor used in environmental monitoring can read a sudden drastically high value in case of fire. This can cause the reading r to approach *max* within the defined window. However, the designer would have ensured that the operational behavior of the sensor is captured in the built-in test. There is thus an important aspect in designing such a built-in test, to choose the window appropriately for a given application and a given sensor.

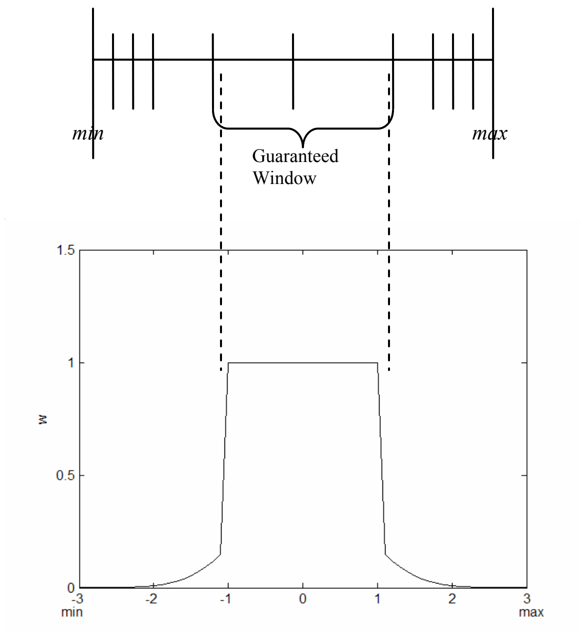


Figure 2.4 Weighted Method as Built-In Test

2. ***Weighted Method*:** A robust built-in test mechanism is to determine the performance degradation of a sensor based on how close the reading *r* gets to *min* or *max* of the usable window. This will help the decision making process to evaluate the likelihood of the sensor failure. It should be noted that, as a preventive measure for sensor failure (or a complete node failure), redundant sensors (or nodes) can be deployed in the region. This is, however, an expensive solution. The idea behind this second method is to assign a weighting factor to each of the sensors in a sensor node based on the reading *r*. Depending on how close the reading is to the usable window boundaries, the weight can be adaptively decreased. As described earlier, each sensor can operate within a

guaranteed window and the weight is set to 1, if the output of the sensor is within this window, i.e., w_{ij}= 1, where *i =(1..m)* signifies the sensor node number and *j= (1..n)* signifies the sensors on-board the *i-th* sensor node. If the sensor reading goes beyond the guaranteed window and approaches the usable window boundary, the weights are decreased. A simple methodology is to use a bell-function as shown in figure 2.4, where the weight exponentially decreases as the reading deviates from the guaranteed window. Since such a method imposes a heavy computation burden on the sensor node, the built-in test is performed for each sensor in a given node on the cluster-head or at higher level in the cluster hierarchy.

2.3 Protocol Design for Weighted Method

For the weighted method, we appropriately change the message structure as shown in figure 2.5. The message header now contains the sensor-id (or node-id), a *difference value* indicating how far the sensor output has deviated from the guaranteed window threshold. In order to realize such a design, we consider a sample working scenario with three sensors and a parent node. When the parent node interrogates the three sensors, each sensor replies with a message carrying its identification, a computed difference value and sensed value *r*. We assume that the parent node has the knowledge of *guaranteed window and usable window* for each of the sensors. Upon receiving the sensor reading, the parent node computes the *difference value* and compares it with the received *difference value*. This is similar to checksum. This will ensure that the message was correctly received without any communication errors. Once this test is passed, an appropriate weighting factor is assigned based on how far the value has deviated from the guaranteed working window.

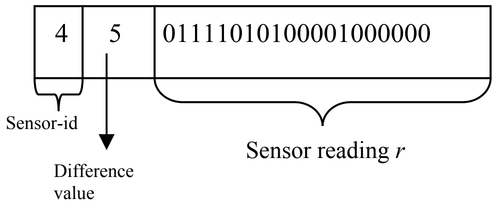


Figure 2.5 Modified Message Structure for Weighted Method

Algorithmic Design

Input: define *min* and *max* as threshold window. α and β as guaranteed window.

Output: w_{ij}, weighting factor for each sensor within the given cluster. $Pr_i(j)$, likelihood of failure of senor *j* in the given sensor node *i*. Final sensor reading r_{ij}

```
for each node i = 1 to m
   for each sensor j = 1 to n
     Initialize min, max, α, β for the given application
     set $w_{ij}$= 1
  end for
end for
for each node i = 1 to m
   for each sensor j = 1 to n
   if  $\alpha<r_{ij}<\beta$ then
                 set $w_{ij}$= 1
        else
                 set $w_{ij} = e^{-(r_{ij}/\varepsilon)^2}$ // where ε is chosen such that 0<w<1
                 $Pr_i(j) = 1- w_{ij}$
        end if
     $r_{ij} = w_{ij} \times r_{ij}$
    end for
end for
```

The computational complexity of the above mentioned algorithm is *O(m*n)*. However, *n*, number of sensors on-board a sensor node is usually small (<= 4). So the overall complexity is *O(k*m),* where *k* is the number of clusters and *m* is the number of sensor nodes within each cluster.

2.4 Limitation of Built-In Test Methods

Both Boolean Test and Weighted Method can prove to be useful in determining sensors that are operating outside their specified guaranteed window. In the case of Boolean Test, a sensor is said to have failed if the sensor reading is outside the guaranteed window. In Weighted Method, if the sensor reading is outside the guaranteed window, then an appropriate weighting factor is applied to the sensor reading in order to compensate for the degradation in accuracy of the sensor. It should be noted, however, that under certain circumstances, even when the sensor is working within a specified guaranteed window, the sensor precision and accuracy might be compromised due to the presence of external noise, environmental disturbances, etc. For example, consider a scenario, where a temperature sensor is transmitting erroneous/noisy data (say, temperature as -1^{o}C instead of +10^{o}C). However, if the sensor reading is still within the guaranteed window (-20^{o}C to +120^{o}C) then sensor is still deemed to be working good according to the Built-in Tests, even though in reality it is reporting a faulty reading.

2.5 Approaches to Support Built-In Tests

In situations where sensor linearity and sensor reading are critical, a redundancy feature helps to determine the accuracy of the sensor reading. Redundant nodes can be deployed in order to validate the true sensor reading. Applications involving sensor networks require dense deployment of sensors [12]. The sensors deployed in large numbers are ***spatially correlated*** within the region of events, that is, the sensor *i* reads the same event value (with minimal variation) as the neighboring *k* sensors which are closely deployed. Validation mechanism can be performed as follows. Consider three sensor readings a,b, and c from three redundant sensors. Two sensor readings are averaged ((a+b)/2, (a+c)/2, and (b+c)/2) at each predetermined time interval. The actual reading is set to the value at which the majority of the three averaged values agree upon – a *majority voting principle*. This helps to eliminate the faulty sensor in the group of the three sensors under consideration.

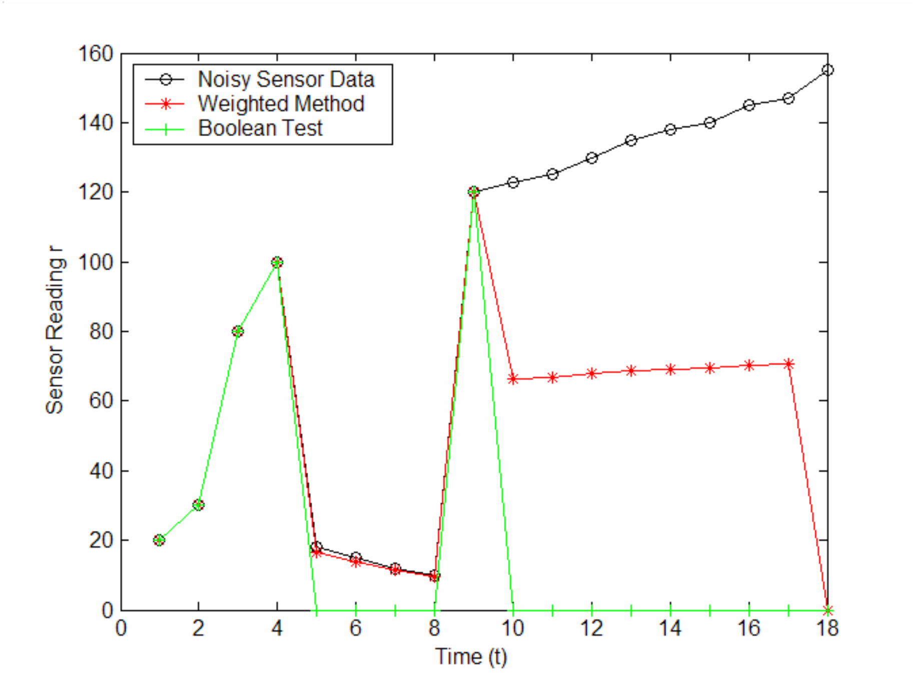
Depending on the type of sensor and the application, there might be a high degree of correlation between each consecutive sensor reading over time. This suggests the sensor readings are ***temporally correlated***. Consider a temperature sensor reporting a steady increase or decrease in the temperature over time. The readings are thus correlated over time. If there is a drastic change (large fluctuations) in the temperature over a small period of time, there is a high likelihood that the sensor is faulty. However, temporal correlation does not guarantee necessary and sufficient condition to claim that the sensor has failed. It vastly depends on the physical phenomenon being sensed.

If there is a drastic change in the reading of a given sensor, either due to environmental effects or sensor failure, then an interrogation signal can be sent to the sensor (with its associated signal conditioning circuit) for validating its operation. Interrogation could be in the form of a specific code, upon receiving which, the sensor responds with a unique code that identifies whether the sensor has failed or not. This interrogation can be done periodically or when there is an unusual reading reported by the sensor.

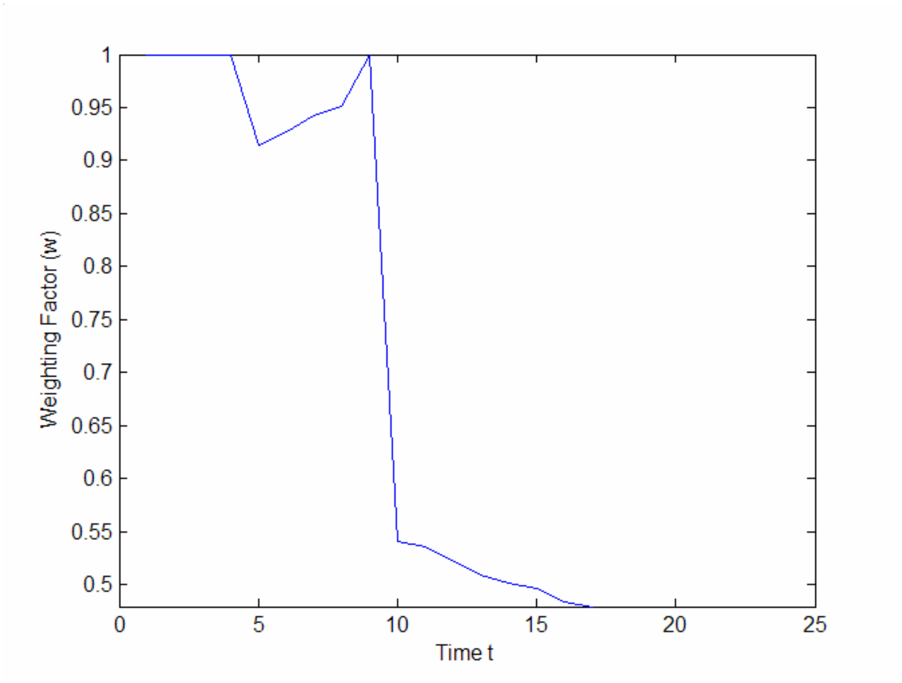
2.6 Simulation Benchmarking

We experiment both Boolean Test and Weighted Method for a temperature sensor using simulated data set. Guaranteed window is set to [+20^{o}C, +120^{o}C] and usable window to [+10^{o}C, +150^{o}C].

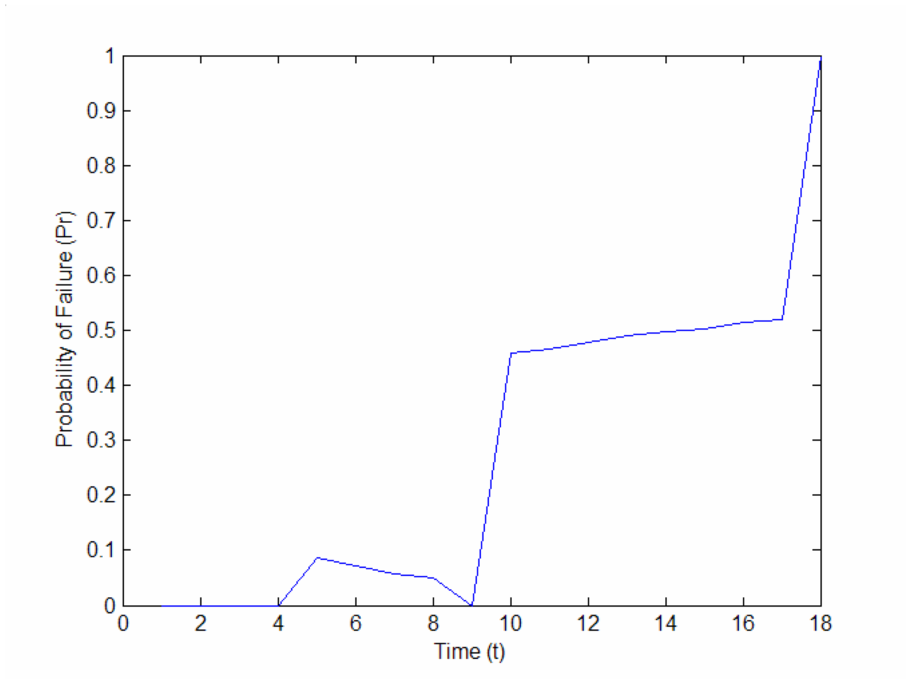
Discussion: Figure 2.6 (a) shows simulated sensor data reading and the corresponding compensated reading obtained from Boolean Test and Weighted method. In the Weighted method, when the temperature reading is within the guaranteed window, the weighting factor is set to 1. As the temperature crosses the limits of the guaranteed window and approaches the limits of usable window, weighting factor is adaptively decreased (as shown in figure 2.6(b)) and the corresponding sensor reading also decreases (figure 2.6 (a)). An interesting factor to analyze is that the slope of weighting factor (w), as shown in figure 2.6 (b), helps in failure prediction of the given sensor. However, the slope decrease could also be a result of sudden increase or decrease in the sensor reading. Figure 2.6 (c) shows the how likely the sensor will fail over time indicating the performance degradation of the given sensor. As seen from the above results, at time interval 8-9, the sensor reading jumps from very low (+10^{o}C) to a high value (+120^{o}C). There is thus an uncertainty as to whether the sensor has actually recorded a high event or has failed. This uncertainty is well demonstrated in probability of failure (refer figure 2.6c).



(a) Comparison of Built-In Test Methods



(b) Weighting Factor



(c) Performance Degradation of the Sensor

Figure 2.6 Experimental Benchmarking

2.7 Effect of Built-in Test on Data Aggregation

Faulty data acquisition can have effects on decision making and sensor data fusion [13-15]. In the second benchmarking scenario, we consider three homogenous sensors (say all temperature sensors) with guaranteed window set to [+20^{o}C, +120^{o}C] and usable window to [+10^{o}C, +150^{o}C]. We aggregate the data obtained from all the three sensors and compare the actual aggregated value with the aggregated value obtained from Weighed Method. As the temperature reading from any one of the sensors approaches the usable window, the corresponding sensor value is decreased by an appropriate weighting factor. Hence, the aggregated value obtained from the Weighted Method decreases, thus indicating the decrease in the "contribution" of the faulty sensor to the fusion (aggregation) process. This scenario is demonstrated in figure 2.7. We see that sensor-1 consistently reports a high temperature value (+180^{o}C) outside the usable window. Reading from sensor-2 gradually approaches the usable window, thereby suggesting that there degradation of the sensor. Our proposed Weighed Method incorporates the degradation of sensor-2 and failure of sensor-1 during the aggregation process, and thus has a lower aggregated value compared to aggregating the actual sensor readings.

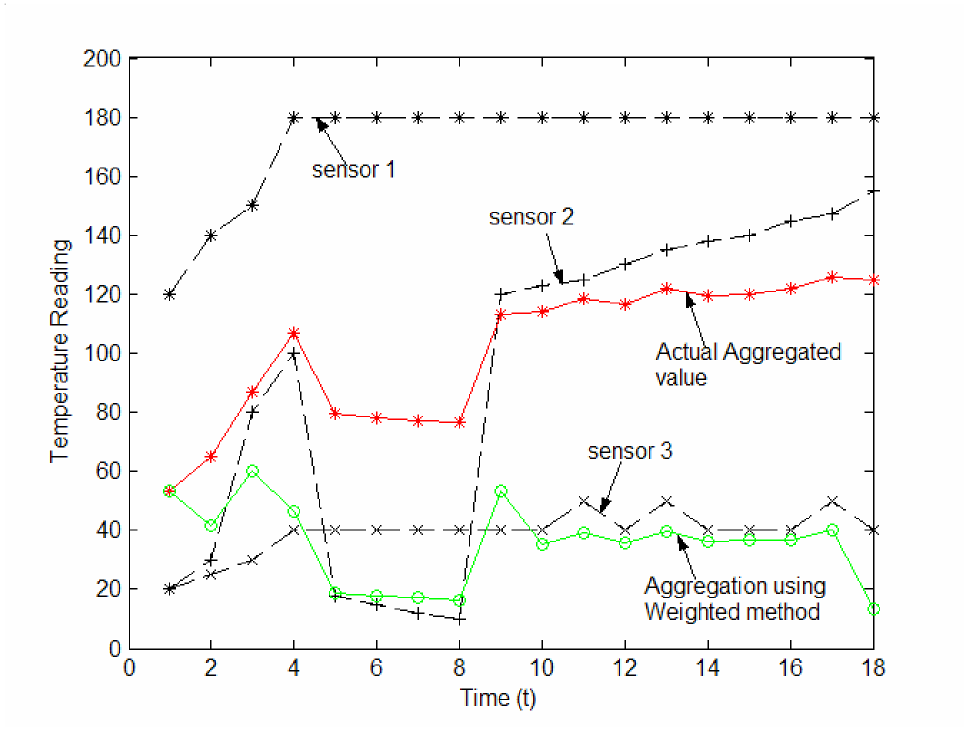


Figure 2.7 Aggregated Data and Weighted Method

The built-in test methods discussed help to answer the likelihood of component failure in any fault-tolerant design technique. Although built-in test methods such as Boolean test do not provide a fool-proof mechanism to validate the accuracy of a given sensor, they provide a mechanism to capture any intermittent faults. Weighted Method helps achieve failure prediction (or likelihood to failure) and provides gradual performance degradation between usable and guaranteed windows, thereby extending the performance characteristics of a given sensor in a multi-sensor network environment.

In case of criticality, having redundant nodes and performing built-in test provides a robust feature in the design for fault tolerance. We focus on the summarization and in-network aggregation of data in order to achieve energy efficient and cost effective scheme for optimizing data transmission. Specifically, we use the concept of ***spatial correlation*** of the distributed sensors to aggregate the "built-in tested" (fault-tolerant) data.

Multi-sensor data aggregation is an important application in data acquisition systems with low communication power. Parallel fused data from multiple sensors can represent decision milestones which will incur less communication cost than serially processing raw data acquired by individual sensors. It is an intractable problem to actually detect if a sensor is faulty by looking at the raw data acquired from the sensors [16]. However, because of faulty sensors, the fused data will deviate from the actual physical value being sensed. In order to reduce the impact of faulty information prior to fusing, we propose a novel method to aggregate the data from the distributed sensors.

We will first identify different aggregation operators and aggregation process. We then analyze the necessity and impact of data aggregation in sensor networks. Following which we will propose a methodology for aggregating data and finally discuss other competing power-aware data transmission techniques.

2.8 Common Aggregation Operators and Aggregation Process

Definition: Data aggregation is a process in which data or information from different sources is expressed in a summary form.

Data aggregation and data fusion are often interrelated and interchangeable. The importance of data aggregation arises from the fact that there is need for reducing redundant data and number of transmissions (network packets) in sensor networks. Data fusion is a broad area which could include aggregation as a sub-process and focuses on information rather than data with the use of several interdisciplinary techniques such as signal processing, statistical analysis, machine learning, and probability. Reference [17] including references therein, provide detailed information on data fusion architectures and methods.

Common aggregation operators found in literature are *max, min, median, quasi-arithmetic mean, weighted min, weighted max, weighted average* and *ordered weighted average* [18]. Although most of these aggregation methods offer reduction in data, they generally incur data loss and do not represent the actual physical phenomenon being sensed by the sensors. For example, *max* and *min* operators do not perform well when the standard deviation of the given data set is large. Operators such as *quasi-arithmetic means* are not stable under positive linear transformation [19], i.e., it does not satisfy the equation:

$$H(\alpha x_1+t, \alpha x_2+t, \alpha x_3+t\ldots, \alpha x_n+t) = \alpha\, H(x_1, x_2, x_3 \ldots, x_n) + t \qquad (1)$$

where H is the aggregation operator.

However, with aggregation process such as *weighted average*, the user can set weights and thus can control the aggregation process. In fact, more advanced aggregation operators such as Choquet and Sugeno Integrals [20] are special cases of weighted average method. Since weighted average is stable and not computationally intensive, it is very well suited for data aggregation in sensor networks.

Data aggregation process can be either *flat* or *hierarchical*. In a flat structure, all the sensory information is fused to produce a global estimate of the sensor data. However, this fusion or aggregation method has higher computation overhead on the aggregation or fusion node. In a hierarchical structure, sensor information is fused in each cluster to produce a local estimate which is then fused to obtain a global estimate of the sensed information. Several fusion steps are needed in each cluster; however, each of these local estimates can be performed in parallel as shown conceptually in figure 2.8. Weighted adaptation can be easily managed resulting in more reliable information from each sensor/cluster heads.

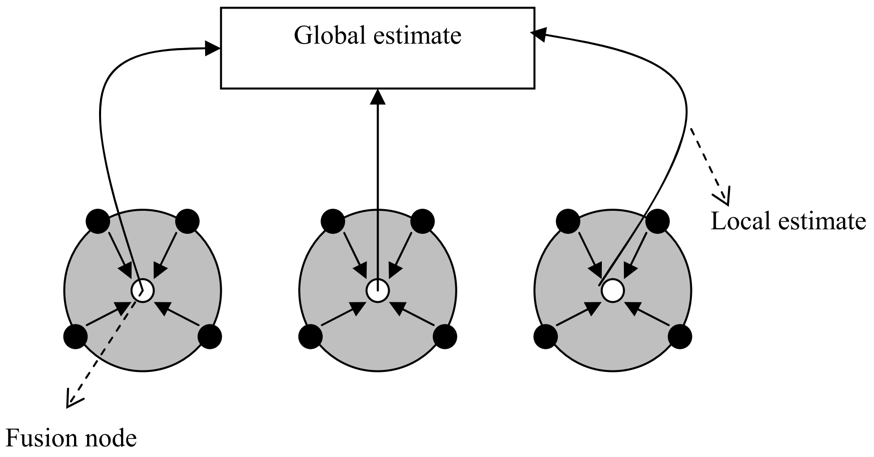


Figure 2.8 Hierarchical Data Aggregation from Difference Sources

2.9 Importance of Data Aggregation in Sensor Networks

We stress on the issue of energy conservation as the major factor while designing computational or communication intensive operations in sensor networks. Most of the energy is consumed in communication – transmitting sensed information from sensors to the gateway or base station for information processing.

Sensor networks are primarily used where focus is on aggregated query. For example, a query such as "what is the chemical concentration in that *area*?" is common compared to a query such as "what is the chemical concentration reported by a single sensor?" Such queries causes several sensors to report information from the area of interest causing a large flow of data from distributed sensing devices thereby causing increased communication cost, congestion and high battery usage. Therefore, there is a very high need to summarize information from different sensors. Such in-network aggregation helps in reducing redundant information, minimizing number of transmission (or packets) and thus conserving energy [21].

With in-network data aggregation, there is always an energy-latency tradeoff. Data aggregation incurs end-to-end latency in data delivery to the processing station. However, by reducing the number of transmissions, we achieve better performance optimization and increased lifetime through aggregation.

2.10 Sensor Network Protocols for Data Aggregation

An important issue for data aggregation in sensor network is time synchronization. To this end, several protocols are developed to reduce communication cost and also achieve synchronization during the aggregation process. Directed diffusion [22], SPIN [23], TAG [24], Time Synchronization in Sensor networks [25] and *cascading timeouts* [26] protocols have been proposed to achieve either explicit time synchronization or simple timeout to guarantee "data freshness" [26-27]. The timeout mechanism best suits our aggregation process.

2.11 Proposed Aggregation Mechanism

Our proposed algorithm for data aggregation can be broadly divided into two phases. The first phase is the fault-tolerant data acquisition process using test methods described in Section 2.2.1. The second phase of our approach is how well the acquired data can be aggregated inducing more fault-tolerance before transmitting data to the command center.

We use the concept of *hierarchical aggregation scheme* with *weighted average* aggregation operator for information fusion. Our innovation lies in the fact on how we dynamically update the weights during the aggregation process based on the spatial event correlation and likelihood of sensor failure.

Consider three overlapping sensing regions. The region of interest is the aggregated data obtained around the region of the intersection of these sensing regions. For a large deployment scenario, these sensing regions can be extended to cluster regions. The idea here is to parallel process rather than serial process data from each sensor node. Data aggregation can be either flat or hierarchical. In a flat structure, all the sensor information is fused to produce a global estimate of the sensor data. This fusion or aggregation method has higher computation overhead on the aggregation or fusion node. In a hierarchical structure, sensor information is fused in each cluster to produce a local estimate which is then fused to obtain a global estimate of the sensed information. Several fusion steps are needed in each cluster, however, each of these local estimates can be done in parallel. Weighted adaptation can be easily managed resulting in more reliable information from each sensor/cluster heads.

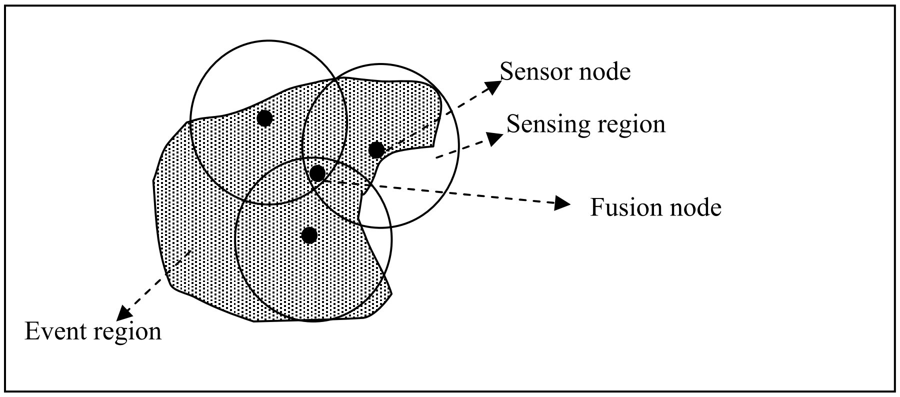


Figure 2.9 Event Region Data Aggregation with Three Sensor Nodes

In Figure 2.9, we investigate our weighted average data aggregation method. Each sensor node has a ***weighting factor*** at any instance of time *t,* given by $w_i(t)$. In the event of sensor failure, the proposed algorithm adaptively decreases the weight for sensors which have failed or demonstrate likelihood to fail. At the same time, weighting factors for the neighboring sensor nodes is increased. Hence, every reading from each sensor is weighted at each predetermined time interval *t*, and weight updates are computed as follows:

$$w_i(t+1) = w_i(t) \pm \Delta w_i(t) \qquad (2)$$

A fusion node (cluster-head or base station) can simply query the nodes for sensor reading in the event region. Based on a specific timeout, the fusion node performs weighted average aggregation based on the data it has received currently from the sensors.

In traditional neural networks, the change in weights $\Delta w_i(t)$ is a function of the error estimate, which is based on the difference between the expected reading and the actual reading. However, in sensor nodes, we do not know the expected or desired reading *a priori*. In order to estimate $\Delta w_i(t)$, we use the concept of ***spatial correlation***.

Sensor i reads the same event value (with minimal variation) as the neighboring k sensors which are closely deployed. In order to estimate $\Delta w_i(t)$, we propose the following model:

$$\Delta w_i(t) = |\tau_i| * \varepsilon \qquad (3)$$

where, τ, the ***adaptation parameter*** is given by,

$$\tau_i = \frac{r_1 + r_2 \ldots r_{i-1} + r_{i+1} \ldots r_{k+1}}{k} - r_i \qquad (4)$$

r_i is the reading from the i-th sensor, k is the number of neighboring sensors and ε, the ***scaling factor,*** is a small value $0 < \varepsilon < 1$ and is chosen appropriately for a given application. The scaling factor ensures that $0 < \Delta w_i(t) < 1$. An algorithm to update such weights based on equation (3) and (4) is given below:

Algorithm for Weight Updates

1. Initialize all weights at t=0, $w_i(0)=1$
2. At time t+1, calculate τ_i for all sensor readings within the region of event.
3. Calculate $\boldsymbol{\Delta w_i(t) = |\tau_i| * \varepsilon}$. Choose $\boldsymbol{\varepsilon}$ appropriately.

4. For *i=1 to k,*

    a. if Δw_i is minimum for all *i,* then $w_i(t+1) = w_i(t) + \Delta w_i(t)$

    b. if Δw_i is not minimum for all *i,* then $w_i(t+1) = w_i(t) - \Delta w_i(t)$

5. Repeat step 2-4 for next time interval

We provide a theoretical proof to validate the above algorithm in various working conditions of the sensors.

Theorem 1: *The weighting factor is increased only if the sensor reading is correlated with the k-neighboring sensor readings.*

Proof: In other words, we need to prove that $w_i(t+1) = w_i(t) + \Delta w_i(t)$ for the ith sensor when its reading agrees with the majority of the neighboring sensors in the given event region. Alternatively, we prove that if Δw (and in turn, τ given in equation (4)) is minimum for all *k* sensors, then we increase the weighting factor (proving 4a of the algorithm).

Case 1: Consider the case when the reading of the i-th sensor perfectly correlates with the k-neighboring sensors, i.e., r = *x* for all sensors. τ evaluates to $(k \times x)/k - x = 0$, and thus Δw= 0, which means that w(t+1) does not change.

Case 2: In the second case, we consider a situation where majority of sensors (say *p* sensors) in the set of *k* sensors are reporting same sensor reading (say r=x) as the i-th sensor. This means that the sensor *i* is correlated with sensors in subset *p*.

$$\underbrace{x,x,x,x,x,x,x,x,x,x}_{p}\ \ \underbrace{y,y,y,y,y}_{k-p}$$

It should be noted that τ in equation (4) simply represents the difference between a *desired output* and an *actual output*. This absolute value of the difference (desired output-actual output) will be a value that is closer to zero (but not necessarily zero) and thus minimum in the set of τ_i and Δw_i. Therefore, when the sensor reading correlates with majority of sensors in the event region, its corresponding adaptation parameter is minimum in the set and thus we increase the weighting factor.

Case 3: In this case, we consider a situation where *m* sensors in the set *k* are reporting same sensor reading (say r=x) as the i-th sensor. This subset of *m* sensors is much smaller than the remaining sensors in the set k, i.e., *m<k-m.* This means, sensor *i* is uncorrelated with *majority* of sensors in the event region. The difference value (from equation (4)) evaluates to value that is maximum in the set of τ_i. Therefore, the weighting factor is reduced as the sensor reading does not correlate with the majority of the neighboring sensors.

Corollary 1: In spatially correlated sensor networks, the adaptation parameter τ, is proportional to the posterior probability of nearest neighbor rule.

Proof: Consider *k* neighboring sensors. Let k_i be the sensors report same sensor reading in a given event as sensor node i. Let *p* be the subset of sensors that are uncorrelated, and remaining (k-p) sensors report a value *x***,** then equation (4) evaluates to:

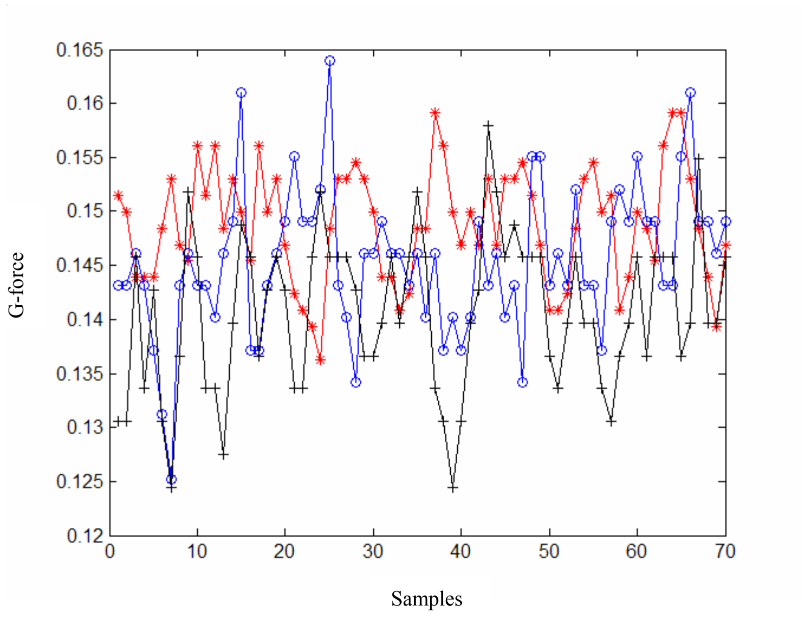
$$\tau \propto (k-p)x/k$$
$$\tau \propto (k-p)/k$$

According to *k-nearest neighbor rule,* the posterior-probability estimate that a reading r_i is equal to value **x** is given by $P(r_i\,|\mathbf{x}) = k_i/k$. (k-p)/k gives the posterior probability estimate that (k-p) nodes read the sensor reading **x**, in the groups of k sensors.

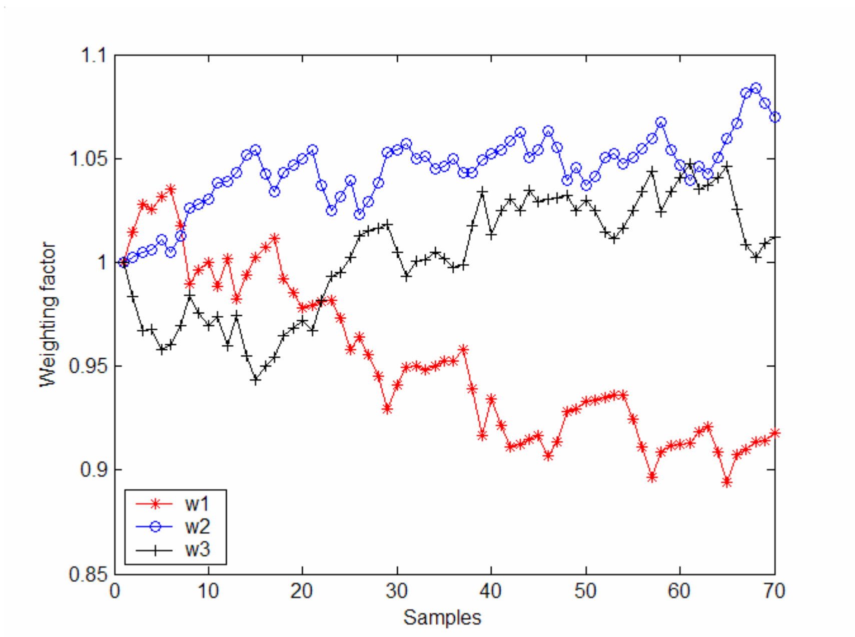
2.12 Simulation Benchmarking

Case 1: Consider three sensors. If all the three sensors are reading different physical value with large variations, then we trigger built-in test. We determine the likelihood of failure of a sensor by looking into how close the value is to the *usable threshold window.*

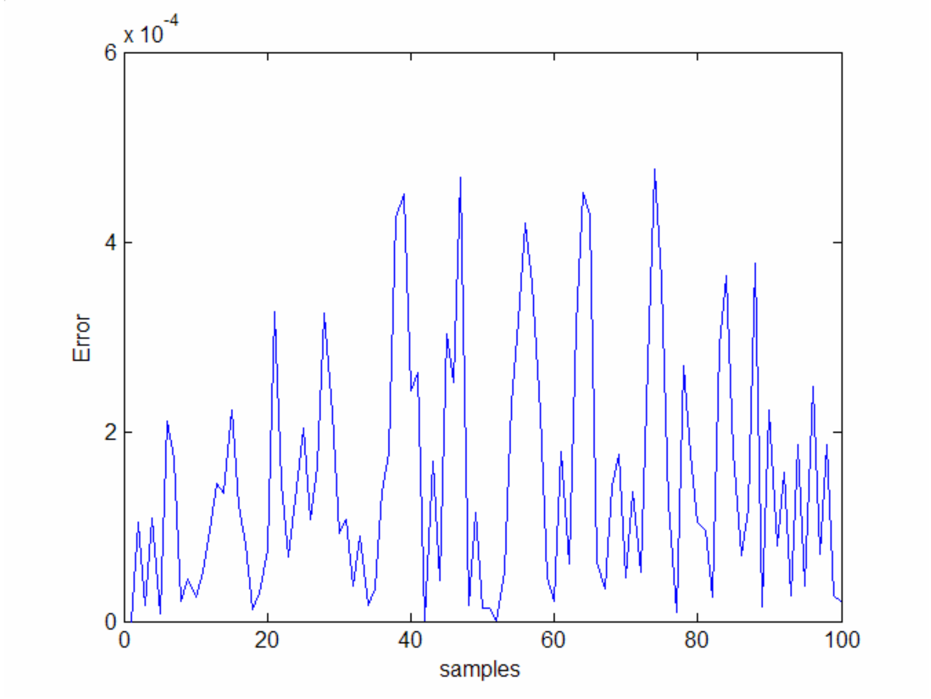
Case 2: Given 3 sensors deployed geographically close to each other. If two of the sensors are reading same value (say, temperature) and the third sensor reads a high value, then we can say that third sensor has a likelihood of failure. The weighted average ensures that the weight for the third sensor is reduced. We demonstrate this case by deploying accelerometer sensors to sense vibration on a dummy structural bridge. The weight updates for three sensors are as shown in figure 2.10(b). It is clear from the figure that sensor-2 and 3 are more correlated than sensor-1 and this can be observed from the data obtained from the sensors. Also, from the experimental set up, it is evident from the fact that sensor-1 is deployed in an area where there is more vibration in the structure compared to sensor-2 and sensor-3. In this experimental setup, we collect the data for period of 0.2 seconds sampled at 500Hz (sampling time of 0.002 seconds). We run the algorithm offline on stand-alone computer rather then online computation on a sensor node.



(c) Raw Sensor Readings



(b) Weight Updates for Three Sensors



(c) Absolute Error Variations

Figure 2.10 Data Aggregation Benchmarking Results

Figure 2.10(c) gives the error obtained from the difference between the aggregating data with our proposed algorithm and aggregating reading samples from three sensors without spatial correlation.

Case 3: If three sensors are reading the same value (with minimal variations), then they are spatially correlated and we have $\Delta w=0$.

2.13 Advantages of the proposed approach

The proposed model considers spatial correlation while computing the adaptation parameter τ. By using adaptation parameter, we increase or decrease the weights on the sensor reading dynamically. The model does not zero on the faulty sensors, but decreases the weight on the reading, which decreases the “contribution” of the faulty sensor to the aggregation. This helps in identifying any intermittent faults or communication faults that might occur for only small interval of time. If the sensor reading gets correlated with the neighboring sensors in the next time interval, then the weights are dynamically increased due to the adaptation parameter. This approach can be easily extended to all sensors within a cluster and the cluster-head acting as the fusion node to aggregate the data.

Data aggregation in sensor networks in general helps to reduce communication cost. However, a faulty reading by a sensor can represent a false estimate for the aggregated data. In this chapter, we proposed a robust mechanism to aggregate data from different sensors with some tolerance to faults.

In Section 2.2 and Section 2.11 we have proposed a methodology to identify and deal with faults and summarize the fault-tolerant data in sensor networks. In the next section, we identify other competing data reduction methods such as in-network data compression.

2.14 Data Compression Techniques

Data compression techniques can be broadly classified into two groups – general purpose and special purpose data compression. Special purpose data compression algorithms are focused on audio, video or image compression. General purpose data compression algorithms are generally applied to data buffers and text files. In both these categories, we can have lossless or lossy data compression [28]. In lossy data compression technique, decompressing the compressed output (upon retrieval) will not yield original data, but is close enough to be useful. This is most commonly seen in compressing multimedia data (audio, video, and image). Lossless compression technique allows exact original data to be reconstructed from the compressed data.

Generally, in sensor networks, it is desirable to have lossless data compression. However, depending on application where sensors are deployed, a tolerance is admissible for lossy data. We will see some of the lossless data compression techniques and their application and/or limitations in sensor networks. Although data compression is a vast research area by itself, we will identify common techniques and do a theoretical study on their application in sensor networks

Lossless compression techniques can be further be divided into:

1. Entropy methods:
    a. *Huffman coding* [29]: The basic idea in Huffman coding is to replace each symbol by code based on a known input probability distribution. The inputs are symbols and its corresponding weights (usually probabilities). This is not known *a priori* in many live applications including readings from sensors.
    b. Arithmetic coding [30]: Similar to Huffman coding, however, the symbols are replaced by real numbers in the range [0,1]. Arithmetic coding is expensive both in terms of computation and space (memory) requirements.
2. Encoding methods:
    a. *Run-length encoding* [31]: Very simple compression technique where

repeated data value are stored as single data value and corresponding count. Example, WWWWWBBBBACD can be replaced by 5W4BACD. Such compression techniques computationally less expensive.

3. Dictionary methods:

   a. *LZ77* [32]: This is a generalization of Run-length encoding method. Instead of looking at the whole data buffer at a time, LZ77 uses *windowing* method to replace repeated symbol by a single symbol. Therefore, LZ77 algorithm is sometimes referred to as *sliding window data compression* technique. This algorithm is highly suitable for sensor network data compression due to its simplicity and low computation overhead. We programmed our sensor node with an available open source implementation of LZ77 algorithm [33].

2.14.1 Data Aggregation and Compression

An important feature of our proposed data aggregation scheme is the adaptive weight change to deal with faulty information. Compression techniques such as LZ77 does not take into account such fault detection or recovery mechanism.

LZ77 implementation for sensor node (on TinyOS) looks into repeating 16-bit sensor reading (for example 0x17A8) within a given sliding window. From empirical analysis with temperature sensors, we found out that such repeating values rarely occur. Also, LZ77 uses buffer to temporarily hold the data before being compressed and therefore, there is an end-to-end delay in delivery of data to the base-station from the time a sensor reading was taken. Data aggregation scheme can be supplemented with data compression schemes to reduce communication costs in network. But such schemes would have high computation overhead.

2.15 Chapter Summary

In this chapter, we developed a data aggregation scheme that when implemented hierarchically will reduce the number of network transmission by an order of number of nodes transmitting. Moreover, such aggregation scheme also exploits the spatial distribution and correlation often seen in sensor networks to generate a weight adaptation scheme for fault tolerance. Such technique when augmented with Built-in Test (BIT) provides robust mechanism to process and acquire large amounts of data. To demonstrate the effectiveness of the aggregation scheme proposed in this chapter, we developed a middleware at the cluster-head node that implemented the timeout mechanism and aggregation of temperature sensor data. The detailed description of the design and implementation of the middleware as well as the experimental results and discussions are given chapter 6 of this thesis.

CHAPTER 3

INTELLIGENT DIAGONOSTICS IN SENSOR NETWORKS

3.1 Uncertainty, Fuzzy Logic and Approximate Reasoning

In most of the complex large-scale systems, uncertainty can be found both in information received as well as the operational state of the system. A comparative description of different systems and their uncertainties is shown in figure 3.1. As seen from figure 3.1, in sensor networks, there is a high uncertainty both in terms of data acquired and system operations.

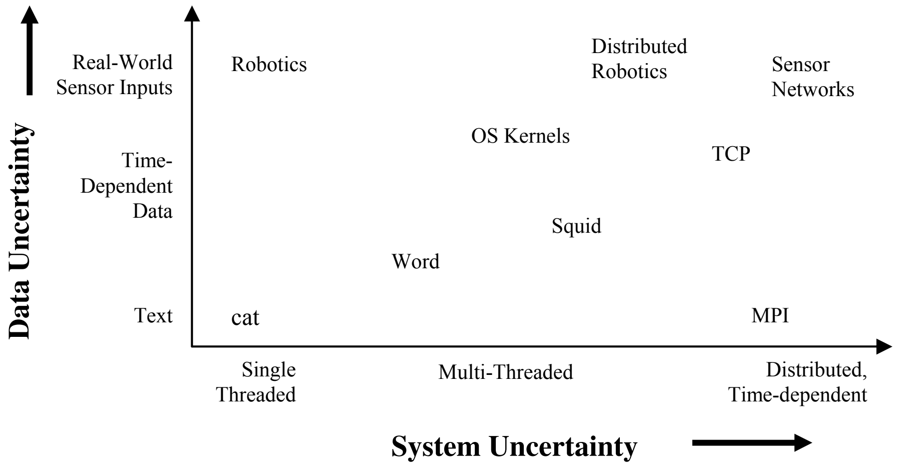


Figure 3.1 Study of Uncertainties in Systems. Image Courtesy of [34]

Uncertainty generally takes the form of *vagueness* or *ambiguity* in the context of information [35]. Vagueness in information is characterized by fuzziness, unclearness, cloudiness, etc. The level of uncertainty in both data and the state in sensor networks demands a simplistic approach to handle such uncertainties rather than designing a state-space model. Even designing a state-space model for a multi-parameter sensor networks is generally hard. Often times solutions for characterizing behavior (for control) of such complex parameter space problems is to use simple rule based techniques with expert knowledge. Also, due to the fact that uncertainties in these systems are common, a fuzzy rule based approach is an effective solution for characterizing the behavior of the given system (sensor networks). Fuzzy rule-based approach provides a simplified approach to multi-parameter problem that is persistent in sensor network. Even though fuzzy inference is not new in control/decision making, its application is a significant contribution to provide an approximate reasoning in sensor networks.

3.2 Fuzzy Logic Based Controller

3.2.1 Performance Degradation and Network Integrity

With fault-tolerant hierarchical data aggregation, we can ensure high data integrity and energy-efficient data delivery process. Our next key issue is to focus on a cost-effective way of prioritizing critical parameters of a given sensor network while maintaining high data integrity. With in-network data aggregation, there is always an energy-latency tradeoff. Data aggregation incurs end-to-end latency in data delivery to the processing station. By carefully analyzing the network and node parameters, we can control the process of data aggregation. For example, consider a decision making problem in which a cluster is to be selected to aggregate data based on critical parameters such as level of network congestion, power level of the aggregating node, data criticality, event levels, etc. In such situations, a feedback mechanism, often used in control theory can help to efficiently monitor the current state of a given cluster and to determine appropriate actions (say, to aggregate or not) to improve the operational performance. By dynamically changing the operations of the nodes based on a changing environment, we can extend the performance (for example network lifetime) of a given network. Specifically, we represent the ideal state of each cluster as a reference input to our controller. The ideal state for a sensor cluster would depend on the threshold limit on the operating values for parameters such as, bandwidth usage, network congestion, number of dead nodes, activity in the region, and overall energy (power) consumption. These parameters depend on the application for which the sensor nodes are deployed.

3.2.2 Need for State Feedback and Fuzzy Logic Based Control

Designing a state-space model for managing the parameters for sensor networks is difficult, simply because of unpredictable state and uncertainties in the operation of the network. One such way to handle uncertainties in the system and to characterize the behavior of the system (sensor network) using human knowledge and experience is by fuzzy logic. Fuzzy logic provides an alternative solution to non-linear control; non-linearity handled by rules, membership functions and inference process. Conventional controller such as PD or PID relies heavily on understanding the physical system (full knowledge of a mathematical model), and being able to define its transfer function mathematically [36].

3.2.3 System Design

We consider four parameters – network congestion, data burst due to activity in the region, data criticality, and battery power, as critical parameters that evaluate the state of the given network of unattended sensors. These parameters are fed to a fuzzy logic controller running on a base station as shown in figure 3.2, which gives an estimate based on the expertise or knowledge of the current state of the dynamic system. For instance, fuzzy rules can be designed to adaptively change the routing of the query based on the traffic (or congestion) in the network. Simple common-sense rules can be devised, such as "IF traffic is HIGH and battery is LOW, then *delayQuery* is HIGH". The controller makes intelligent analysis of the state of each cluster by considering the parameters and also their combinatorial effect, so as to idealistically distribute the information to all the nodes.

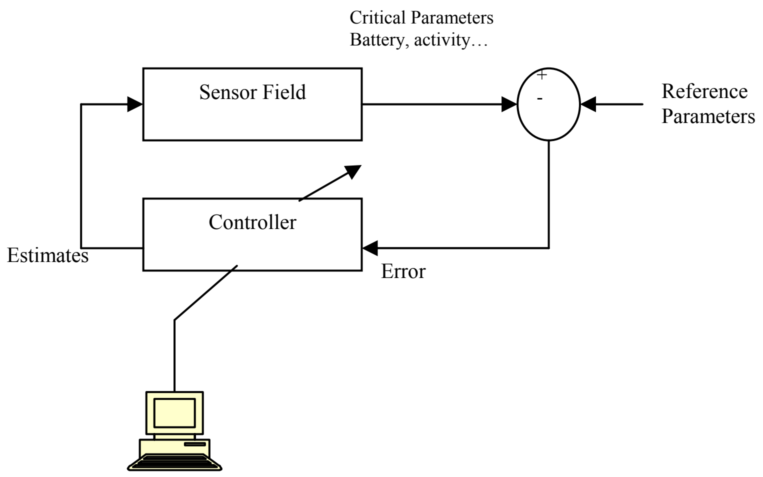


Figure 3.2 Conceptual Design of Intelligent Monitor

3.2.4 Detailed design

Typical fuzzy logic system consists of fuzzification, inference and defuzzification process [37]. Fuzzification creates fuzzy variables from crisp inputs that are then fed into the inference system. Fuzzy rule base drive the inference system to produce fuzzy outputs, which are defuzzified to get system outputs. The fuzzy rule base consists of fuzzy rules (IF antecedent THEN consequent) that are devised by an expert knowledge base or through system input-output learning. The core of fuzzy system is this rule base which mimics human reasoning [38].

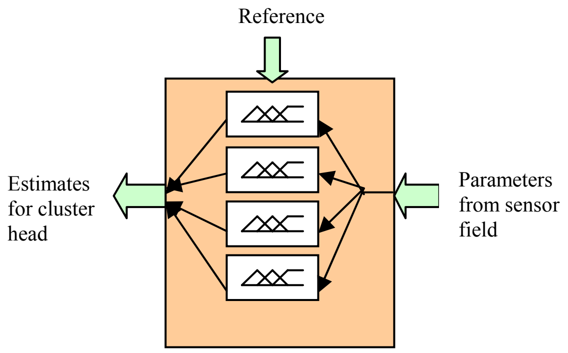


Figure 3.3 Inside a Fuzzy Controller/Monitor

The reference inputs to the fuzzy system helps in designing the membership functions for fuzzification as shown in figure 3.3. The crisp inputs to the fuzzy systems are the critical parameters – level of congestion, battery level, data criticality and burst.

Level of congestion plays an important role in routing our query from base station to the sensor nodes in the event region. We borrow the definition of *depth of congestion* from [39], to define level of congestion as the level in the routing hierarchical tree at which the backpressure message has traversed before a non-congested node is encountered. Whenever congestion occurs, the source node simply sets the congestion bit and sends the message back to the parent node (in the routing tree) as a backpressure message.

In our proposed approach, fuzzy rules have multiple consequents to achieve the following goals:

1. Minimizing congestion - Avoid overloading of a given node or cluster of nodes
2. Optimizing density - Optimal number of sensor nodes in a given cluster without loss of information quality.
3. Optimizing power - Optimal use of sensor nodes by conserving battery power.

3.2.5 Fuzzy Inference Engine

We define following rules for different functionality of our estimator.

For Congestion mitigation:

Define Q as queue length and DQ as rate of change of queue

```
R1: IF Q is empty and DQ is zero THEN congestion_alert is zero
R2: IF Q is empty and DQ is increasing THEN congestion_alert is low
R3: IF Q is moderate and DQ is zero THEN congestion_alert is medium
R4: IF Q is moderate and DQ is decreasing THEN congestion_alert is low
R5: IF Q is moderate and DQ is increasing THEN congestion_alert is high
R6: IF Q is full and DQ is zero THEN congestion_alert is high
R7: IF Q is full and DQ is decreasing THEN congestion_alert is medium
```

For power control:

Define P as battery power, E as events detected in the region, delay_query as delaying the query from base station to event region (caching the query/data) and Decrease_nodes as putting the nodes to sleep so as to sustain the lifetime of network (or decrease the *sleep time* of the nodes). Here, we assume that sensor nodes are not powered by a central power unit, but each sensor node is driven by their own power, thereby having a distributed power supply for the entire network.

```
R8: IF P is low and E is high THEN delay_query is high and
Decrease_nodes is zero
R9: IF P is high and E is high THEN delay_query is low and
Decrease_nodes is zero
```

```
R10: IF P is low and E is low THEN delay_query is medium and
Decrease_nodes is zero
R11: IF P is high and E is low THEN delay_query is zero and
Decrease_nodes is high
```

For Congestion Control:

Define LC as level of congestion, P as battery power

```
R12: IF LC is zero and P is low then delay_query is zero and
Decrease_nodes is zero
R13: IF LC is medium and P is low then delay_query is medium and
Decrease_nodes is zero
R14: IF LC is high and P is low then delay_query is high and
Decrease_nodes is zero
R15: IF LC is zero and P is high then delay_query is zero and
Decrease_nodes is high
R16: IF LC is medium and P is high then delay_query is medium and
Decrease_nodes is high
R17: IF LC is high and P is high then delay_query is high and
Decrease_nodes is high
```

We define burst rate as data transmission mode in which large amount of data appears for a small interval of time in the network. In sensor network, burst can occur during high events or when data is requested (periodic or aperiodic intervals). During burst, important sensed information needs to be delivered to the base station. Burst usually causes high traffic rates (high channel capacity/bandwidth utilization) which will need more intermediate nodes to transmit data. There is thus a higher requirement of resources - number of nodes.

Define BR as burst rate and decrease_node as putting some of the nodes to sleep (or decrease the *sleep time* of the nodes)

```
R18: If BR is high then Decrease_node is zero
R19: If BR is medium then Decrease_node is medium
R20: If BR is low then Decrease_node is high
```

3.2.6 Advantages of the approach

The controller can be easily adapted to varying application scenarios by changing the fuzzy rules. Having a continuous monitoring system such as the one proposed here, helps in sustaining a longer network lifetime by appropriately balancing critical parameters thereby ensuring survivability of the network. Efficient routing decisions can be adaptively made based on the congestion, power level and activity level in the region of interest.

3.2.7 Hierarchical Fuzzy Scheme

In section 3.2.5 we organize the fuzzy rules based on the functionality. However, a more efficient scheme to organize the fuzzy rules is in hierarchical fashion based on the criticality; top-level being highly critical to the system as shown in figure 3.4.

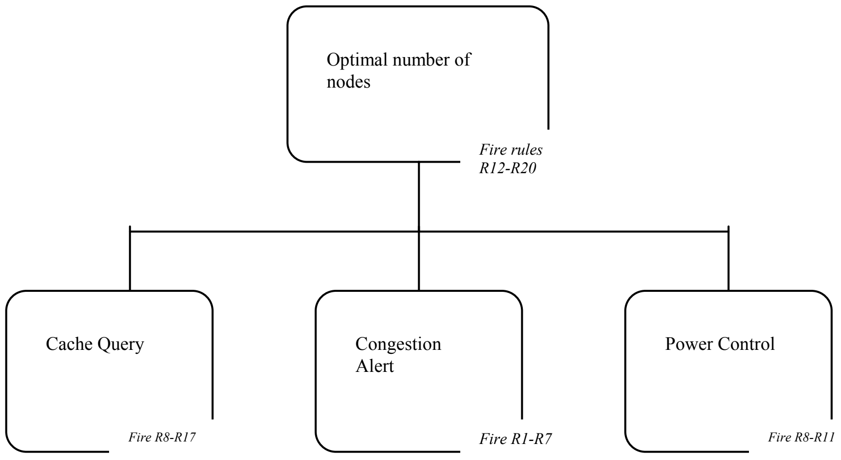


Figure 3.4 Hierarchical Rule Base

In addition to the hierarchical organization of fuzzy rules for criticality, we can design fuzzy rules to ensure data reliability and quality. These fuzzy rules are applied to fuse the data from different sensors which are organized in a tiered architecture. Thus, we not only ensure the quality and survivability of the sensor network, but also guarantee the quality of data delivered by the network.

As a case study, we consider the tiered sensory fusion approach [40] in an environmental monitoring application. Specifically, we consider inputs from three different sensory sources - temperature, visual, and heat-sensing IR camera. Single sensory information, such as from temperature, alone can not accurately determine the presence or absence of fire. However, with multiple sensors, detection, severity and localization of fire can be determined with lower false positives.

Consider group of low powered sensing devices capable of sensing temperature, distributed spatially to monitor environment (such as forest) for potential fire threats. A break-out fire produces enough thermal radiation for the temperature sensors to signal a presence of fire. In order to backup the information received from these sensors, a camera can provide visual aid to validate the presence of fire. However, smoke generated from fire can hinder the visualization of fire and therefore becomes harder to determine the severity of fire and hence difficult to isolate the hazard. We also note that camera alone cannot provide full information about the fire but relies initially on temperature sensors. An aerial information (such as from an UAV) can provide a thermal camera (heat-sensing IR camera) to determine location and spread (severity) of the fire. All three sensors compliment each other in monitoring, detecting and isolation of fire. A tiered approach helps to detect feature of the threat in each tier as shown below:

Tier	Problem
1	Detection
2	Detection, Validation
3	Detection, Validation, Severity

In tier 1, group of in-situ sensing devices provides the initial detection of fire based on temperature readings. In tier 2, validation of the initial detection is done by camera. The severity of the threat (fire) is estimated using an IR camera based on the validation and detection from camera and temperature sensors respectively.

We develop a multi-level *danger or severity score* for each small region of the environment. This score *d*, can take the value $0 \leq d(x,y) \leq 1$. Fuzzy rules can be developed at each tier and then fused based on a weighting factor for each of the defuzzified value for each sensor. Defuzzified values at each tier gives an estimate of severity score *d* for a region defined within the boundary *x,y*.

d_t: Severity score obtained from temperature sensory information.
Number of sensors reporting fire (*low, medium, high)*
$\oplus$
Temperature range *(low, medium, high)*
=
d_t *(low, medium, high)*

d_i: Severity score obtained from Infra-red camera.
Intensity (*yellow, orange, red)*
$\oplus$
Area covered *(small, moderate, large)*
=
d_i *(low, medium, high)*

d_c: Severity score obtained from camera.
Fire presence (none, low, medium, high*)*
=
d_c *(zero, low, medium, high)*

$\oplus$ represents the fuzzy *t-conorm* and *low, medium, high* are fuzzy linguistic variables.

Total severity score is then calculated based on the defuzzified value of d_c, d_i, and d_t by weighted averaging the values:

$$d(x,y) = \frac{w_1 d_t + w_2 d_i + w_3 d_c}{\sum_{k=1}^{3} w_k} \tag{5}$$

where w_k is the weighting factor for each sensor. An important contribution of our approach is how these weights can be updated based on the environmental changes observed. That is, based on the condition of fire, the weights should be changed adaptively. For example, fuzzy inference can again be applied to evaluate the weight for each sensor (say, camera) in the following fashion:

```
if smoke is high and range is far,  w3 (weight for camera) is low
if smoke is high and range is near, w3 is med
if smoke is low and range is far, w3 is med
if smoke is low and range is near, w3 is high
```

3.2.8 Protocol Design for Multi-Sensor Nodes

In order to realize an implementation of such a monitoring method for sensor network, it is necessary to develop a protocol. Such a protocol aids in message passing of critical parameters from the sensor field to the base station. In this section, we describe in detail the high-level communication message structure and a protocol to communicate and interpret the message to and from the sensor network.

We consider two types of messages - beacon message and a report message. Beacon message is a broadcast type message sent by the base station. Beacon messages acts like stimuli for the sensor nodes to report their current operational status back to the cluster-heads. In order to save on the costly communication to and from the sensor network, the cluster-heads can make simple but effective analysis of the sensor node reported information.

If there is no substantial change in the information reported or there is no adverse conditions (say high congestion, low battery, etc.) reported, the cluster-head does not report back the information to the base station. This reduces considerable amount of packets in the network, thereby reducing communication cost as well as congestion and latency within the network.

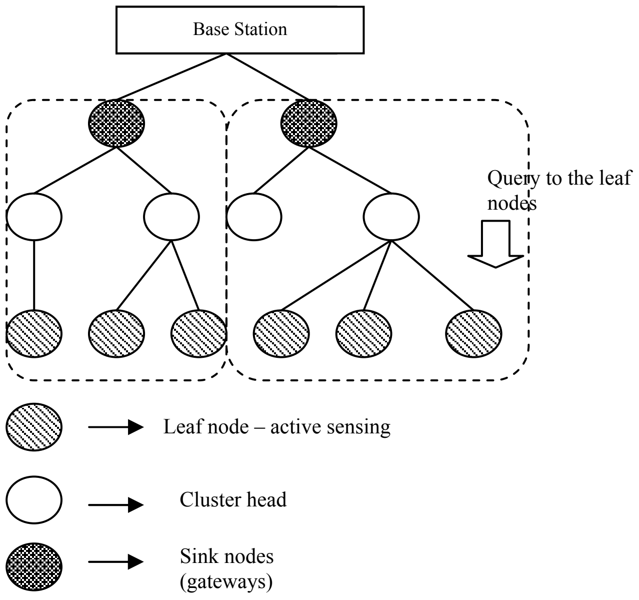


Figure 3.5 A Sample Tree Structure for Distributed Sensor Routing

Each beacon message is sent at pre-determined time interval from the base station. These beacon messages traverse from the base station to sink nodes to cluster-heads and finally to the sensor nodes. A sample illustration of such a tree structure is seen in figure 3.5. Cluster-heads are generally sensor nodes with higher functional capabilities than simply sensing the environment. Once beacon message is received, each sensor node sends a message to its cluster-head. This message consists of current sensor reading of interest, along with important header information. The message structure is as shown in figure 3.6.

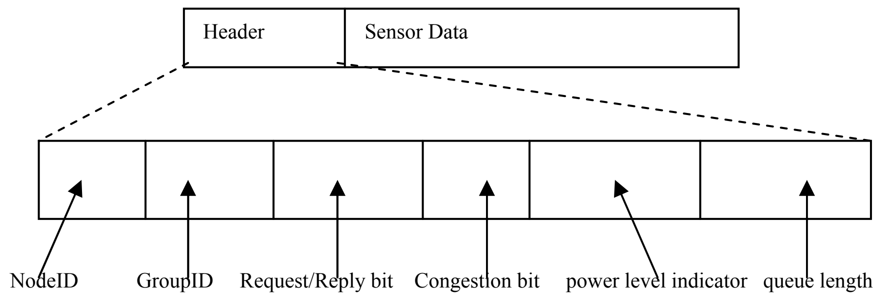


Figure 3.6 Message Structure

NodeID uniquely identifies each of the sensor nodes in the network. GroupID uniquely identifies the cluster to which the sensor node belongs. Request/Reply bit can take the value 0 or 1. 0 signifies a beacon or cluster-head request and 1 signifies a reply to a request from the sensor nodes. The congestion bit is set to 1 if the sensor node is overloaded. If the sensor node is acting like a message router involved is message hopping, then there is high likelihood that the sensor node can be overloaded. Power level indicator identifies the power at which the sensor node is working. It could be simply a battery voltage level or power usage represented as percentage of total power available to the sensor node. Finally, the queue length signifies the messages that occupy the queue and waiting to be transmitted. This message header can be implemented without much difficulty in real-world sensor nodes that provide high level programming capabilities (such as TinyOS running on Crossbow motes [41]).

The beacon request sent from the base station is forwarded down the routing tree to the sensor nodes. Each of the sensor nodes reply back to the cluster-head (or their parent in the routing tree) with the message which contains the sensed data and appropriately setting the header. The number of sensor nodes deployed varies, depending on the given application. The fuzzy inference engine can be run either on the base-station, if there is a single cluster of sensor nodes or run hierarchically on the cluster-heads. If there are several clustered sensor nodes, each cluster-head can run fuzzy rules based on the header information in the message, and send their recommendations to the base-station. The base-station in turn runs fuzzy logic controller for the recommendations received from all the cluster-heads in order to determine best possible parameter estimate for the entire sensor network. This is critical since a recommendation from one cluster might be varying or conflicting with its neighboring cluster.

3.3 Application-specific Example

In the last section we considered fuzzy based inference for each cluster so as to optimize the performance of the whole network. In this section, we present another fuzzy based approach for clustering deployed sensor nodes. Such clustering mechanism can be highly applicable for applications such as object tracking. We will give a detailed explanation of how such clustering mechanism can be used for “intelligently” tracking a moving object without having to know the underlying dynamics of the moving object.

The concept of object tracking has been studied extensively in mobile robotics [42-44]. Object tracking is usually combined with other processes such as object detection and object classification. Based on the type of sensor used, object detection can be either visual (using camera) or motion based (using motion detectors). Object classification involves comparing the detected object with a known object, generally using pattern recognition/image processing techniques. The principle of object tracking relies on the sensory feedback in order to calculate the new position (or state) of the moving object. Thus, the process of object tracking is an estimation process. Several estimation techniques such as Kalman filters [45], Bayesian estimation [46], and Kernel particle filtering [47] have been studied for object detection and tracking.

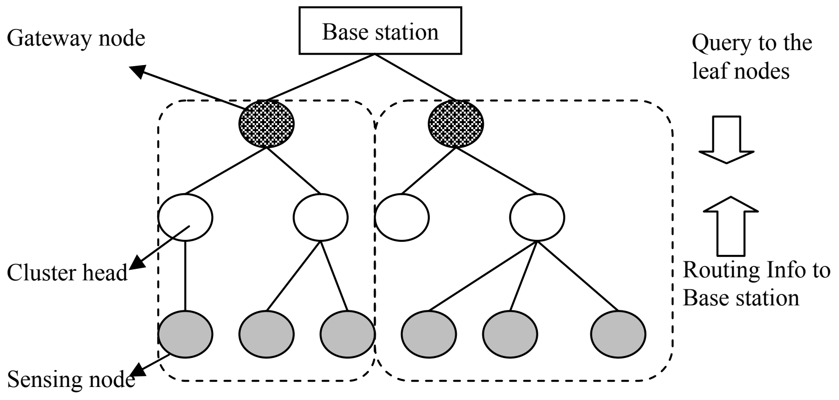
Target or object tracking in sensor networks has been proposed in [48-50]. Our method of object tracking is based on the topology of the sensor nodes deployed in order to estimate the object feature (speed or position) without the underlying knowledge of the dynamics of the object. We use the principle of overlapping clustering and data-driven techniques [51] to predict the motion of a given object.

Randomly deployed sensor nodes are grouped into several clusters based on some metrics (say, distance). Clustering can be organized in a hierarchical fashion, with sensing nodes at the lowest level in the hierarchy. The sensor information is passed on to cluster heads which, in turn, pass information to the base station. The criteria for clustering could also be the number of hops from the sensor nodes to the cluster head.

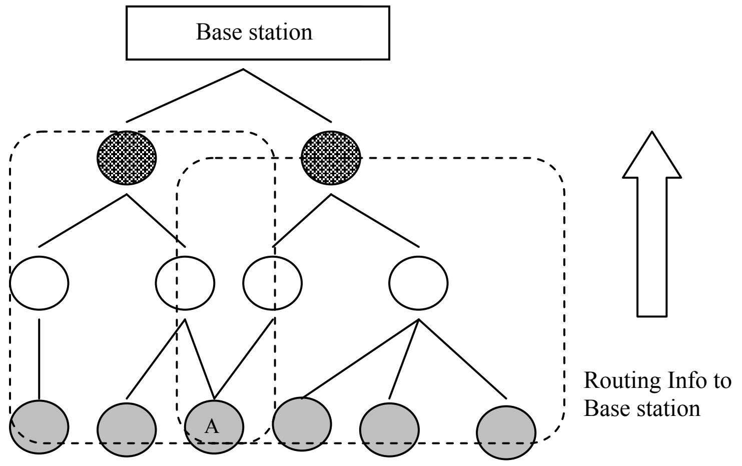
The main advantage of clustering is to divide the problem space into several sub-problems and solve each sub-problem for estimation; a divide-and-conquer approach.

As note in chapter 1, although there are several proposed clustering algorithms for sensor networks, to the best of our knowledge, there is no literature which utilizes *overlapping* (fuzzy) clustering mechanism in sensor networks. Such an overlapping clustering topology has several advantages when seen from a routing point of view.

A hierarchical routing tree structure based on non-overlapping clustering is presented in figure 3.7(a). In this type of routing, sensing nodes (lowest level of hierarchy) pass information to their respective parent nodes (cluster-heads). A routing tree structure based on overlapping clustering is presented in figure 3.7(b). As seen from figure 3.7(b), when routing information, node-A sends data packets to both cluster heads and eventually to the base station. At the base station we have redundant information from two cluster heads. Since the packet header of the communicated data contains the node identification (nodeID), base station knows from which node it is receiving the data and therefore can simply ignore the data or use it based on data fusion algorithms, if there is any redundant or duplication of information.



(a) Routing Tree with Single Path



(b) Routing Tree with Single Path

Figure 3.7 Hierarchical Routing Tree Structure

We see that there is redundant query and information passed from and to the sensor nodes. This will incur more communication cost than normal hierarchical routing, as in figure 3.7(a). However, with overlapping clustering, even if one of the cluster head fails, the information is still passed on to the base station.

Another advantage of overlapping clustering is in object tracking. If the nodes belong to different clusters to some degree, then this degree can be used as the probability of detection of the moving object in that particular cluster. Therefore, clustering based distributed computation helps in predicting the object feature (speed, position) without having to know the dynamics of the object being tracked.

An important design consideration in overlapping clustering is to find the minimal (threshold) number of sensor nodes that belong to two or more clusters. Sharing more nodes among clusters increases the network traffic which could overload the routing nodes (in multi-hop scenarios) and can cause network congestion. On the other hand, fewer nodes in the overlapping region reduce redundancy.

3.3.1 Clustering mechanism

If the deployment of the sensor nodes is known *a priori*, then the base station can run a clustering algorithm to identify cluster-heads and assign each sensor node the degree to which it belongs to multiple clusters. One such clustering algorithm to generate overlapping clusters is Fuzzy c-means (FCM) clustering [52]. FCM which is commonly used in pattern recognition is based on the principle of minimizing the objective function given by:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} \| x_i - c_j \|^2 \tag{6}$$

where u_{ij} is the degree of membership of x_i in cluster j, c_j is the center of the j-th cluster and m (m>1) is the fuzziness measure. FCM is an iterative algorithm which starts by randomly selecting cluster-heads for the given dataset. By iteratively updating the cluster centers (minimizing the objective function), FCM moves the cluster-head to the right location within a given dataset. Such an algorithm can be used for partitioning deployed sensor nodes.

Another alternative is to probabilistically select a cluster-head [53]. A chosen sensor node broadcasts a message as a cluster-head. All the neighboring sensor nodes reply with an acknowledgement message. Based on the signal strength of the received message, the cluster-head can assign degree of membership for each of the sensor nodes and thus forms a cluster. A single sensor node could potentially receive more than one broadcast message from multiple cluster-heads. In these situations, the sensor node has the possibility that it will belong to two or more clusters. The degree of membership to a cluster, however, depends on how far the sensor node is from the cluster-heads.

3.3.2 Object Tracking

With overlapping clustering, each node belongs to two or more clusters with a certain degree of presence (μ) in each.

Hypothesis 1: *A high weighted average of aggregated information from the cluster center CH_i implies the object is in that cluster.*

As stated above, for densely deployed sensor nodes where some of the sensors belong to two or more clusters, if the sensor nodes in cluster *i* are close to the cluster head CH_i, then μ_i will be high compared to their μ_j with neighboring cluster head CH_j. With this we can average out the sensor reading (s_i) for *k* sensor nodes which have detected an object, thus giving us the aggregated value:

$$\frac{\sum_{i=1:k} s_i \times \mu_i}{k} \tag{7}$$

In figure 3.8, an object (represented as star) is sensed by neighboring sensors and the information (sensed value – binary or real) is sent to the cluster heads.

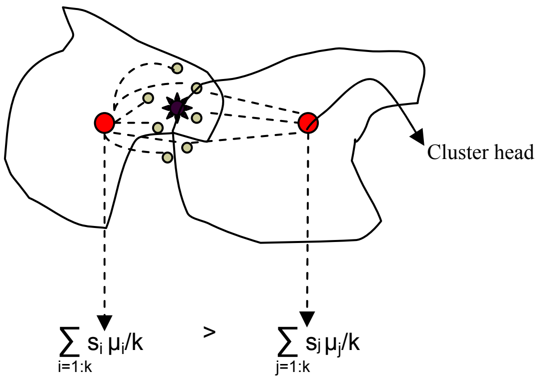


Figure 3.8 Object Detection by Multiple Sensors

The aggregated value given by (7) will be high for cluster head where the object is located at present, since the μ_i for the sensors in that cluster head is high.

Different scenarios are analyzed in order to evaluate the performance of our algorithm. We define *common nodes* as those sensor nodes that belong to two or more clusters. *Independent nodes* are sensor nodes whose degree of membership to cluster is 1, i.e., they always report information to one cluster head. Also, the sensor reading s_i is normalized to read between 0 and 1.

Consider the following three scenarios in object tracking:

Case 1: A moving object is detected by group of *k* independent sensor nodes inside the region of a specific cluster. Since the independent nodes have degree of membership equal to 1, the aggregated value from equation (7) is nothing but the aggregated value of all the sensor reading.

Case 2: A moving object is detected by a group of sensor nodes common to two clusters. This is a situation, where the target object is in the *overlapping* region of the two clusters. The sensing devices in this region report to both cluster heads. The aggregated value then depends on the degree of membership for each of the sensors in the overlapping region.

Supporting example: Consider k=4, the number of sensors in the overlapped region that detect an object at a given time instance t. Let μ_i={0.1,0.8.0.5,0.4} and μ_j = {0.1,0.2.0.5,0.6} are degree of membership of sensors to two clusters *i* and *j* and normalized sensed values is s={0.1, 0.2, 0.4, 0.4}. The sensed values could be set based on the voltage levels in case of analog sensors. From our hypothesis, we estimate the chances of object entering a cluster by finding the aggregated value given in equation (7). The aggregated value for cluster i is high, indicating that the object is entering cluster *i*.

Case 3: A moving object is detected by group of independent as well as common nodes. If the number of independent nodes detecting the object is higher than the common nodes, the aggregated value in equation (7) will be higher than aggregated value obtained from *Case 2*, because of the high degree of membership for independent nodes. The base station will be able to determine the delayed trajectory response of the moving object by comparing the aggregated values from different cluster head in the region of event.

Supporting example: Consider k=4, the number of common nodes that detect an object at a given time instance t. Let μ_i={0.1,0.8,0.5,0.4} and μ_j = {0.1,0.2.0.5,0.6} are degree of membership of sensors to two clusters *i* and *j* and normalized sensed values is s={0.1, 0.2, 0.4, 0.4}. We will also assume that some independent nodes (m=3) within cluster *i* has detected the object. From equation (7), the aggregated value for obtained in cluster *i* is high compared to aggregated value generated at cluster *j*. From the time series (looking at object's location at time t-1) and the aggregated value at the given time instance t, the base station can estimate the course of action of the moving object. Since this algorithm is online and data-driven, even if the object changes its trajectory, the base-station can quickly estimate the new changed trajectory comparing the aggregated value generated at the cluster-heads.

3.3.3 Protocol and Algorithmic Design

In order to realize an implementation of such a tracking method for sensor network, it is necessary to develop a protocol. Such a protocol aids in message passing from the sensor nodes to the cluster-head. In this subsection, we describe in detail the high-level communication message structure and an algorithmic design to interpret the message to and from a node.

Each sensor node sends sensor reading in a message packet with header information containing its identification (node-id) as shown in figure 3.9.

node-id	sensor reading

Figure 3.9 Simulation Snapshot

Upon receiving such message, each cluster-head (CH) looks up for the degree of membership for each node based on the node-id. A simple *look-up table* mechanism can help in retrieving stored degree of membership for each node within the cluster. Each cluster-head then computes the aggregated value (see equation (2)) and sends the result to the base-station (sink node) with its identification. The detailed algorithms at sensor node, cluster head and base station are given below.

Algorithm 1: Running on each sensor node

```
If (events detected)
{
   Send message with node-id and object info
}
```

Algorithm 2: Running on each cluster-head (CH)

```
If (message received)
{
Start timer
   While (timeout)
   {
   Store messages received
   }
```

```
    For (each node_id in the received messages)
         Retrieve membership degree
         Perform aggregation based on equation (2)
}
```

Algorithm 3: Running on base-station (sink node)

```
For (each message received from CHs)
{
Generate timestamp t
Compare the aggregated values from other CHs
Based on previous timestamps t-1, t-2,…t-n, from the time
series    aggregated    data    received    from    different    CH
calculate the speed of moving object.
}
```

3.3.4 Limitations

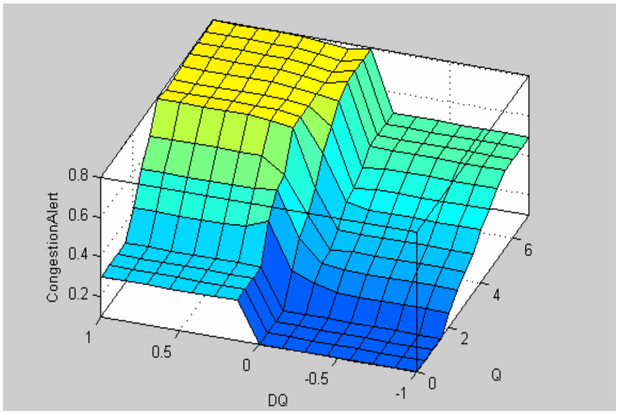
The proposed algorithm relies heavily on the clustering of the deployed network. Although the algorithm can track a moving object, the exact positioning of the moving object can only be known if position of the cluster-heads is known (either through GPS or relative positioning).

In case of multiple objects being tracked, having relatively high number of clusters helps in determining high number of aggregates (equation 2) and thus helping in efficiently tracking each objects separately.

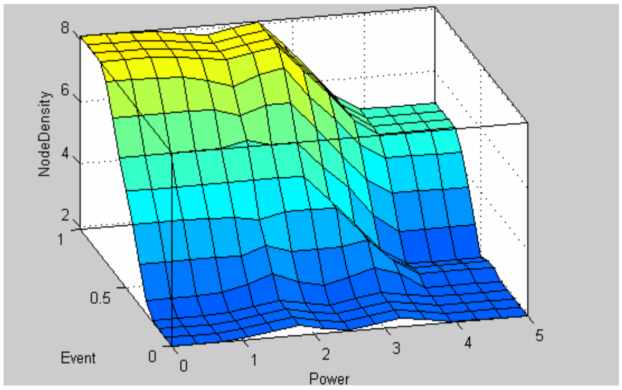
3.4 Simulation Benchmarking

Part(a): In this part, we develop the fuzzy rules using fuzzy logic toolbox in MatLab. Figure 3.10a and 3.10b shows the control surface for rules 1 through 7 and rules 8 through 11 respectively.

In this section, we develop a conceptual simulation of subset of the problem described in this chapter. Specifically, we run a fuzzy logic controller with rules 8-11 for optimizing node density based on the battery power levels and activities/events in the given cluster or region of interest.



(a)



(b)

Figure 3.9 Control Surface from Fuzzy Rules

We define two important variables – α as minimum number of nodes necessary to guarantee coverage and β as probability of failure of event detection. Here α is chosen appropriately for a given application. The number α can also be determined by using optimization schemes that guarantee optimal coverage for a given application. Finding an optimal α depends on the current state of the network including operating power level (battery voltage) of the sensors within the cluster, number of events in the network, criticality of data, etc.

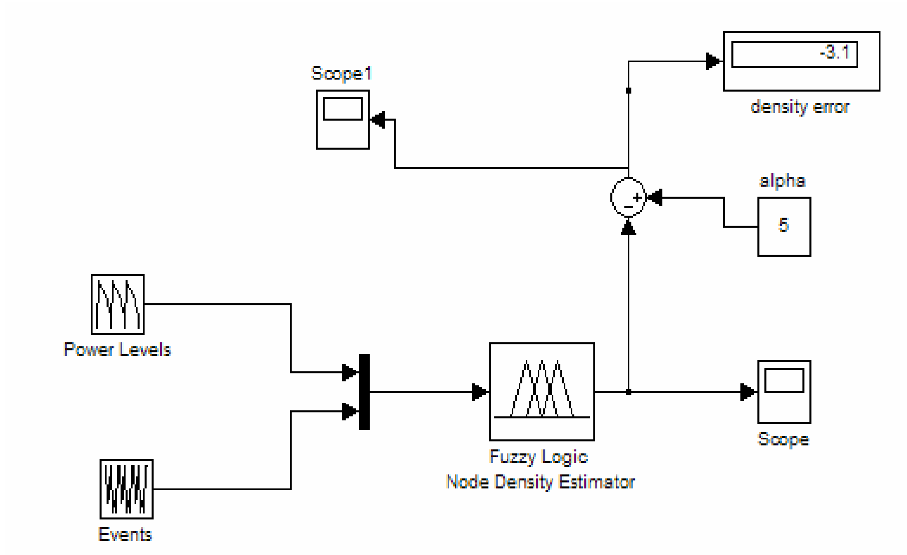
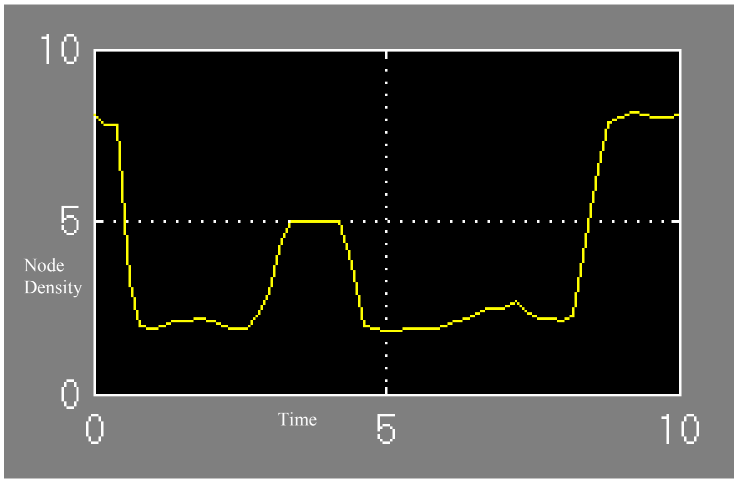


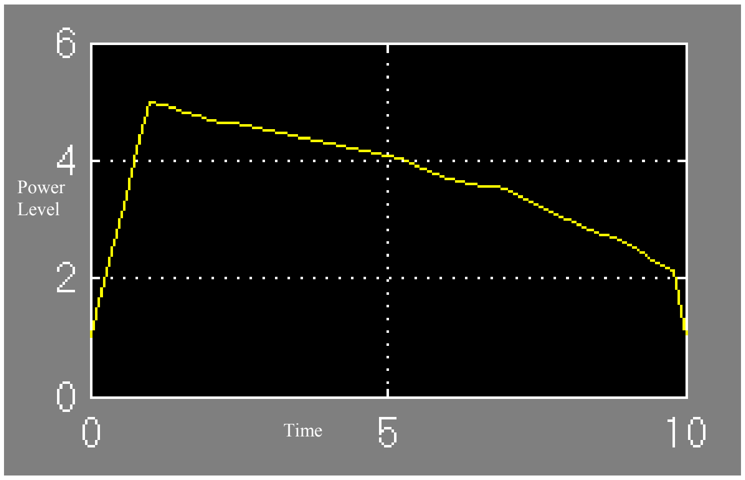
Figure 3.11 Simulation Design for Fuzzy Inference with Power and Event Levels

Figure 3.11 shows the conceptual simulation developed in MatLab Simulink. The input parameters to fuzzy controller are the power level and events detected (normalized to 1) at discrete-time steps. The output of the fuzzy controller is the estimate for number of nodes based on the input parameters. The *density error* ε is used to evaluate β. If error is positive, then β= ε/α, gives the probability of failure to detect an event in next time step. If error is negative, then the region of interest or cluster region has sufficient nodes to meet the coverage criteria.

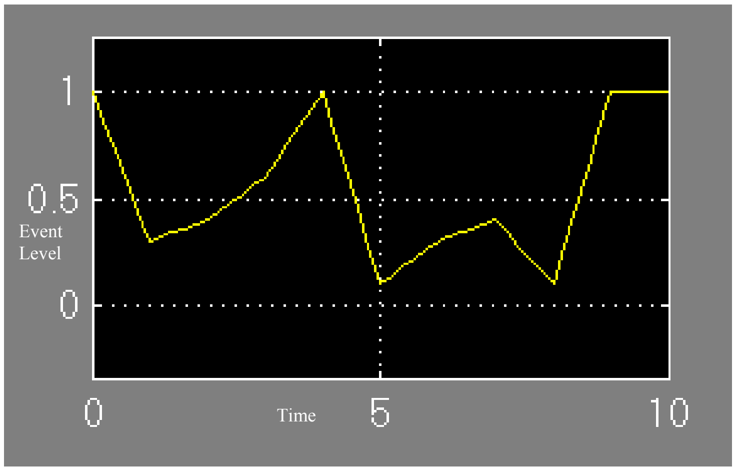
Figure 3.12a shows the simulation results for density estimates for different power levels (figure 3.12b) and event levels (figure 3.12c). We can see a clear raise in the estimates for node density during the final simulation time steps, due to the decrease in power level and an increase in events in the region.



(a)



(b)



(c)

Figure 3.11 (a): Density (b):Power and (c):Event levels at different time steps

<u>Part(b):</u> In this part, we will simulate the object tracking algorithm based on the principles of overlapping clustering. We simulate different scenarios when the moving object has entered the sensor field (region deployed with sensor nodes). For practical implementation we can assume that each sensor node is equipped with motion detectors (such as Passive Infrared (PIR) sensors) along with processor-radio board for limited computation and communications (for example, sensor nodes such as Crossbow motes).

Scenario 1: In this scenario, we consider a moving object which moves from one clustered region to another without changing direction (trajectory) as seen from figure 3.13 (moving object is represented as a circle). At each step, the aggregated value is calculated based on the number of sensors sensing the object. By comparing the aggregated value from multiple cluster-heads, the base-station can estimate the trajectory. Note that, since the object's trajectory is constant, the base station can also perform *temporal correlation*. Based on the aggregated values (spatial correlation) and time series data, the base-station can generate the trajectory of the moving object (for example, as given by dotted line in figure 3.13). The actual orientation of the object might be different from the estimation, which will constitute the *miss rate*. Since the algorithm is online, at the next iteration, based on the current location of the object, the aggregation is again computed giving a new estimation of the object's orientation thus minimizing the overall *miss rate*.

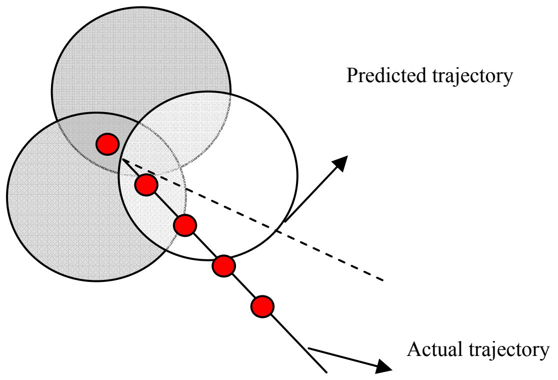


Figure 3.13 Overlapping Clustering Intuition

Scenario 2: In this scenario, we consider a moving object which moves randomly (without a constant trajectory) in the sensor field.

In order to simulate both scenarios, we consider a path traced by a moving object as show in figure 3.14.

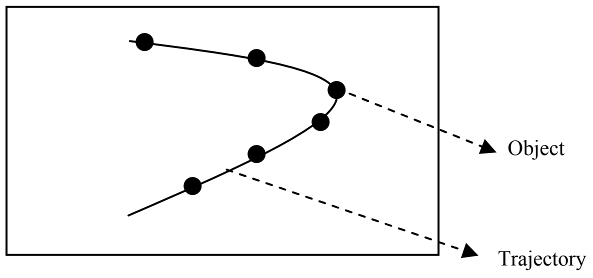


Figure 3.14 Trajectory of a Moving Object

We simulated both the scenarios with our proposed hypothesis, in a discrete-event structure. DEVS-Java [54] a discrete-event simulation environment developed in Java and based on DEVS formalism was used to simulate the object tracking scenario. We run our fuzzy clustering and get cluster centers for the sensors deployed. The simulation shows an object entering the cluster field and detected by sensors. The snap-shot of the simulation is presented in figure 3.15.

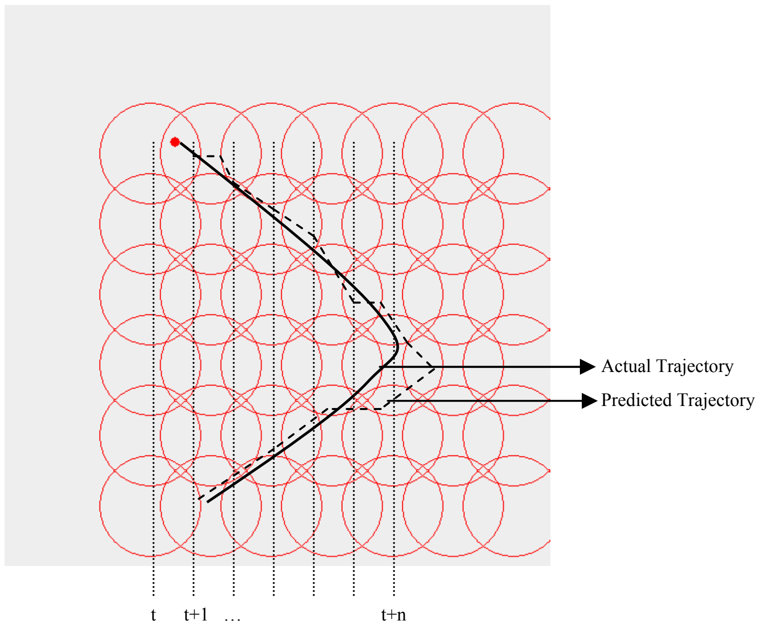


Figure 3.15 Simulation Snapshot

As seen from figure 3.15, the algorithm does well in predicting the trajectory of the object. However, when the object suddenly changes its trajectory (as seen at time *t+n*), there is an overshoot in the prediction. This overshoot results from the base station looking at the motion of the object at previous time intervals (t-Δt) and the current aggregated value.

Since the initial movement of the object has constant trajectory, the prediction relies on temporal correlation. As the object changes it course, the algorithm has to rely on the aggregated value sent by the cluster-heads.

3.5 Chapter Summary

Conventional feedback mechanism (such as PD or PID) and iterative formulation of dynamics (such as Newton-Euler formulation) could be prohibitively time consuming. The basic idea for probing the given sensor network (system in this case) is to mimic human mind to deal with complex systems. Human mind maintains a modular perception with relatively simple nonlinearities [55]. Our approach was to use human reasoning (by designing common sense rules) to deal with complex multi-parametric systems such as network of sensors. Fuzzy based reasoning is not new in the area of controls and decision making. The use of fuzzy controller for estimation has been proven to be easier to handle multiple variables/parameters for large-scale systems. System identification of sensor network is generally difficult due to uncertainty in the system (sensor network) variables. We exploit the nature of fuzzy logic controller which efficiently handles uncertainty and nonlinearity in the system. It should be noted that our approach is to provide a simplistic rather than accurate reasoning about the working condition of the network (diagnose) and eventually use this reasoning to update the operating parameters (such as sleep time, power level, etc.) of each or group of sensor nodes so as to improve the performance of the entire network. The real-world experimentation for node criticality based on fuzzy logic approach is presented in Chapter 6.

CHAPTER 4

MULTICRITERIA DECISION MAKING

4.1 Decision Making Process

The process of decision making is to simply choose an *action* among set of *alternatives* based on some *criterion.* For example, consider the systems diagnostics using Fuzzy logic discussed in chapter 3. The number of sensor nodes deployed generally varies, depending on the given application. The fuzzy inference engine can be run either on the base-station, if there is a single cluster of sensor nodes or run hierarchically on the cluster-heads. If there are several clustered sensor nodes, each cluster-head can run fuzzy rules and send its recommendations to the base-station. In some cases, the recommendations sent to the base-station from each one of the cluster-heads are conflicting in nature. When such conflicting recommendations exist, it is generally desirable to automate the decision making process, for example, to select the right cluster for managing the critical operating parameters. There are several decision making models available [56] and such models have been extensively used in the field of economics.

4.2 Multi-Criteria Decision Making

The problem of selecting an action among set of alternatives becomes harder when the decision making process involves several *criteria* rather than a single criterion. Such problems are referred to as Multi-Criteria Decision Making (MCDM) [57] problems. MCDM is the study of discrete decision making involving two or more criteria (sometimes conflicting) or objectives. In MCDM problems, the goal is to select an alternative (choice or a system) from a set of relevant alternatives by evaluating a set of criteria. For example, consider the problem of selecting a car from a given set of three cars S={A, B, C}. This set S, represents out set of *alternatives*. Selecting a car of our choice is the *action.* The sample set of *criteria* to be evaluated can be C^S=*{Fuel efficiency, Luxury, Price}*. A conventional methodology to select a car is based on prioritizing the criteria for selection. Such priorities are generally user-dependent. A simple weighting factor for each criterion can prioritize the selection process.

Let us now generalize the problem of MCDM by taking finite number of actions and criteria. Let $\Omega = \{s_1, s_2, \ldots s_m\}$ and $X = \{x_1, x_2, \ldots x_n\}$ be set of alternatives and set of criteria respectively. The decision making process proceeds by formulating a matrix **A** with set of criteria and set of alternatives given by:

$$\mathbf{A} = \begin{array}{cc} \begin{array}{c} s_1 \\ s_2 \\ \vdots \\ s_m \end{array} & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \\ & \begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \end{array} \qquad (8)$$

Each entry a_{ij} denotes the degree to which the criterion x_j is satisfied by the alternative s_i. The idea is to now reduce the multi-criteria problem into a single global criterion problem by aggregating all the elements of matrix **A**, given by $a = H(a_{1j}, a_{2j} \ldots a_{mj})$, where H is the aggregation operator. Most common aggregation operator is the weighted arithmetic mean. In this chapter, we will investigate the necessity of MCDM in sensor networks, the pitfalls of common aggregation operators (such as weighted mean) and provide a counter-measure for aggregating criteria without using common aggregators.

4.2.1 Motivating Examples in Sensor Networks

Consider an application of monitoring a large structure such as a bridge, using sensor networks. Ideally, we would want to sustain the lifetime of the deployed sensors for a long time, since the redeployment generally can be difficult, both in terms of ease and cost of deployment. In this case, network lifetime is more important criterion than accuracy of data, and hence we assign *network lifetime* a higher weighting factor. Consider another instance, application such as habitat monitoring. High network lifetime is desired but not a required behavior. However, more importance or priority needs to be given to efficient communication from the habitat to a command center. Consider yet another application of monitoring chemical or nuclear spill in a region. Such applications have high demands for larger node deployment in order to capture and localize all critical events in the region. Each application thus has varying demands or requirements that need to be satisfied by properly prioritizing the behavior or properties of sensor networks.

In order to prioritize the system behavior, we will need to establish *criteria* for prioritizing. This system behavior can be thought of as action to be selected from set of alternatives. For example, if there were three different tasks that needed to be completed, a human might prioritize them based on time, cost or importance. Therefore, the problem of assigning behavior to a given system then becomes a MCDM problem.

4.2.2 Interacting Criteria

A common method as discussed earlier to evaluate set of criteria is to use aggregator operator to reduce the multi-criteria problem into a single global criterion problem by aggregating all the elements of matrix **A**. A tradition method is to use weight sum (or weighted mean) on the row of matrix A given by:

$$\sum_{i=1}^{n} w_i \times a_{1i} \tag{9}$$

This is a simple approach; however, despite its simplicity it has drawback in that it assumes that the criteria are independent. The criteria can interact with each other which requires the replacement of weighting factor *w* by a more comprehensive non-additive set function on set X (set of criteria) which not only considers weighting factor on each criterion but also weighting on each subset of criteria. [58] gives an overview of different types of interaction among criteria that could exist in the decision making problem. Three kinds of interaction defined and described in [58] are as follows: correlation, complementary, and preferential dependency.

Correlation can be further divided into positive correlation and negative correlation. Positive correlation is existent two or more criteria present some form of redundancy. For example, consider again the problem of evaluating a given car based on three criteria {fuel efficiency, luxury, price}. A highly luxurious car generally comes with a higher cost. In this case, luxury and price form positive correlating criteria, and the evaluation will be an overestimate. As discussed before, this problem can be overcome by using weighting factor on subset of criteria, such that $w(ij) < w(i) + w(j)$, where i and j are two criteria and sub-additive feature overcomes the overestimate during the criteria evaluation. In the reverse case (negative correlation), weighting factor $w(ij)$ will be super-additive given by $w(ij)>w(i) + w(j)$.

In complementary type of interaction, one criterion can replace the effect of multiple criteria. This means that importance of criteria pair (*ij)* is close to the importance of having single criterion *i* or *j*. Clearly, when such criteria pair exists, a weighted sum cannot be helpful during the evaluation process. A more complex weighting factor needs to be considered.

The third type of interaction is the preferential dependence. In this type of interaction, the decision maker's preference for selecting an alternative is simply given by a logical comparison, i.e., if there exists a function *M* such that, for any two alternatives *a1* and *a2*, then the decision maker selects one of the alternatives (say *a1*) if *M(a1)>M(a2).*

Clearly, when such complex interactions exist among criteria, it is necessary to use a well-defined weighting function on subset of criteria rather than a single criterion during global evaluation. One such methodology for evaluation is Choquet integral with the use of fuzzy measure [57] as weighting function.

4.3 Fuzzy Measure and Choquet Integral

A fuzzy measure [59] on a set of criteria (X) is defined as a mapping function μ: $2^X \rightarrow [0,1]$, where 2^X is the power set of X. Additionally, μ should satisfy the following properties:

1. $\mu(Ø) = 0$ and $\mu(X) = 1$, where Ø represents the null-set
2. If A is a subset of B, then $\mu(A) \leq \mu(B)$

For example, consider a set X = { x_1, x_2 }. Power set of X is given by, P(X) = { Ø, {x1}, {x2},{x1,x2}}. The fuzzy measure on the elements of set P, for example, can be defined as: $\mu(\emptyset) = 0$, $\mu(\{x_1\}) = 0.4$, $\mu(\{x_2\}) = 0.5$ and $\mu(\{x_1,x_2\}) = 1$. If μ is the fuzzy measure on X (set of criteria), then Choquet integral [60] of a function $f : X \rightarrow [0,1]$ with respect to μ is defined as:

$$C_\mu (f(x_1)...f(x_n)) = \sum_{i=1}^{n} (f(x_{(i)}) - f(x_{(i-1)})) \times \mu(Y_{(i)}) \qquad (10)$$

where $x_{(i)}$ indicates that the indices have been permuted such that $f(x_{(1)}) < f(x_{(2)}) <\dots < f(x_{(n)})$ and $Y_{(i)} = \{x_{(i)}, \dots, x_{(n)}\}$. If the fuzzy measure μ is additive (i.e. $\mu(xy)= \mu(x)+ \mu(y)$), then C_μ represents discrete Lebesgue integral [61]. The above equation (10) for discrete Choquet integral can also be given as:

$$C_\mu (f(x_1)...f(x_n)) = \sum_{i=1}^{n} f(x_{(i)}) \times (\mu(Y_{(i)})-\mu(Y_{(i+1)})) \qquad (11)$$

A graphical representation of Choquet integral as compared to other aggregation operators in the interaction space is given in figure 4.1 [62].

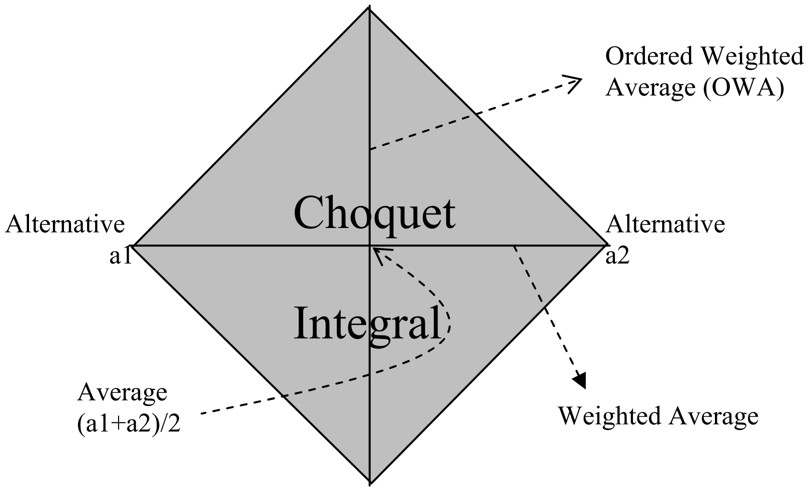


Figure 4.1 Graphical Representation of Choquet Integral

4.3.1 Sample Example

We propose a case study for multi-criteria decision making in mobile robot path planning in an environment deployed with sensor nodes. We can generalize such a decision making process to a more complex system management. We develop an efficient data collection and sensor node replacement scheme for sensor network in a cluttered environment. The autonomous sensor nodes embedded in the environment are generally low powered devices. High events in the environment usually require constant monitoring and dense deployment for precisely localizing the threat events. In order to capture all important events, we would ideally want more nodes deployed in the region of event compared to other regions in the environment. Any dying nodes would also require a replacement (redeployment) in order to sustain the lifetime of entire network. This is a novel methodology for a mobile robot to collect data, replace any dying node and to deploy more nodes in the region of higher events.

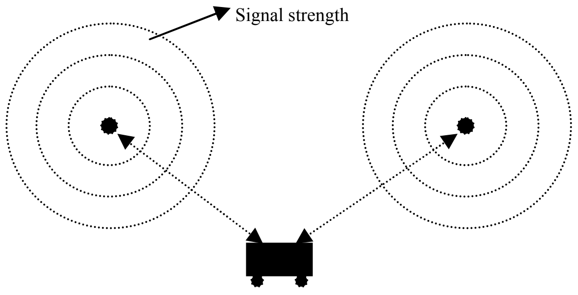


Figure 4.2 Basic Robot-Sensor Architecture

Our initial premise to use radio frequency (RF) signal strength alone to determine distance to node was inadequate in providing high data integrity for the following reasons:

Consider an analog test signal being transmitted from the robot to a node as show in figure 4.2. The amplitude (signal strength) of the returned signal detected may be true or may be the result of weak battery power. If multiple analog signals are being transmitted from mobile robot to several nodes, the algebraic addition of these signals may provide an erroneous reading. Hence signal strength or amplitude detection by itself can only be used as supplementary information in determining the distance to node. An easier implementation is to send out a synchronized pulse from the robot and receive the returned pulse from the sensor node and determine the travel time of pulse. In essence, this is similar to the functionality of a sonar rangefinder. Another way of determining distance is to send out a predetermined beacon signal with node ID. The robot can determine the distance by looking at any two consecutive beacon signals. These signals can be directly generated by the battery. This is an added advantage since the beacon signal in addition to providing distance-to-node information also provides a relative reading of the battery power of the node.

The decision making problem for the robot is to efficiently navigate through the sensor field to reach all the nodes. In the event of multiple paths available to the robot, the robot path planning algorithm would intelligently decide which node to reach first. The robot is challenged with equally “important” paths to navigate in order to fulfill its goal. The goal is to collect data and/or to deploy a node. With advanced technology, robots maybe able to even recharge the battery on the sensor node. However, due to low cost in sensor node construction, we assume it is economical to redeploy a node instead of recharging the battery. The importance of a given path is based on several parameters relating to the sensor nodes in the field. Given a deployed embedded network of sensors, the task of the robot is to reach the sensor nodes based on several competing criteria. For example, a sensor could have critical data that needs to be collected. At the same time, another sensor node may be dying due to low battery power, requiring immediate attention.

We formulate the above problem by defining the set of criteria, alternatives and the goal as follows:

Criteria: X = $\{x_1, x_2, \ldots x_n\}$ – set of criteria

X = {distance, battery power, event level, data criticality}

Alternatives: **Ω** = $\{s_1, s_2, \ldots s_m\}$ – set of systems on which criteria is to be evaluated

Ω = set of sensor nodes

Goal: Evaluate the set of systems/alternatives $\{s_1, s_2, \ldots s_m\}$ based on set of criteria $\{x_1, x_2, \ldots x_n\}$.

G = Select a sensor node to be reached first.

The criteria and alternatives are organized in a tabular fashion as shown in table 4.1. Distance represents how far the node is to the base-station or the robot. Battery power represents the voltage remaining in the sensor node's battery. Event level signifies the number of events captured over a small period of time. A simple way to represent criticality is to look at the threshold of the sensed value. Generally, if the sensed value is beyond the set threshold, the criticality will be high.

Table 4.1 Evaluation of Alternatives

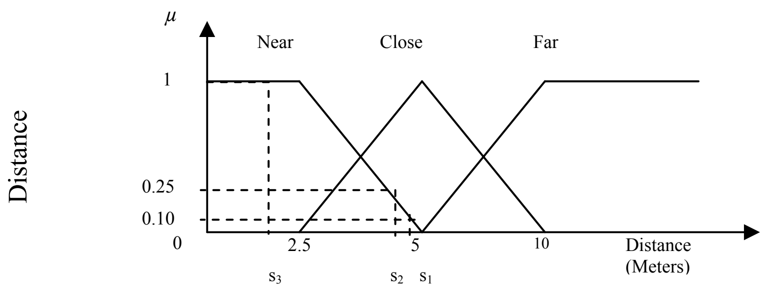
Criteria x / **Sensors** s	Distance	Battery Power	Event Level	Data Criticality	**Evaluation**
sensor *1* (s_1)	d-1	b-1	e-1	cr-1	**C-1**
sensor *2* (s_2)	d-2	b-2	e-2	cr-2	**C-2**
sensor *3* (s_3)	d-3	b-3	e-3	cr-3	**C-3**
…	…	…	…	…	**…**
sensor *m* (s_m)	d-*m*	b-*m*	e-*m*	cr-*m*	**C-*m***

C-1…C-*m* in Table 4.1, are evaluation results based on the current criteria and interaction among the criteria. The methodology used to obtain C-1…C-*m* is by using Choquet integral. A simple pair-wise comparison between two evaluation items can help to determine the *preference* for selecting a particular system (sensor node). For example, if C-1 > C-2, then sensor s_1 is preferred over s_2.

Consider a mobile robot traversing in an environment that is covered with embedded sensors. At each predetermined discrete time interval, the robot evaluates which sensor node to reach first, based on set of criteria X. We identify two different cases for efficient evaluation of three sensor nodes.

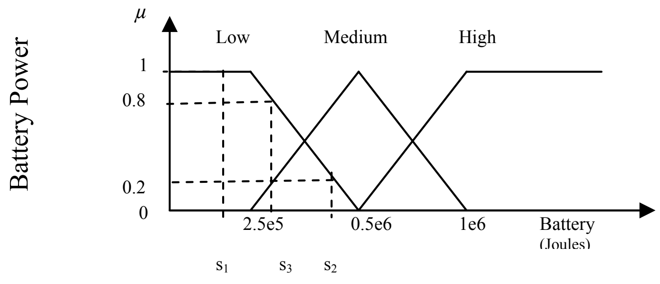
Case 1: Criteria are fuzzy variables without interaction

The goal of the decision maker is to select a node that is nearest (low distance value), has low battery power, and has high events registered. The alternatives are three nodes (s_1, s_2 and s_3) to be evaluated. We define the following fuzzy membership function for each criterion:



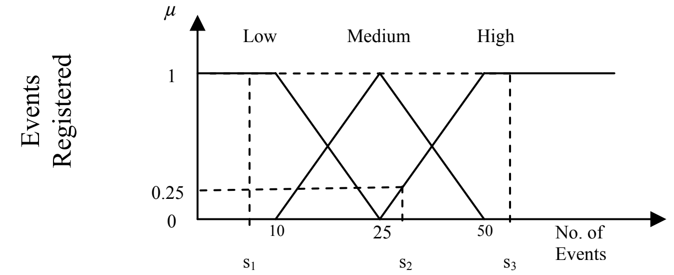
Condition – nearest distance
$C_1 = 0.1/s_1 + 0.25/s_2 + 1/s_3$

(a)



Condition- Low battery
$C_2 = 1/s_1 + 0.25/s_2 + 0.8/s_3$

(b)



Condition- High Events
$C_3 = 0/s_1 + 0.25/s_2 + 1/s_3$

(c)

Figure 4.3 Fuzzy Membership Functions for Criteria (a) Distance, (b) Battery Power and (c) Number of Events

C_1, C_2 and C_3 are the fuzzy sets obtained which expresses goal and conditions in terms of available systems s_1, s_2 and s_3.

The decision maker's solution (D) is obtained by **max-min** inference [63] on the three sets C_1, C_2 and C_3. D is obtained from **min** of each system and represents a fuzzy characterization of the concept of desired system. Using **max**, we can obtain a preference of a given system over another system. In this case, sensor node s_2 is the most desired system to be reached first by the robot.

$$D = 0.1/s_1 + 0.25/s_2 + 0/s_3 \qquad (12)$$

Case 2: Criteria are crisp variables with interaction

As discussed before there are three types of interaction among criteria identified - *correlation*, *complementary* and *preference dependency* as three different forms of interaction among criteria. In our case study on sensor network, criteria such as power level and capturing events are correlated and complementary. For example, in order to capture critical environmental events, a deployed sensor should ideally have a low sleep-time and high sampling frequency. This means that power consumed by the sensor is high, suggesting that power consumption and events are correlated and complementary.

Recall that Choquet integral is defined over the function $f: X \rightarrow [0,1]$. This function f is often called the *utility function* or *score* [64]. The utility function is required to make the criteria comparable, since criteria generally are not measured on a common scale. By using utility function we map the criteria to a common scale, making them *commensurable criteria* as shown in figure 4.4.

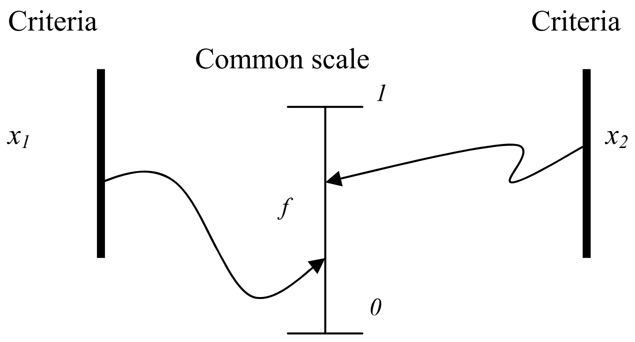
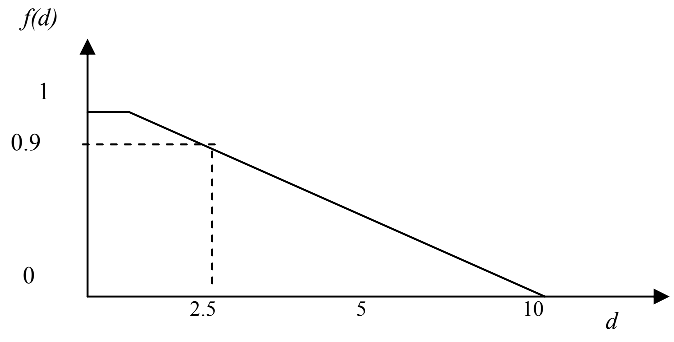
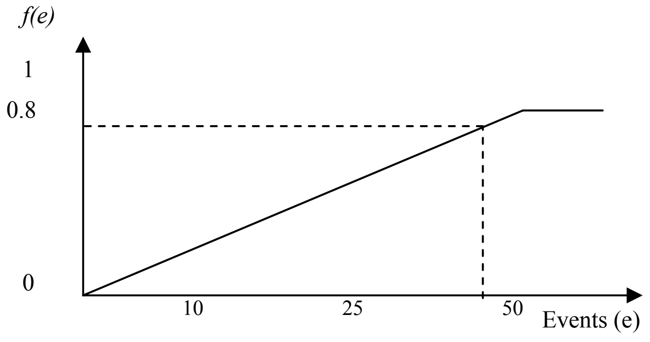


Figure 4.4 Mapping Criteria

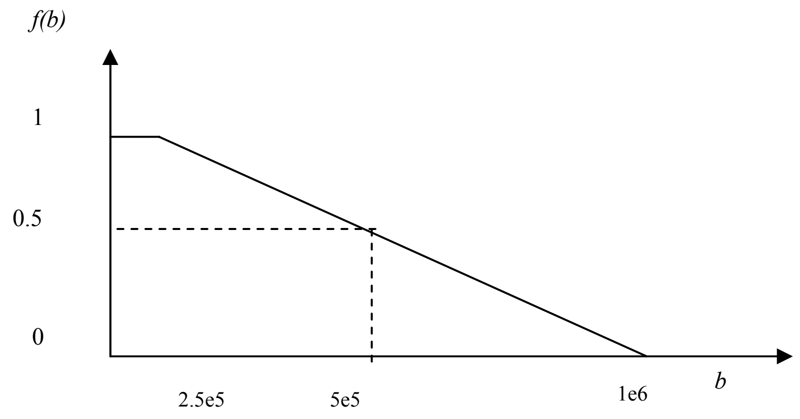
Given the three criteria/attributes related to a sensor node – distance, events registered, and battery power, we can generate the utility function based on the defined goal as follows:



(a)



(b)



(c)
Figure 4.5 Generating Utility Functions for Distance, Events and Battery

From figure 4.5, for example, a shorter distance to a given node, generates a high score or utility function. For example, a distance of 1m, will generate a score *f(d)*=0.9. Similarly, if the number of events generated is high (say 40), then the score is high (*f(e)*=0.8). The overall evaluation of different alternatives (sensor nodes) is obtained by aggregating the utility functions using Choquet integral with appropriate fuzzy measure (which acts like a weighting factor). The weights, fuzzy measure and resultant Choquet integral for three sensor nodes at varying distances, battery level and event (activity) level are tabulated as given below:

Table 4.2 Fuzzy Measures for Subset of Criteria

Sets	Fuzzy Measure
{}	0
{Distance}	0.854756
{Battery}	0.515547
{Distance,Battery}	0.978599
{Events}	0.164453
{Distance,Events}	0.89426
{Battery,Events}	0.604638
{Distance,Battery,Events}	1

Table 4.3 Input and their Corresponding Choquet Integrated Values

No.	Distance	Battery	Events	Choquet Integrated Values
1	0.9	0.5	0.1	C_{-1} = 0.833342
2	0.5	0.9	0.1	C_{-2} = 0.697658
3	0.1	0.1	0.9	C_{-3} = 0.231562

A working example for the real-world experiments conducted is presented in chapter 6.

4.4 Chapter Summary

The problem of sensor behavior assignment is defined as an efficient planning process for determining the sensor functions and usage according to changing situations. Two important processes involved in the behavior assignment are, 1) decision about set of tasks that sensors need to accomplish and 2) scheduling of actions for the sensors. We believe that decision making process is the hardest and important step in behavior assignment. This is because, once the decision is made on what tasks that sensor needs to be doing, scheduling actions for that decision can be implemented simply as a look-up table. The decision making process on what tasks the sensor needs to accomplish depending on the mission plan and situation generally depends on the various criteria involved. Once the behavior pattern for a given application is identified, the state of the sensor network and its performance can be used as feedback for creating training set for learning algorithms such as neural networks.

The above mentioned decision making process for mobile robot can be adapted to cluster of sensor nodes rather than individual sensor node. For example, based on activity level, number of sensor nodes and importance of activity, preference can be given to a particular cluster for management (power management, node density management, etc.). If the number of nodes is critical for the given application, then it gets a high weighting factor. This means that we would require some nodes to be put to sleep. We are thus changing the behavior of the network by tuning one of the parameters (increasing sleep time) based on the needs of the application. We are intelligently analyzing the characteristic of the deployed sensor network for a given application and adaptively changing its operational behavior to suit the changing demands. From decision-theoretic viewpoint, appropriate sensor action needs to be scheduled in order to achieve maximum utility.

CHAPTER 5

DYNAMIC POWER MANAGEMENT

An important consideration when dealing with sensor networks is power consumption. Generally, the sensor nodes are driven by a limited power supply on-board the node (i.e., the network has distributed power supply). The impact of conserving power on each sensor node can have tremendous effects on the lifetime of the entire network.

There are two main categories that can be identified to sustain the lifetime of sensor network [65]:

Global level or system-wide: Increase the number of redundant sensor nodes. These redundant nodes act as back-up nodes and can take over the task of sensing and signal communication from any dying nodes in order to sustain overall network lifetime.

Local level: Scheduling and low power operation of each individual sensor node.

By adjusting either network parameter (increase in number of nodes) or node parameter (power scheduling), we can sustain the lifetime of the given sensor network. Whenever such a parameter adjustment is performed, the behavior of the network changes and a sensitivity analysis can be performed to evaluate the behavioral changes. At the sensor node level, in order to conserve battery power, the sensors can be scheduled to sense the environment at different samples or time intervals. Although, some information might be lost, this is an effective way to optimize energy consumption. Figure 6 shows the ON/OFF (sleep mode) scheduling of the sensors.

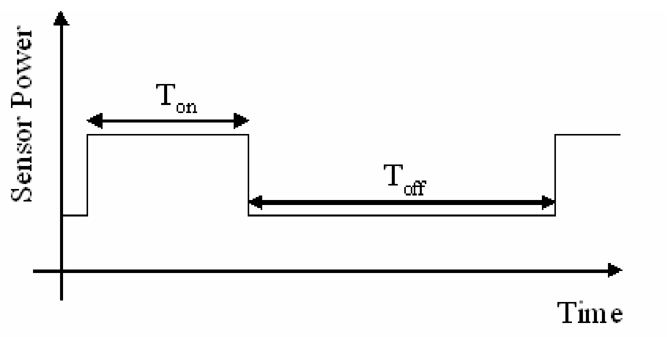


Figure 5.1 Typical Time Scheduling

The energy consumed E_c by each sensor is given by:

$$E_c = \sum_{i=1}^{n} T_i P_i \tag{13}$$

where T_i is the time period and P_i is the power at which the sensor operators in the given time period T_i. This is a simplified energy consumption model [66]. The network lifetime is the reciprocal of the energy consumed. Let the sensor work in full power, i.e., $P_i = P_f$ without any scheduling and T_{L1} be the network lifetime based on the energy consumption E_{c1}. If the sensor is scheduled (changing input parameter), then the energy consumed after scheduling E_{c2} will be less than E_{c1}, suggesting that lifetime T_{L2} will be greater than T_{L1}. The sensitivity measure is given by:

$$S_{P}^{E_c} = \frac{\partial E_c}{\partial P_j} = T_j \tag{14}$$

During the T_{off} period, the sensor could be completely put to sleep (meaning zero power consumption) or it could work at a lower power. Commercially available sensor nodes such as Crossbow motes generally operate through a cycle of modes in order to reduce the energy consumption – 1. Sleep 2.Wake-up 3. Sample sensor reading (read ADC port) 4. Communicate 5. Go back to sleep mode. This is an in-built power scheduling mechanism depicted in figure 5.2.

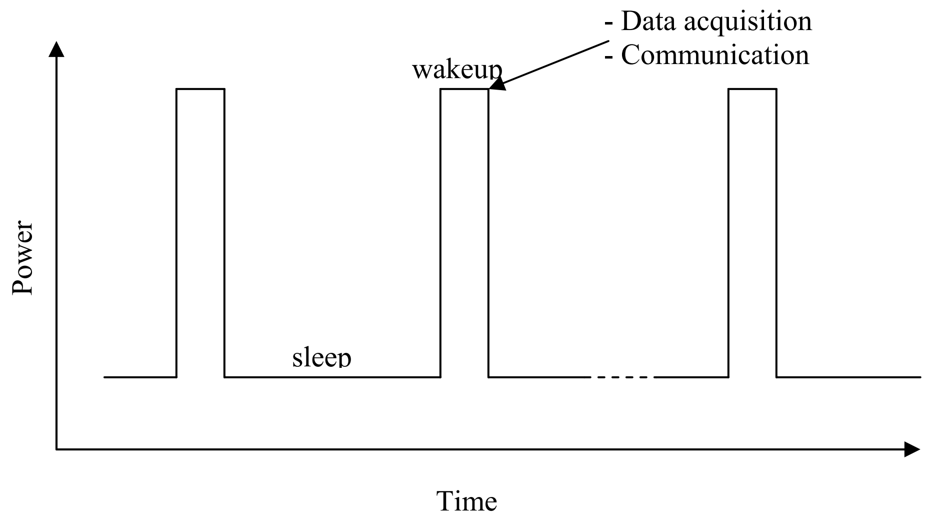


Figure 5.2 Sensor Node Operation Cycle

5.1 Motivation

One of the objectives of a sensor network with on-board batteries is to survive as long as possible and derive meaningful feature level information from the environment. The overall effectiveness of the sensor network depends on how well the mutually contradicting objectives of conserving the limited on-board battery power and keeping the sensors awake for stimuli, are managed. The sensor nodes should ideally sleep as much as possible; however, it should be able to capture high number of events. "Sleep" here means that the sensor node's radio, sensors and EEPROM (memory) are turned off and the processor is in an idle state. The processor can wake-up after the timer expires and acquires data from the ADC (Analog-to-Digital Converter) ports of the sensors. In order to sleep as much as possible but still able to capture events, we will need an adaptive technique that not only depends on the change in sensed value but also on the degradation of the battery power. The node should be "smart" enough to adaptively adjust its sleep time based on these two conditions – temporal difference in sensed value and current battery state of the node.

In order to build such "smart" sensor node, a rigorous learning process should guide the node to evaluate multi-objective decision making. Due to the high spatial distribution, low computation and energy capabilities, WSNs often pose a challenge to classical machine learning. The concept of supervised learning has been extensively used in object/target tracking and detection in sensor networks. Reference [67] and references therein, provide an excellent survey of existing supervisory learning methods and provide models for nonparametric approach to distributed inference in WSN. Although, these literature reveal a great insight into distributed learning, and in specific to distributed inference in energy and bandwidth challenged environment, very few directed research have implemented reinforcement learning in sensor networks. The application of reinforcement learning specific to sensor networks have only been researched mostly for routing information from sensors back to a base-station [68-69]. [70] gives basic concepts of learning theory approach in sensor networks based on several specific sensor network applications. Other discussions on learning (such as in [71]) have been specific to detection and classification using sensor networks. In this chapter, we will present an implementation of a multi-objective critic based autonomous decision making mechanism that takes both the temporal state transition of the sensor and that of the environment into account. Although, the temporal based reinforcement scheme proposed can be used to adaptively change several behaviors of a given sensor, for experimental purposes, we consider only one such behavior (sleep time) for power management. This experimentation will not mask the generality of the theory proposed. Specifically, we propose an *actor-critic* based reinforcement learning mechanism that can be practically implemented on an embedded sensor. The key to such reinforcement learning mechanism is the development of the value function (or critic/reinforcement function) that is implemented on each sensor node which aids in dynamic power scheduling based on different situations. In the next section, we introduce the concept of reinforcement learning, the need for reinforcement learning and our approach to solve adaptive power scheduling scheme problem with such learning mechanism.

5.2 Reinforcement Learning

Reinforcement learning is different form of supervisory learning [72]. Reinforcement based learning is adopted when there is no feedback available in the action space of the learning agent. For instance, in the case of a neural network [73] that maps a given function; supervised learning can be applied if the input and output data pairs are available, so that the estimated output of the neural network can be compared against the desired output. In situations where the desired output is not available, the network can be trained using reinforcement based learning if the output of the network can be evaluated in terms of a reward or a penalty. Neural networks and other regression model [74] are computationally too intensive to be implemented on low cost sensors.

In the case of a sensor learning to make internal decisions based on proprioceptive information, there is no set goal against which the decisions can be compared. Therefore, an error cannot be calculated in the decision space. Alternatively, the decisions can be evaluated in a contextual sense to derive a scalar reward. The internal decision making policy can be improved by making it pursue a strategy to maximize total expected rewards.

Reinforcement learning deals with how to map the situation to actions. The learner does not have the knowledge of what actions to take, but instead selects an action that will yield maximum reward (or minimum penalty). For example, a mobile robot is required to map a given building. It should decide whether to enter a new room for mapping or go back to docking station for battery re-charge.

There are three critical elements in reinforcement learning – a *policy*, a *reward function* and a *value or critic function*. A *policy* defines the learner's behavior at a given time. Simply, policy maps the observation into actions. Policies can be stochastic or deterministic in nature. *Reward function* maps the action into a scalar value. The learner either gets a reward or a penalty for taking a certain action. It is the learner's responsibility to maximize the reward. *Value or critic function* what is good for the learning agent in a long run. Critic (V) at a given time *k,* is the total expected reward given by:

$$V(k) = r(k) + \gamma r(k+1) + \gamma^2 r(k+2) \ldots \qquad (15)$$

where, r is the instantaneous reward function defined by the user and γ is the discounting factor. In conventional reinforcement based algorithms, a critic learns to estimate V(t) as shown in figure 5.3.

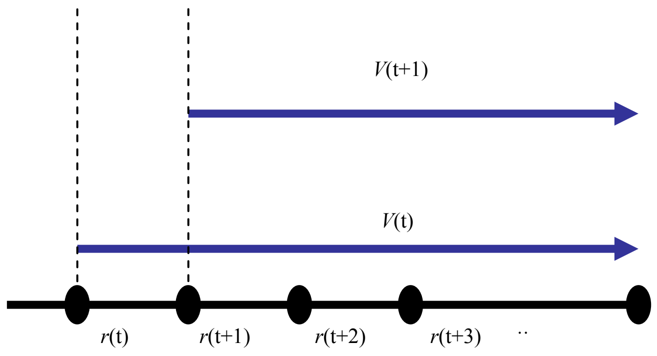


Figure 5.3 Relationship between Rewards and Critic

A major class of reinforcement learning is the *Temporal Difference* (TD) learning scheme. Like Monte Carlo methods [72], TD learning can directly learn from experience without having to know the underlying model of the environment. In TD approach, the learning agent passively observes a temporal sequence of inputs that eventually lead to final reward [75]. The learning agent's main task is to then predict expected reward. One such TD reinforcement method is *actor-critic* learning [76]. Figure 5.4 shows the architecture of the actor-critic method.

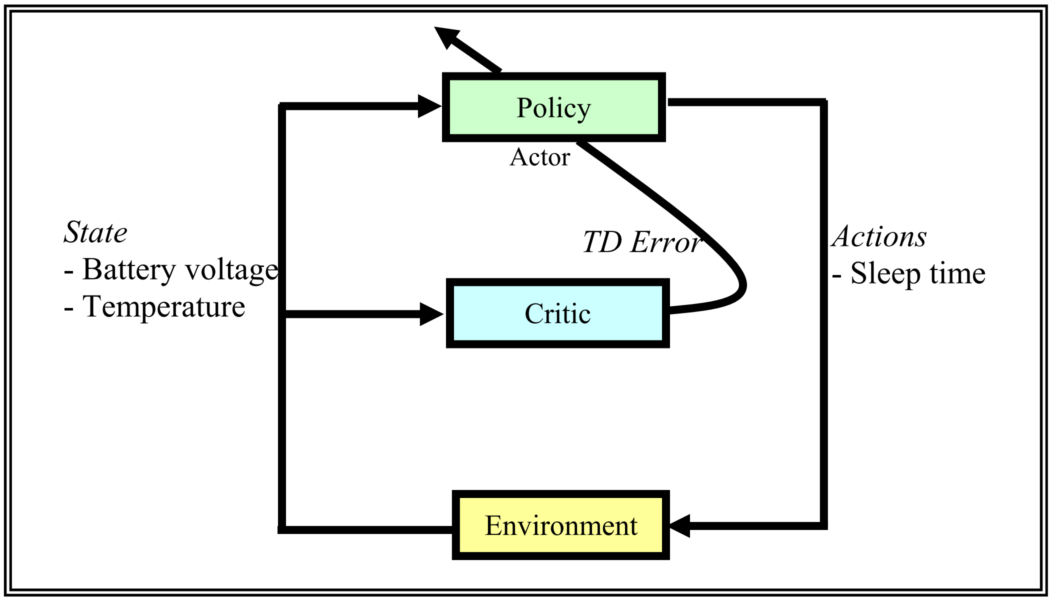


Figure 5.4 Actor-Critic Architecture

The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. Learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. The actual critic is a state-value function which gives the total discounted sum of future rewards given by:

$$V(k) = \sum_{m=0}^{6} \gamma^{m} r(k+m),\ \gamma = 0.6 \qquad (16)$$

We restrict the polynomial to 6, since $(0.6)^6$ is a small number.

5.2.1 Our Approach

The computational burden involved rules out the possibility to implement the estimation algorithm (V(k)) on a commercially available embedded sensor. In this case we reduced the computational burden by directly designing or estimating a critic function given by:

$$\begin{aligned}&\hat{V}(k) = \phi^T(k)\theta(k)\\&\varepsilon = V(k) - \hat{V}(k)\\&\theta, \phi \in \Re^N\end{aligned} \tag{17}$$

where, $\hat{V}(k)$ is the estimated discounted sum of future rewards, $\phi(k) = [r(k) \quad r(k-1) \quad \cdots \quad r(k-N+1)]^T$ is a vector of past rewards, $\theta(k)$ is a vector of scalar parameters of the same size as $\phi(k)$. Here, we assume that the non-linear dynamic behavior of the sensor and the environment can be approximated by a non-linear static reward function given in (18) and a linear dynamic regression function given in (17). The advantage of this method is that any number of non-linear evaluation criteria can be integrated into the reward function in (18). The simplified linear dynamic regression function can be easily implemented on an embedded sensor with limited processing and memory capacity. A more comprehensive approach where the nonlinearities are modeled is by a neural network or a nonlinear regression model. However, this would be computationally intensive to be implemented for low cost sensors.

The instantaneous reward function r is given by:

$$r(k) = \left(\frac{b(k)}{b_{\max}}\right) \times \left(\frac{S_{current}}{S_{avg}}\right)^{\tau \times |(T(k+1) - T(k))|} \tag{18}$$

where, $b(k)$ and $b_{\max}$ are battery voltage at time k and that at full charge respectively, $S_{current}$ and S_{avg} are sleep time at time k and a scalar value representing a mean sleep time respectively, τ is a scalar, and $T(k)$ is environment temperature at time k. The design of this reward function is based on the fact that, the sensor should sleep less to capture events (change in temperature). However, when the battery voltage is running low, the sensor should sleep more but still be able to capture the changes in temperature.

Our approach to reinforcement learning method is in two phase as follows:

1. Offline critic function estimation: Based on data set obtained with varying temperature and sleep times, we evaluate our instantaneous reward function (given in (18)). In order to estimate $\hat{V}(k)$, we will need to first estimate the parameter vector $\theta(k)$. A recursive least squares algorithm given in (19) was used to optimize the parameter vector.

$$\begin{aligned} &\theta(k) = \theta(k-1) + P(k)\phi(k-1)\left[V(k) - \phi(k-1)^T \theta(k-1)\right] \\ &P(k) = P(k-1) - P(k-1)\phi(k-1)\left[1 + \phi(k-1)^T P(k-1)\phi(k-1)\right]^{-1} \phi(k-1)^T P(k-1) \\ &P(k) \in \Re^{N \times N} \end{aligned} \tag{19}$$

For a polynomial of order 4, the estimated critic given in equation (16) was optimized using the recursive least squares algorithm given in equation (19). The resulting vector of polynomial parameters obtained was: $\theta^* = [1.4768 \quad 0.3052 \quad 0.2688 \quad 0.3464]^T$. Figure 5.5 shows the difference between estimated critic and the actual critic. This is first attempt to estimate the critic using a linear polynomial function. Figure 5.6 shows that polynomial order 4 gives the minimum average estimation error $\varepsilon = \sum_{k=1}^{T} \left\| \left(V(k) - \hat{V}(k) \right) \right\|$, where T is the total time span.

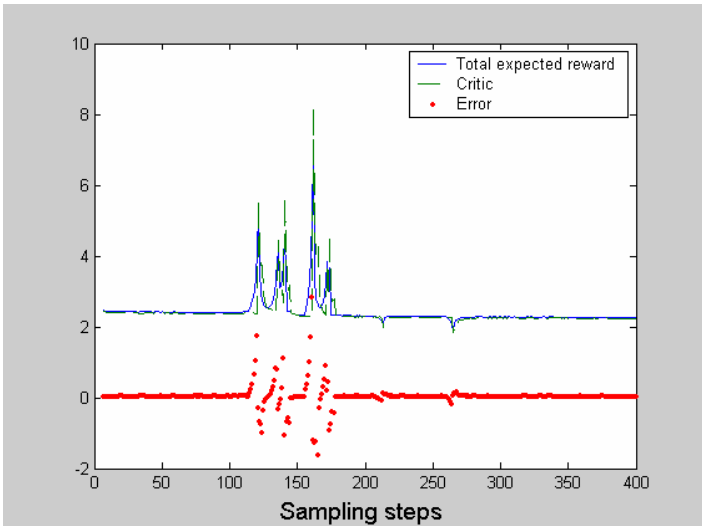


Figure 5.5 Comparison of approximated critic with the actual discounted sum of future rewards

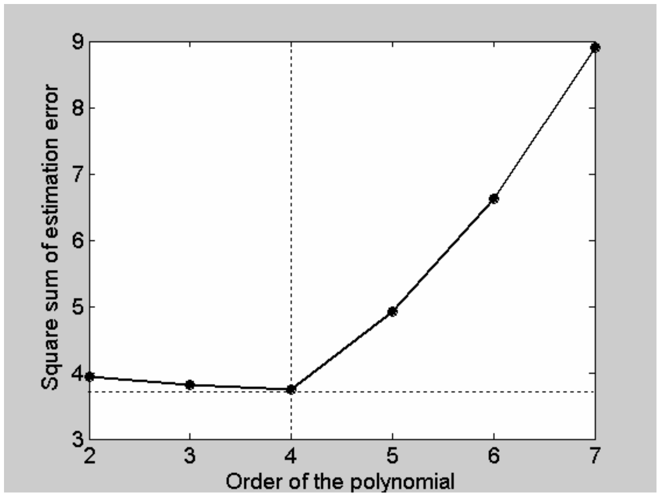


Figure 5.6 Estimation of Polynomial Order

The offline batch learning (or critic estimation) was conducted with varying sleep times and with changes to temperature reading. Instantaneous reward, critic, sleep time and temperature reading from real-world sensors are as shown in figure 5.7

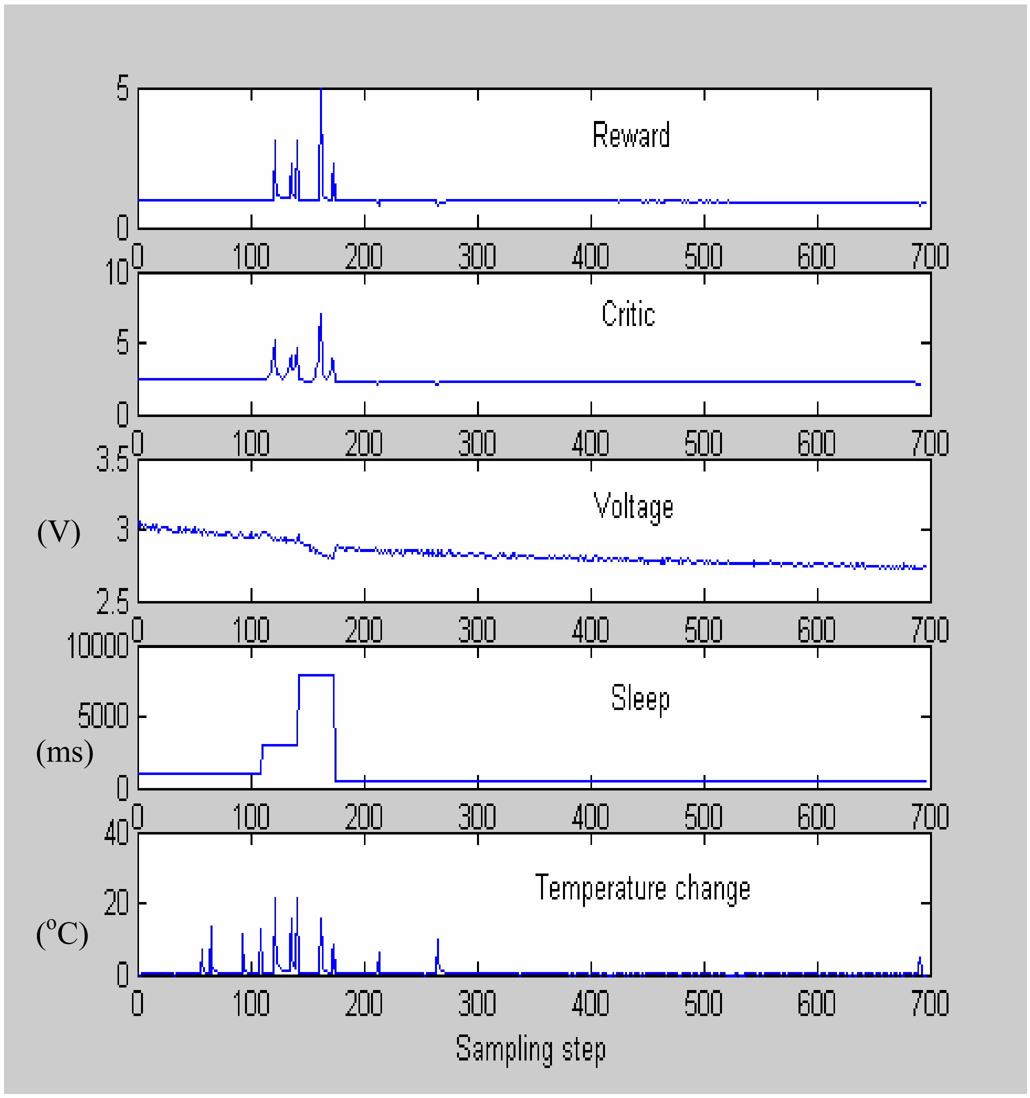


Figure 5.7 Evaluation of Reward and Critic

From figure 5.7, we see that there is an high reward when temperature changes and the sleep time is low. This means that the sensor is awake and is able to capture the events. Similarly, the reward is low with temperature changes when the battery voltage is low.

2. The second phase of our algorithm is to program the sensor node with the estimated critic. The estimated critic function is loaded on-board a sensor node. With varying temperatures, the sensor node adaptively changes the sleep time. For each reading,

we calculate the reward function given in equation (4). We store up to four reward values in a circular buffer. After every fourth reading, we estimate the critic value and compare it with the older critic estimate. If the present critic estimate is greater, we adaptively change the sleep time and calculate the next reward value.

Detailed experimentation and results are shown in chapter 6 of this thesis.

5.3 Chapter Summary

In this chapter, we have presented a novel approach in designing a critic function that will guide the sensor to adaptively sleep so as to reduce the network packets as well as conserve on-board battery power. The node sleeps as much as possible but at the same time should handle the stimuli from the environment. This complex contradicting requirement is embedded in a reward function that is developed and implemented as shown in this chapter. Moreover, this is first attempt to use a polynomial type critic function which approximates non-linear regression model and that can be implemented on a low power (computation/memory) platforms.

CHAPTER 6

DESIGN AND IMPLEMENTATION

In this chapter, we will study in detail the complete design to deal with multi space problem domains in sensor networks. Specifically, we consider implementation details of the design for each sub-problem – data aggregation, critical monitoring and control and power scheduling that has been discussed in previous chapters. The solutions proposed for each of the sub-problems in the previous chapters can be considered as an overall architecture for fault-detection and performance improvement in sensor networks. A block diagram illustrated in figure 6.1 shows the concept of such architectural design.

The architecture presented in figure 6.1 is generic in nature and can be applied to any sensor acquisition system that requires some degree of fault-tolerance and performance improvement. In chapter 2, we proposed a detailed theory on efficiently handling large data sets through hierarchical aggregation. The concept of spatial correlation is used to validate the aggregation process against intermittent faults. Chapter 2 also described built-in test methods adaptively calibrate the sensors to alleviate faults. These processes which are usually performed in-network are incorporated into the complete solution suite described in figure 6.1. Often times, the quality or the process itself might have to be compromised against several parameters such as battery lifetime, node storage capacity, bandwidth, etc. in order to achieve high performance throughput. These algorithms that trade the execution of the process or its quality are often referred to as ***adaptive fidelity*** algorithms. The decision to run a process (such as for example - aggregation, built-in test) depends on the parameters (criticality, battery, etc) and extensive decision making. These decision making process (discussed in chapter 3 and 4) running on base station as shown in figure 6.1, helps us to fine-tune the operational characteristics of each sensor node.

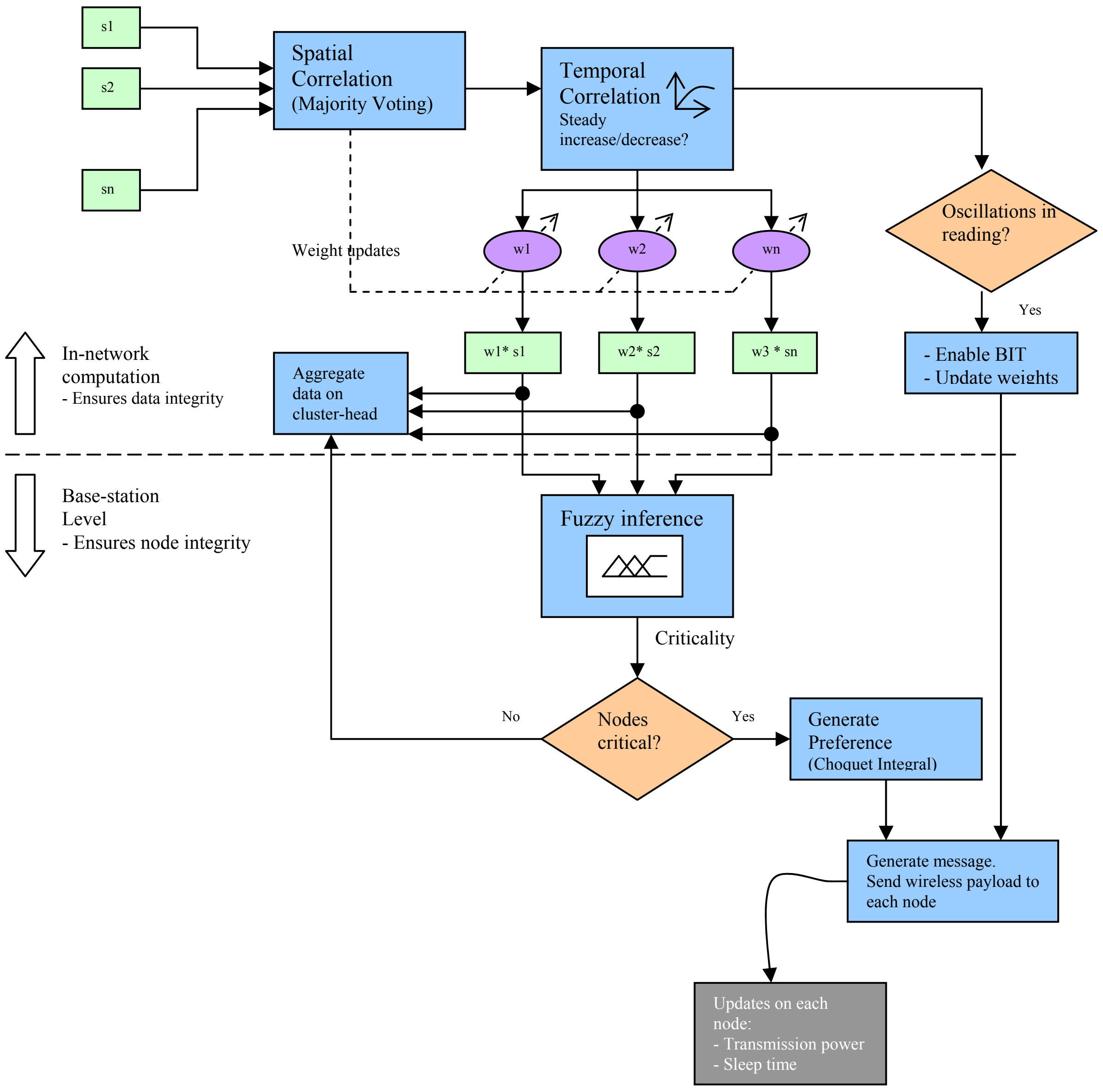


Figure 6.1 Architecture for Performance Improvement in Sensor Networks

6.1 Detailed Description

The block diagram shows the flow of data and control for ensuring complete network integrity. In order to provide such integrity, both node and data integrity should be ensured. We can distribute the functionality shown in the architectural design onto nodes as well as to the base station. Extensive computation load should be handled by base-station which is assumed to have higher computation, power and storage capabilities compared to sensor nodes. Critical, faster, and less expensive computations should be handled at a node or cluster-head level. Generally, data integrity issues are very critical that needs to be handled at the node level. For example, consider a scenario that requires continuous sensor data acquisition. This will generate large amounts of data, thereby affecting the performance of post-processing of data. If the sensor node filters out any redundant (or unwanted) data at acquisition phase, this will greatly influence the performance of data post processing. In fact, a robust method is to aggregate data or compress data (as discussed in chapter 2) rather than throwing the data away. Therefore, in our proposed method, we perform in-network data level integrity (at a given sensor node or cluster-head)

In simple terms, data integrity means ensuring that the data acquired is complete. Also, the aspects that influence data integrity are correctness, accuracy and validity. These aspects suggest that the data acquired and processed should have minimal faults (or ideally, be fault-free). Thus, the theory proposed in chapters 2 will enable us to provide a complete fault-tolerant data acquisition with validation from neighboring sensor nodes.

Feature extraction, pattern classification and other decision theoretic approaches often require high computation power. This would be ideal at a base-station level. This data also reveals important information on the operation status of the node. The decisions taken based on some parameter (in this case, criticality of node) at the base-station are propagated to cluster-heads and eventually to all the sensor nodes. Based on the information received, necessary changes are made to the operating parameters of the sensor nodes. In order to achieve such decision propagation, an efficient protocol and message structure needs to be designed. This chapter will give a detail explanation of the design and implementation of such protocol, the hardware platforms used and the implementation of high-level interface for post-processing the acquired data.

6.2 Hardware Platform

In order to implement the design proposed in the previous section, we use off-the-shelf multi-sensor board (MTS420) from Crossbow Inc. running TinyOS. For this thesis, we have used TinyOS version 1.1. Each of these sensor boards is equipped with different sensors – temperature, humidity, pressure, light, and 2-axis accelerometer. The multi-sensor board is housed on a platform (MICA2) that has a processor-radio board (MPR400) and other accessories (such as antenna and connectors for sensor board). Processor on-board MPR400 is an Atmel ATMega128L 8-bit ARM processor with 7.37MHz clock speed. It has a 128KB program memory and 4KB EEPROM for data. The combination of multi-sensor board and the platform (MICA2) is usually termed as *motes.* Motes are modular in nature, i.e., a platform can house different but compatible multi-sensor boards. We used a 433MHz multi-channel transceiver for our motes, since it provided a very good range (distance). We use temperature sensor (Sensirion SHT11 temperature/humidity sensor) onboard the MTS420 sensor board to collect data for experimentation. Operating temperature range is -40^{o}C to +125^{o}C. Experiments were conducted both in a laboratory setting as well as in outdoors. The sensed information (temperature in this case) is sent wirelessly to the gateway node, which is just another mote that is housed on a programming board (MIB520). The programming board connects to a PC (base-station) using a serial or Universal Serial Bus (USB) interface using Universal Asynchronous Receiver Transmitter (UART) packets. As specified in the user manual [77], in TinyOS 1.x, UART packet format is platform-specific (say MPR400), which requires complex protocol handling and PC-side tools to decipher and handle the messages to and from the motes. Since this is true in our case (since we are using TinyOS 1.1), we had to develop interface tool at the PC-side to decipher the packet. The interface not only handles packets from serial (or USB) interface, but also allows the user to inject packet back into the gateway node, which then broadcasts the message wirelessly to all the deployed sensor nodes. We discuss the details of protocol design for message passing from PC to gateway node and vice versa in section 6.3 of this chapter.

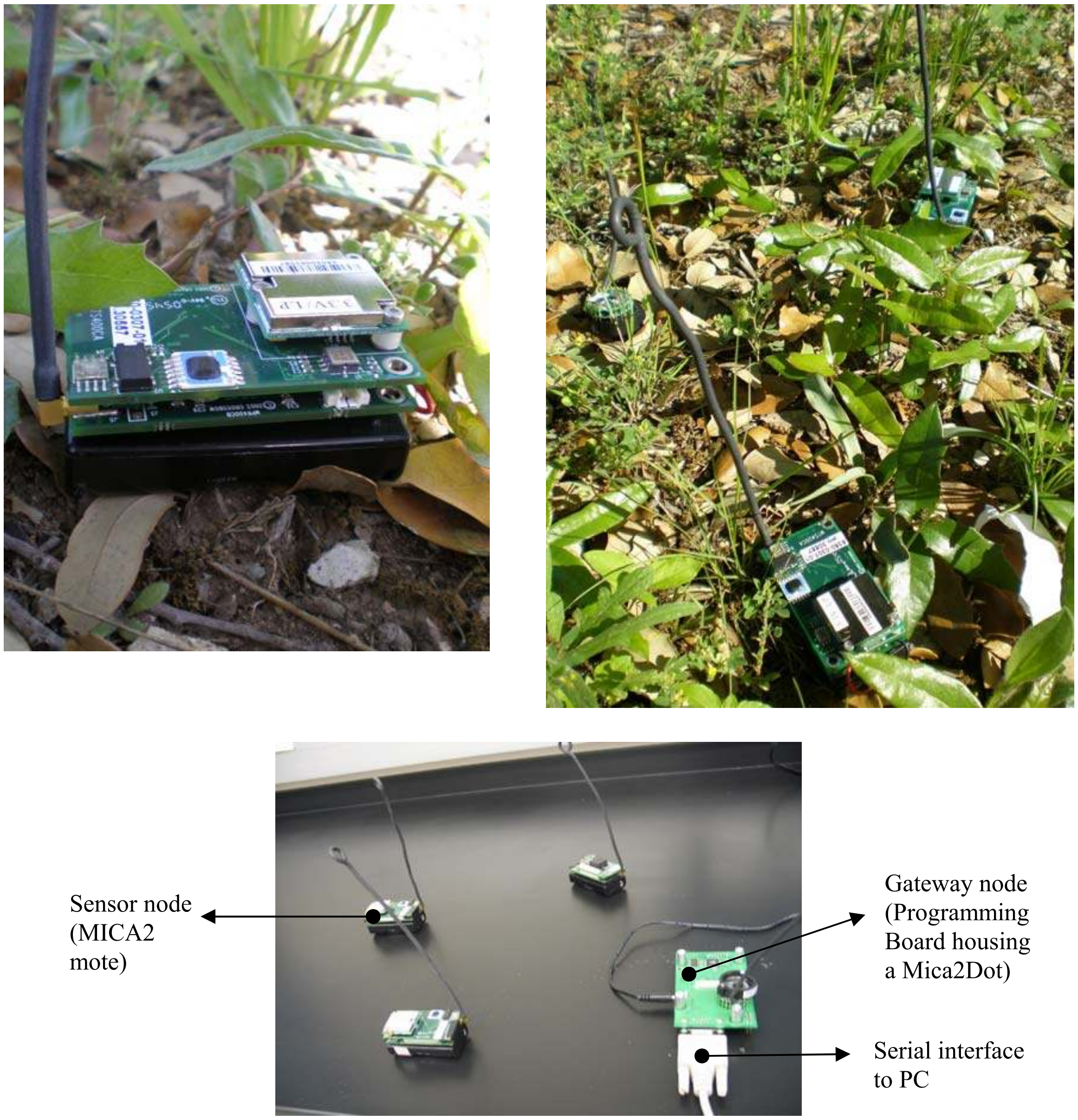


Figure 6.2 Sensor Nodes Deployed Outdoors and in Laboratory Settings

Each of these sensor nodes support an event-driven operating system called TinyOS. A simplified architecture for a sensor node is given in figure 6.3.

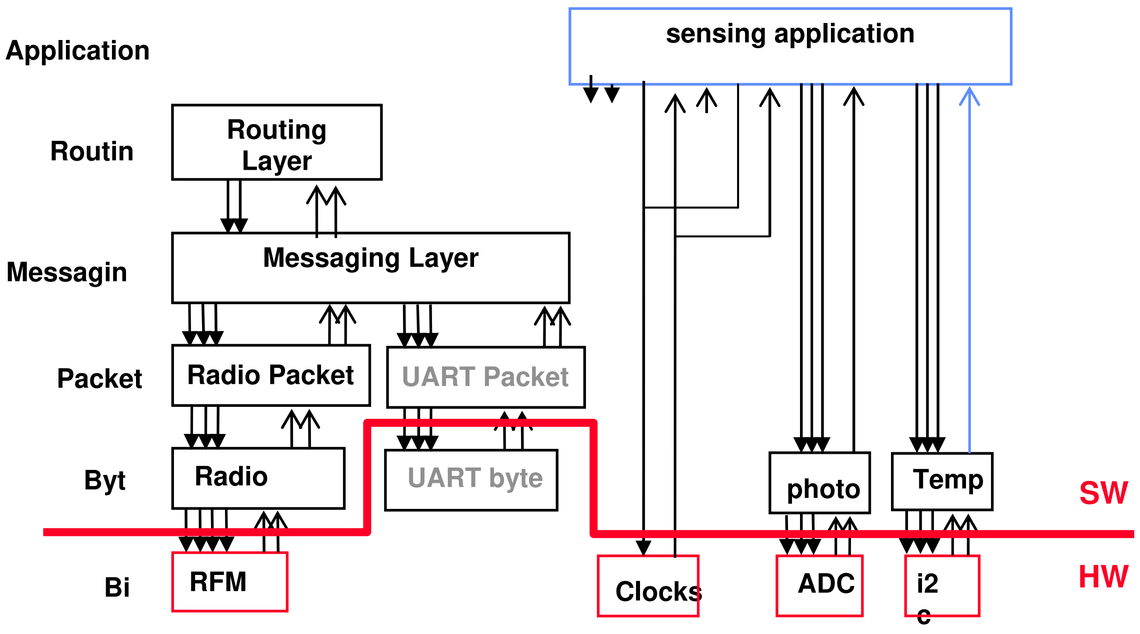
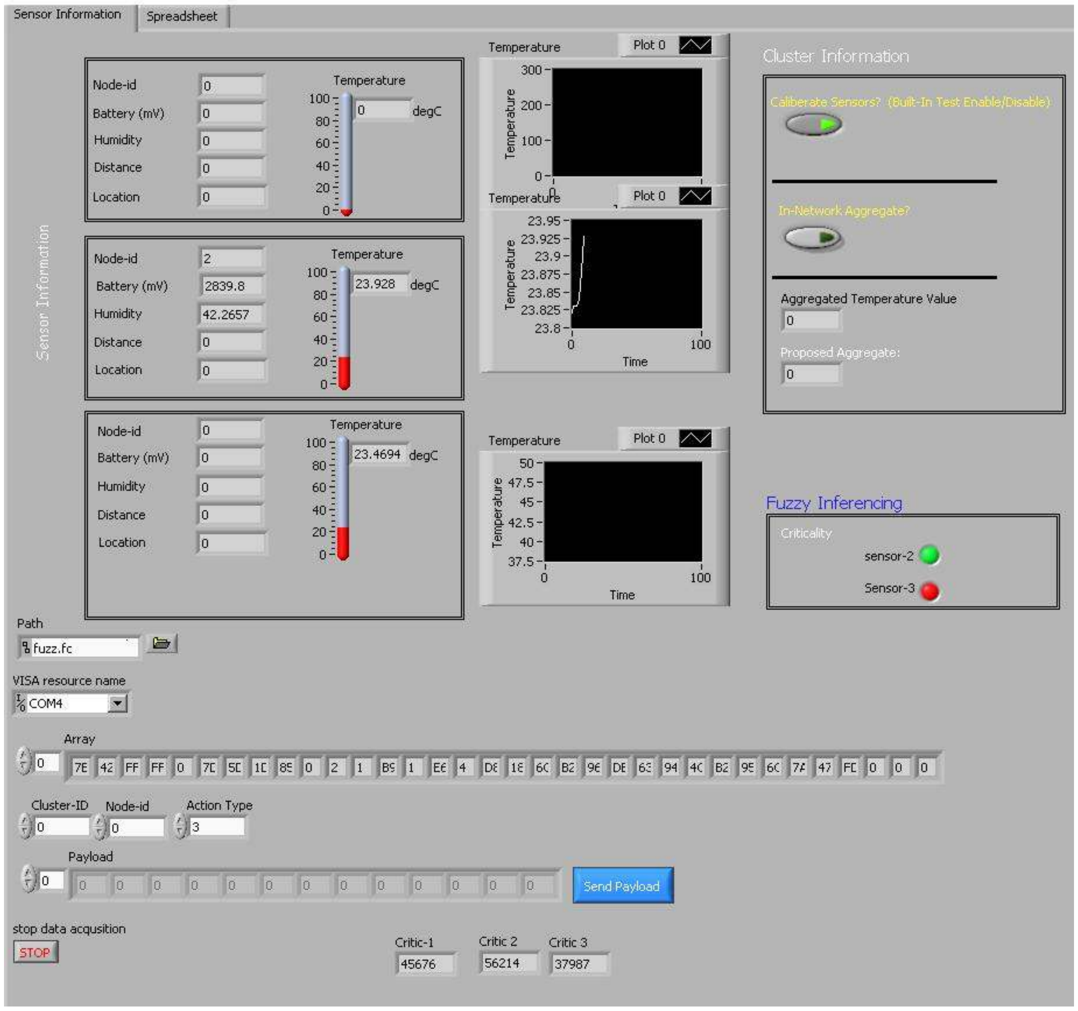


Figure 6.3 Simplified Architecture of a Mote

6.3 High Level Interface Design

The architecture proposed in figure 6.1 is divided horizontally – node-level implementation and base-station level implementation. This presents hybrid architecture, a combination of centralized and decentralized (or distributed) implementation. As stated before, extensive computation load should be handled by base-station which is assumed to have higher computation, power and storage capabilities compared to sensor nodes. This is our centralized implementation. Critical, faster, and less expensive computations should be handled at a node or cluster-head level, which is distributed in nature. At the base-station level, in order to visualize data from different sensors as well as to propagate decision from base-station to all the nodes it is necessary to have a user-level visualization tool. Such visualization tool should also enable the user to control or pass messages (to control) to the deployed sensors. This forms a ***sensing-decision-actuation*** loop. We developed the visualization tool in National Instruments' LabView® [78]. The choice for such development tool was the vast suit of tools LabView provides such as – interface to serial port, fuzzy control block, easy to use conversion tools (string to integer), etc.

Figure 6.4 gives a snapshot of the interface developed in LabView. A spread-sheet type interface (figure 6.5) is also developed for multiple sensor data acquisition.



Node-id	0
Battery (mV)	0
Humidity	0
Distance	0
Location	0
Temperature	0 degC

Node-id	2
Battery (mV)	2839.8
Humidity	42.2657
Distance	0
Location	0
Temperature	23.928 degC

Node-id	0
Battery (mV)	0
Humidity	0
Distance	0
Location	0
Temperature	23.4694 degC

Aggregated Temperature Value: 0

Proposed Aggregate: 0

Path: fuzz.fc

VISA resource name: COM4

Array: 0 | 7E 42 FF FF 0 7C 5C 1C 85 0 2 1 B9 1 E6 4 D8 18 6C B2 9E DE 63 94 4C B2 95 6C 7A 47 FC 0 0 0

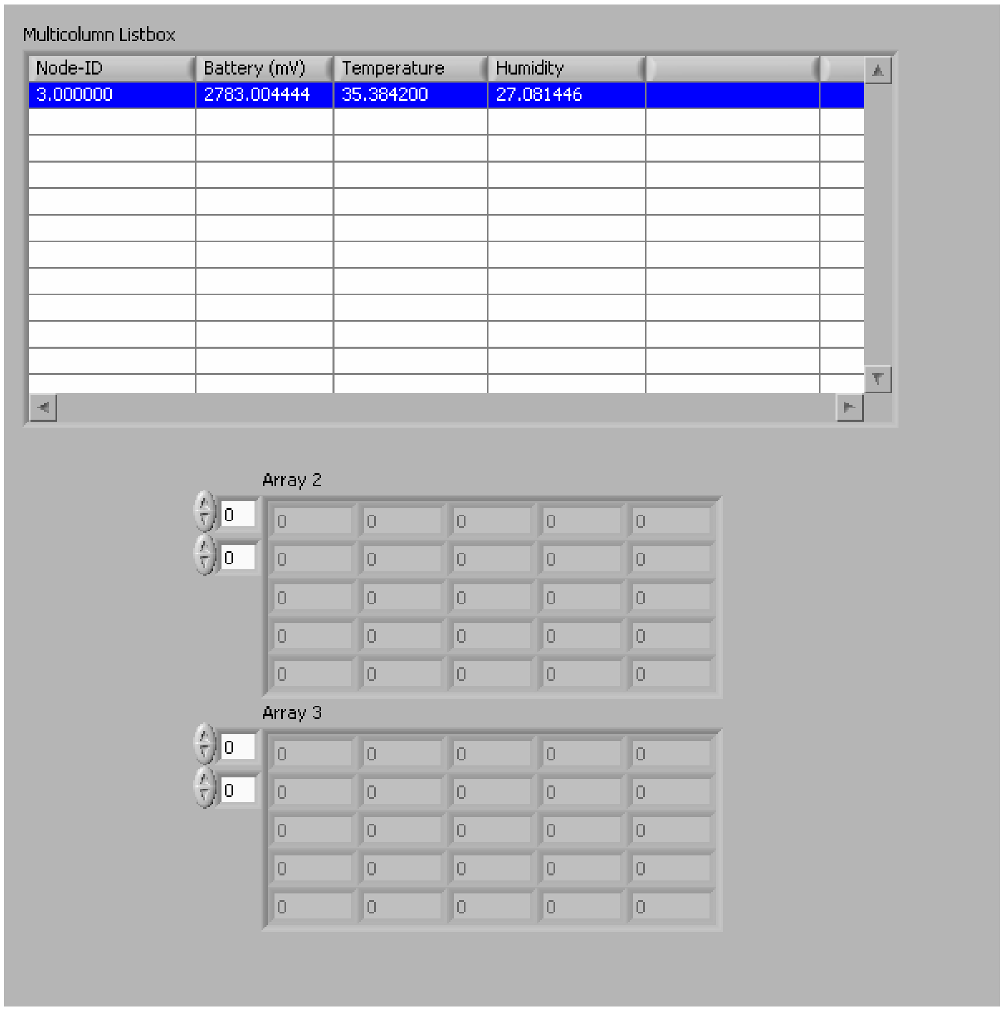
Cluster-ID: 0 Node-id: 0 Action Type: 3

Payload: 0 | 0 0 0 0 0 0 0 0 0 0 0 0 0

stop data acqusition

Critic-1: 45676 Critic 2: 56214 Critic 3: 37987

Figure 6.4 Interface for Data Acquisition and Decision Propagation



Multicolumn Listbox

Node-ID	Battery (mV)	Temperature	Humidity		
3.000000	2783.004444	35.384200	27.081446		

Array 2

Indices: 0, 0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Array 3

Indices: 0, 0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Figure 6.5 Spread-Sheet Type Interface for Data Acquisition

6.3.1 Packet Format (Protocol Design)

In order for the user to control the parameters on a remote sensor through the interface it is necessary to devise a protocol (and a message format). The message or packet format will be understood by the remotely deployed sensor and takes specific action based on the *action type* in the message. The message structure is as shown figure 6.6.

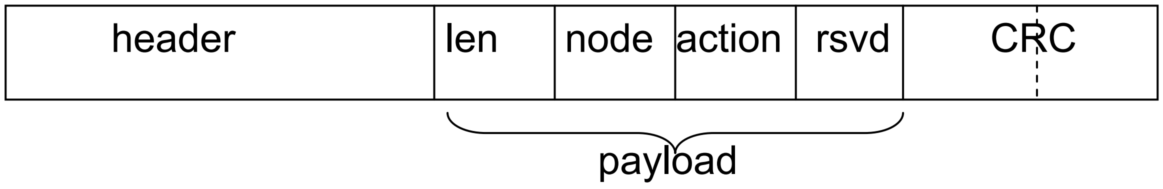


Figure 6.6 Message Structure for Over the Air Programming

The description of fields (8-bit) in the message structure is as follows:

len: Length of the payload
node: node-id to send the message
action: action to be performed on the node

action type:	01 – Aggregate data
	02 – Disable aggregation
	03 – Enable Built-in Test
	04 – Disable Built-in Test
	05 – High Sleep Time
	06 – Low Sleep Time
	07 – Reset Sleep Time
	08 – High Transmission Power (adjust potentiometer)
	09 – Reset Transmission Power

rsvd: reserved field for future use.
CRC: cyclic redundancy check (16-bit).

CRC provides better corruption detection mechanism than a regular checksum or parity bit. CRC used in TinyOS is CRC-CCITT [79]. Any packet that does not have a correct CRC for the payload sent will be dropped at the gateway node. Therefore, we developed the CRC in LabView interface and each time the payload changes, CRC is automatically calculated and appended at the end of the packet. A LabView implementation of CRC is given in figure 6.7

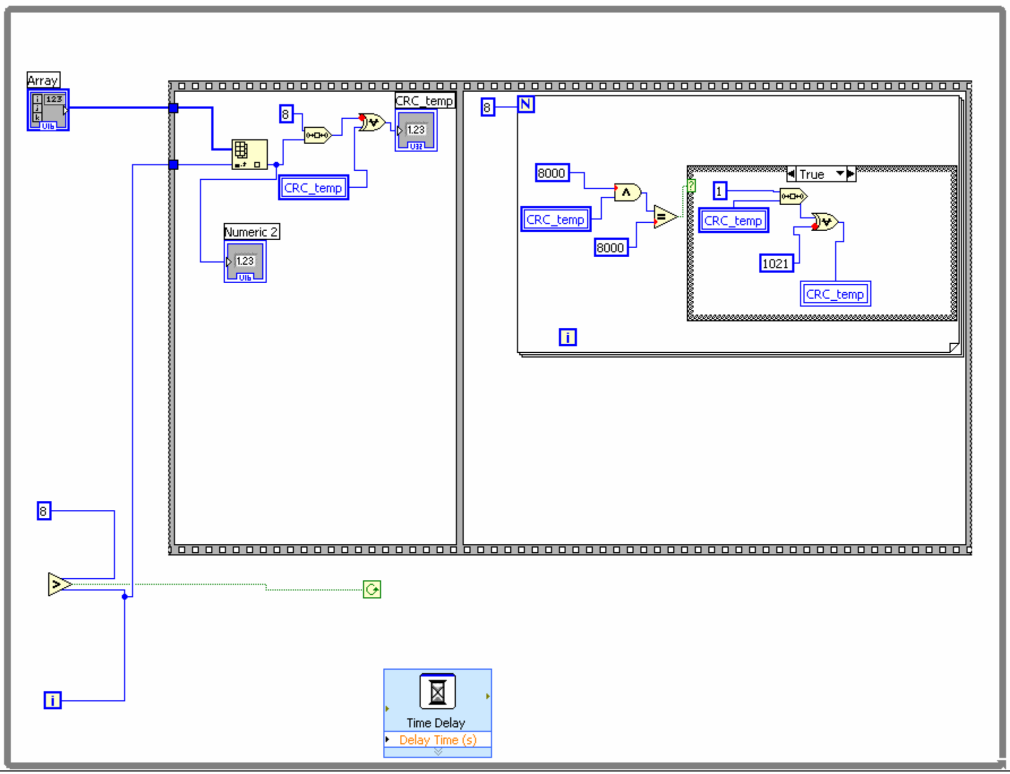


Figure 6.7 CRC for TinyOS Packets

The interface injects our custom message from the PC to the gateway node. The gateway node deciphers this packet and checks to see if the node-id in the message is set to its address. If not, it broadcasts the message over radio to all the sensor nodes. The sensor nodes check if the message belongs to it and take the action based on the action type in the message. In traditional Over-the-Air-Programming (OTAP), entire program is sent over radio to the node to be reprogrammed. The node is stopped, reloaded with new program and restarted. In our approach, we just send a specific action type for the sensor to react and change its behavior rather than complete reprogramming.

6.3.2 Engineering Conversion

The digital data from sensors (either from ADC of the processor or in-built ADC on sensor) is stored as 16-bit data in TinyOS. For example, Sensirion SHT11 temperature sensor on the sensor node (MTS420) that we used has an internal 14-bit ADC. The 16-bit data value for each sensor is the raw reading from the sensor which needs to be converted to engineering units. We implemented the conversion algorithms given in [80] in LabView. A snapshot for engineering conversion for battery voltage, temperature and humidity is given in figure 6.8.

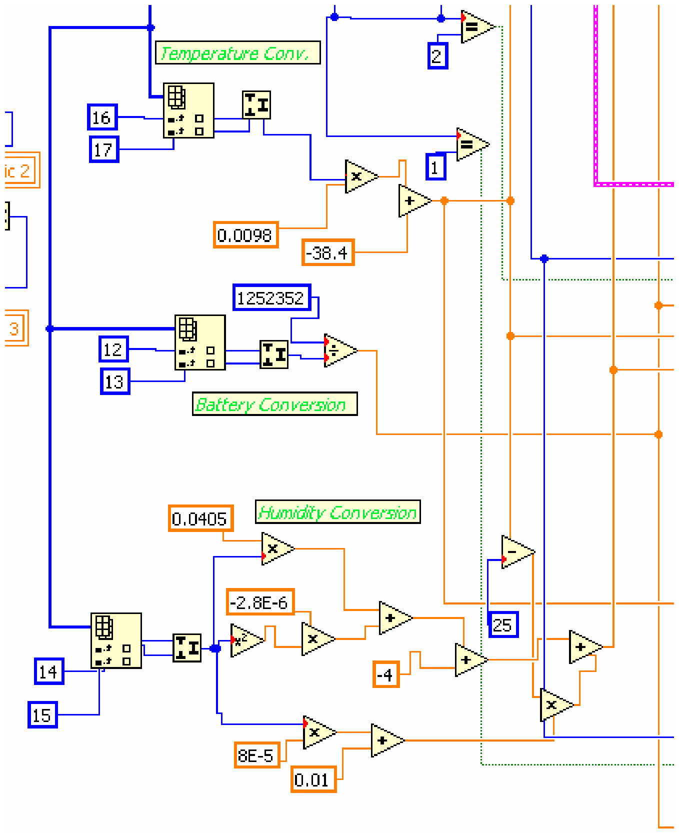


Figure 6.8 Engineering Conversion

We should also note that the packets that arrive from the sensor over radio are in *big-endian* format that needs to be converted to *little-endian* format.

6.4 Middleware Design and Development

For the remotely deployed sensor nodes to understand the commands from the PC, nodes need to decipher the packet (as shown in figure 6.6). So the action selection at the sensor nodes is based on the action type in the message. Therefore, there is a necessity for a middleware or service layer software component that is embedded in these sensor nodes that can decipher the message. Such middleware are often termed as Message Oriented Middleware (MOM) [81]. The term middleware is often used to loosely describe a software component that connects other software components or application. However, very specifically some definitions have stated that a middleware is a layer that lies between operating system and applications. In order to avoid controversy in definition, we call our software component that can understand our custom protocol as a message oriented service layer. Thus, the simplified architecture of the mote as shown in figure 6.3 is extended as shown in figure 6.9.

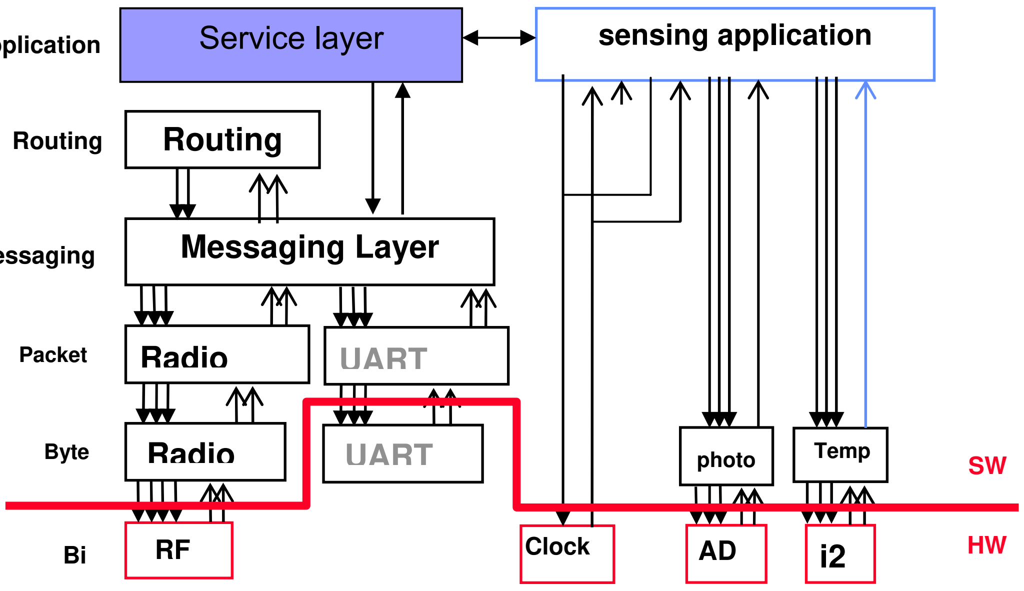


Figure 6.9 Modified Mote Architecture

Service layer developed is efficient in terms of processor, memory and power usage. Based on the general architecture given in figure 6.9, we have two main implementation of this service – one at the sensing level and other at the cluster-head (or aggregating node) level.

At the cluster-head level, the service layer performs fault-tolerance aggregation of data from different sensors. For experimental purposes, as stated before, we used three sensor nodes that are closely deployed to each other to report temperature. Based on the action type within the message packet, aggregation can be enable or disabled. This represents *decision-based aggregation* architecture.

At sensing level, service layer first checks to see if the packet from the gateway node belongs to it. This is done by comparing the node-id in the packet to TOS_LOCAL_ADDRESS, a constant defined while programming the sensor node. If the packet belongs to it, then the sensor node looks into the action field to take appropriate action. Also, based on the critic evaluation, the sleep time of the node is adaptively increased or decreased.

6.5 Experimental Results and Discussions

Part 1: Data Aggregation

As shown in figure 6.2, three temperature sensors deployed closed to each other in a laboratory setting is used to collect data and aggregate on a cluster head (gateway node in this case). We first perform aggregation (simple averaging) of the three temperature readings on the cluster head. We then introduce a uniform noise in one of the sensors (sensor-3) and compare the performance of our proposed algorithm (aggregation with spatial correlation), and averaging without correlation against ground truth.

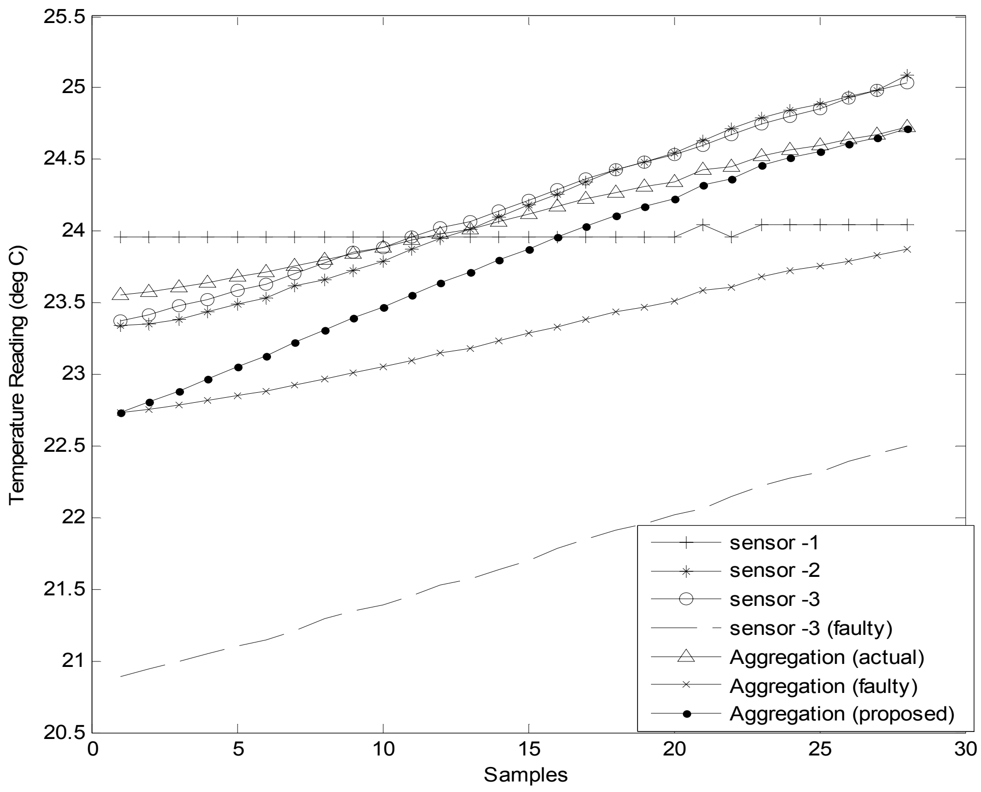


Figure 6.10 Comparison of Aggregation under Faulty Conditions

The temperature reading for three sensors are plotted at various time samples. Fault (noise) is introduced in one of sensors (sensor-3). The actual aggregation before the any fault is introduced represents our ground truth. As seen from figure 6.10, the aggregated temperature drastically reduces when the fault/noise is seen in any or all of the sensors. Our proposed weighted aggregation method compensates the faulty behavior by appropriately adjusting the weights. Therefore, the aggregated value steadily approaches the ground truth as seen from figure 6.10 and figure 6.11. As the aggregated value approaches ground truth (actual aggregated value), the error in the algorithm performance decreases and eventually becomes zero (see figure 6.12).

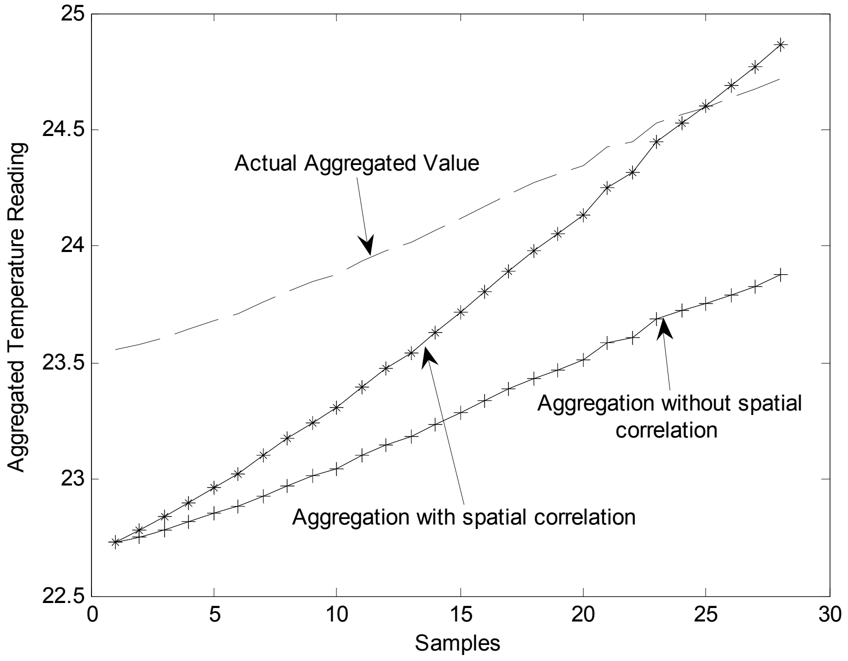


Figure 6.11 Aggregation with and without Spatial Correlation

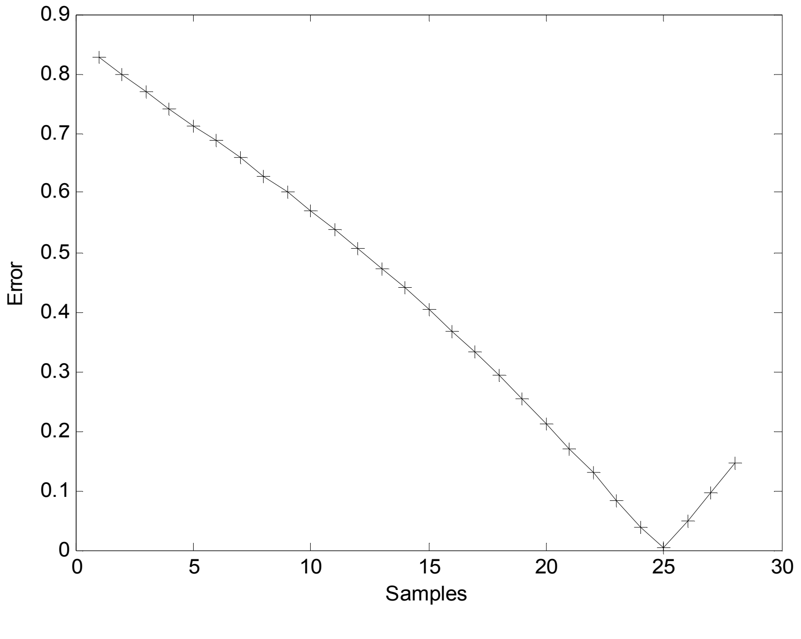


Figure 6.12 Error Between Ground Truth and Proposed Approach

The weight updates in figure 6.13 shows the decrease in weight for faulty sensor-3 thereby reducing its contribution in the aggregation process.

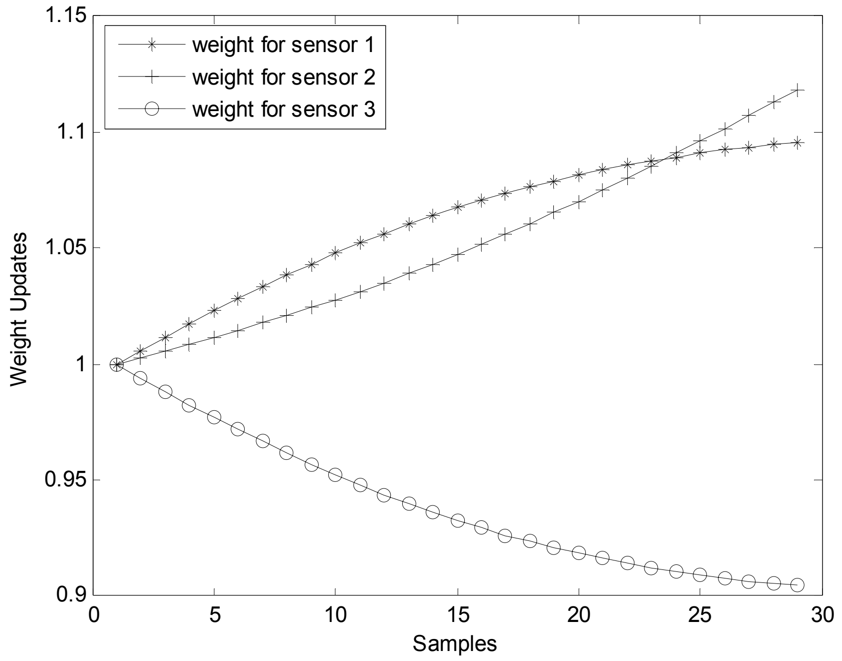
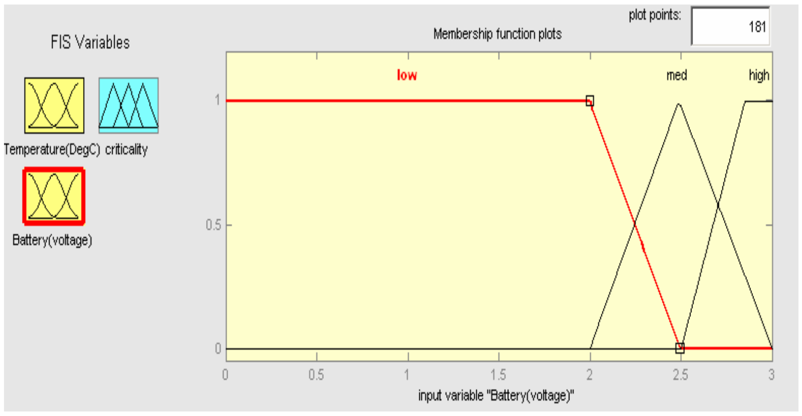


Figure 6.13 Weight Updates

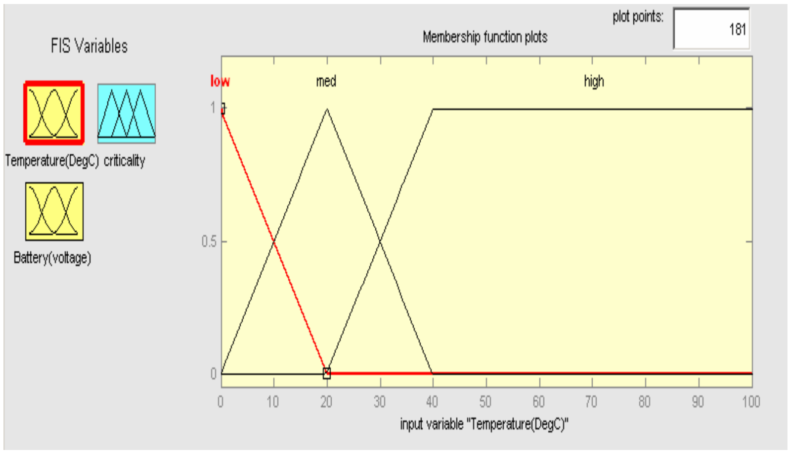
Part 2: Fuzzy Inference and Decision Making

We develop fuzzy rule-base to determine the data criticality depending on two input parameters of the sensor nodes - temperature and battery voltage. This criticality quantifies as to whether the data needs to be aggregated or not. The fuzzy inference engine evaluates simple common-sense rules such as "If activity (temperature) is high and battery power is low, then criticality is high". A high data criticality signifies a low chances to perform data aggregation so as to ensure "data freshness". The fuzzy controller is implemented on the PC with the high level interface.

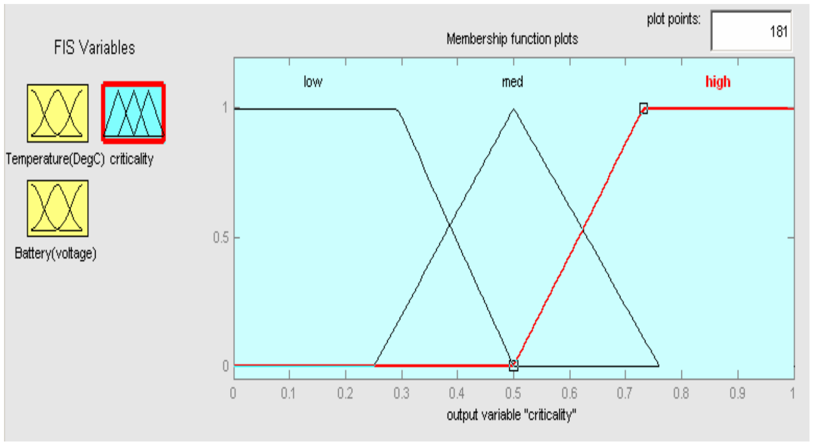
Based on the operating battery voltage of the sensors and the room temperature, fuzzy membership functions are designed as shown in figure 6.14.



(a) Battery Voltage as an Antecedent (input) to the Fuzzy System



(b) Room Temperature (in Deg C) as an Antecedent (input) to the Fuzzy System



(c) Criticality as a Consequent (output) of the Fuzzy System

Figure 6.14 Fuzzy Membership Functions

The control surface for the fuzzy rules developed is as shown in figure 6.15.

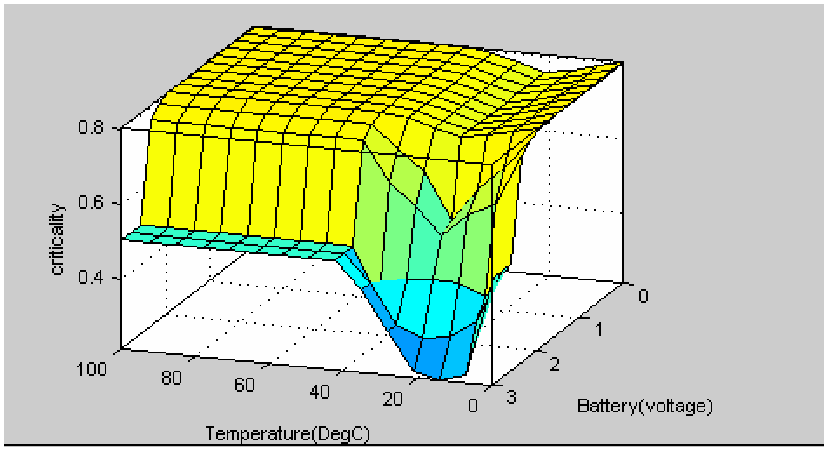


Figure 6.15 Control Surface

The output of the fuzzy system (in this case, criticality) can be now fed into automated decision making process such as MCDM using Choquet integral discussed in chapter 4.

We use criticality, distance from node to base-station, and sleep time (scheduling) as three inputs to the Choquet integral decision comparator. We set λ-fuzzy measure to -0.9 suggesting a *positive interaction*. A *positive interaction* or *positive synergy* (refer chapter 4) between two criteria *i* and *j* represents some degree of opposition between two criteria and the fuzzy measure then becomes *sub-additive*, i.e., $\mu(ij) < \mu(i) + \mu(j)$. $\mu(ij)$ is calculated using the formula: $\mu(ij)= \mu(i)+ \mu(j)+ \lambda\mu(i)\mu(j)$. Therefore, if $\lambda = 0.0$, then fuzzy measure is just additive, $\mu(ij) = \mu(i) + \mu(j)$, and the Choquet integral reduces to weighted average with fuzzy measures acting as weighting factors. Table 6.1 gives the fuzzy measure on each criterion and fuzzy measure on subset of criteria calculated using λ-fuzzy measure.

Table 6.1 Fuzzy Measures for Criteria

{ }	0
{distance}	0.7
{criticality}	0.8
{scheduling}	0.6
{d,c}	0.996
{d,s}	0.922
{s,c}	0.968
{d,s,c}	1

Given input values for distance, criticality and scheduling as 0.9, 0.6 and 0.5 respectively, we obtain Choquet integral value of 0.8096 (refer figure 6.16 for computation).

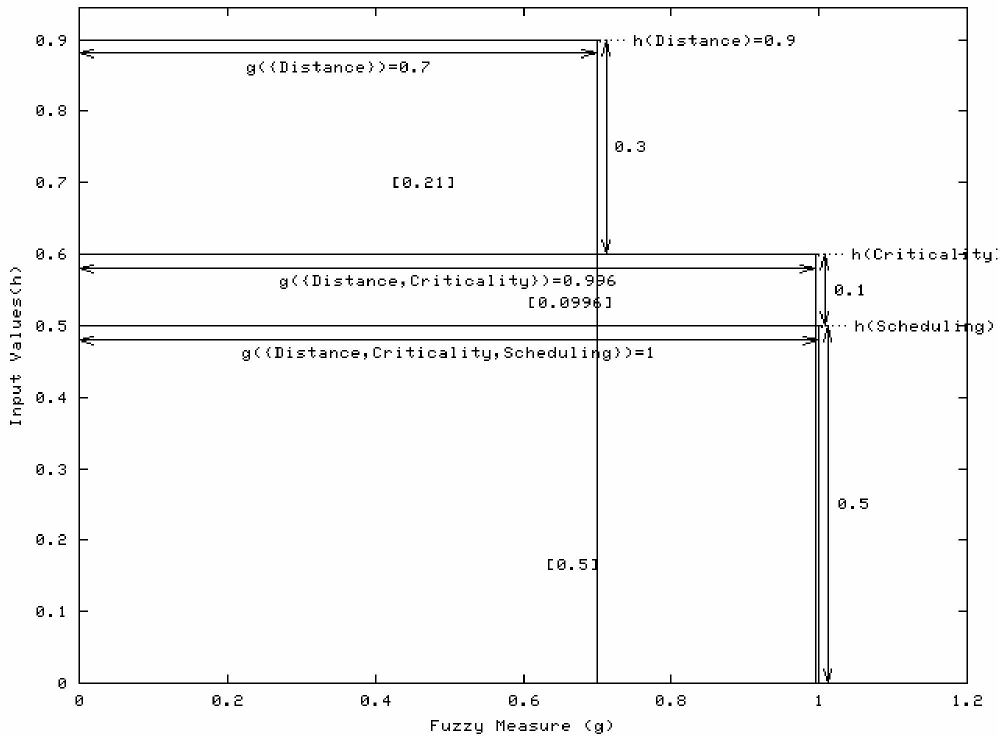


Figure 6.16 Computing Choquet Integral

Moreover, such decision making process based on the state of the sensor node (battery) and the environment (temperature) can be used in our reinforcement learning, so that the sensor node learns to adaptively manage the energy consumption (as discussed in chapter 5). We implement the critic function developed in chapter 5 on the sensor node and vary outside temperature (by blowing hot air) to see the varying sleep time of the sensor node. We compare the performance of sensor node with and without critic. With critic loaded on the sensor node, we attained a reduction in number of packets transmitted by almost 10 times with a very few misses in registering the events. The number of transmitted packets with and without critic running on sensor node is as shown in figure 6.17.

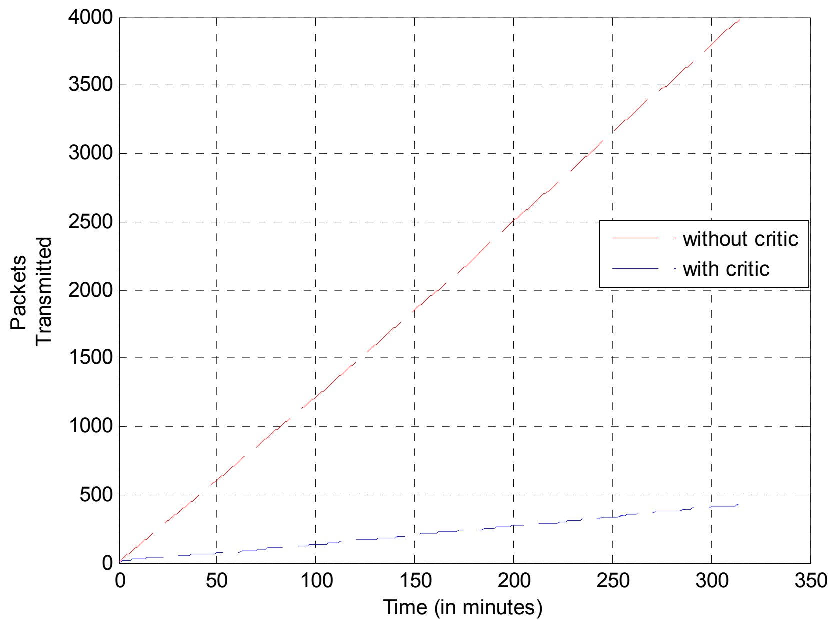


Figure 6.17 Packet Transmission Comparison

The events registered (i.e., temperature reading captured) with and without critic is given in 6.18. As seen from the figure, there are only few temperature differences not captured by using adaptive sleep time. Major missed events are circled in red. Figure

6.19 shows the performance of our critic function. Whenever there is a change in temperature registered, the sleep time is automatically decreased so as to capture the change with a finer resolution. In a general case, this can be thought of as sensor adaptively waking up based on the environmental changes in order to localize the events.

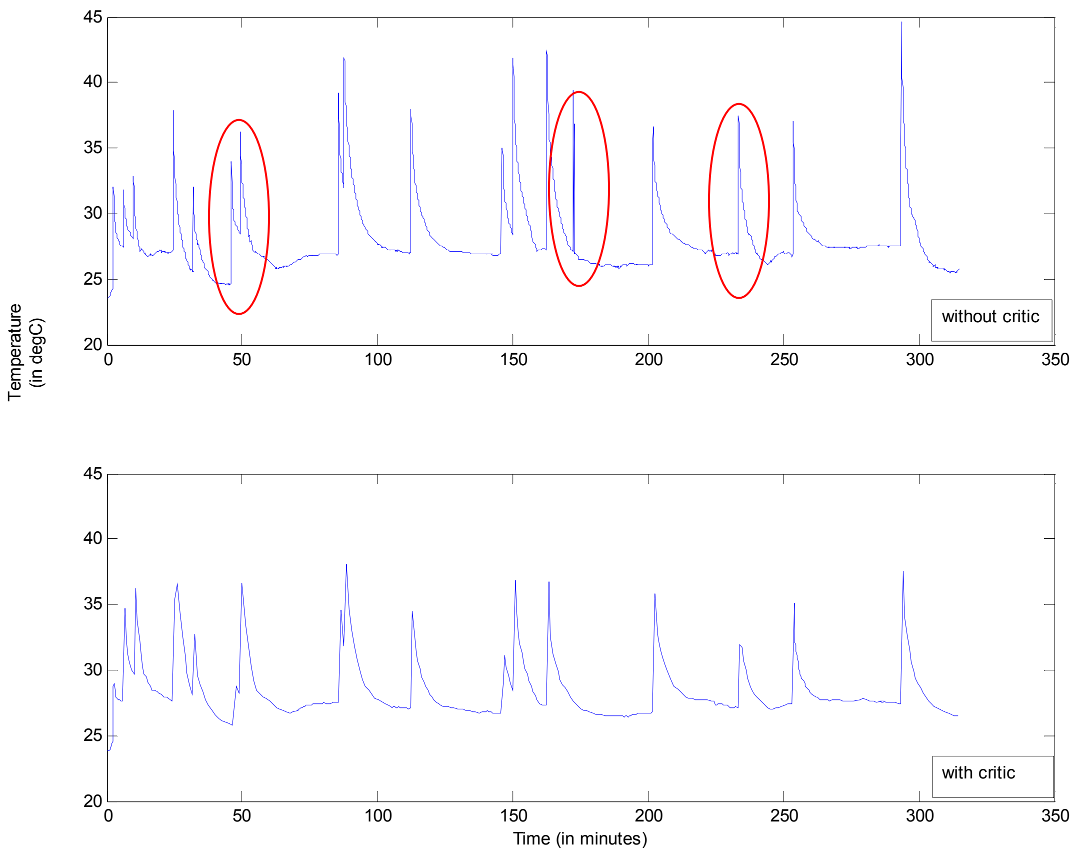


Figure 6.18 Temperature Changes Captured

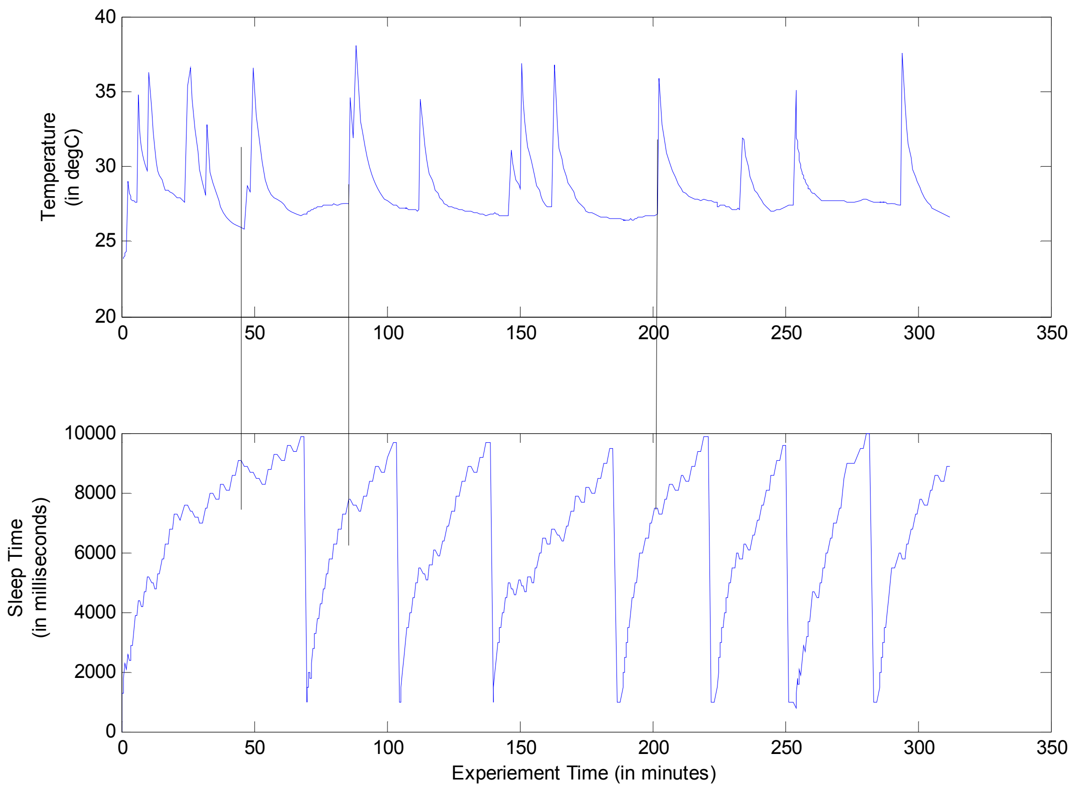


Figure 6.19 Variations in Sleep Time

CHAPTER 7

CONCLUDING REMARKS

7.1 Summary

The techniques and concepts provided in this thesis are generic in nature and are applicable to any multi-sensor application. There are several research problems that still exist in sensor networks. We have made a decent attempt to address only a handful of problems by providing theory, design and implementation of the solution. The vast majority of ongoing research in sensor networks is engaged in network routing, power management, protocol development, and/or application-specific. Alternatively, this research is focused on a sensor abstraction layer and utilizes the underlying attributes that are present in sensor networks (such as high node density, ad-hoc behavior, etc.) in designing our solution. We will summarize the important concepts or techniques provided in this thesis.

In chapter 2, we developed a weighted aggregation method that when implemented hierarchically reduced the number of network packets transmitted by an order of the number of nodes transmitting the packets. Exploiting the spatial correlation that is often seen in sensor networks, weight adaptation mechanism helped to address the issue of soft faults (in-range and slow-drift failure). Soft faults are often seen when the sensors work within the given range of operation. Such technique when augmented with Built-in Test (BIT) provides robust mechanism to process and acquire large amounts of fault-tolerant data. BIT methods help to determine hard-faults (when sensor reading is outside the operating range). BIT together with spatial correlated weighted adaptation method help to determine hard as well as soft faults. To demonstrate the effectiveness of the aggregation and the BIT schemes proposed, we developed a middleware at the cluster-head node that implemented the timeout mechanism and aggregation of temperature sensor data.

Chapter 3 and 4 summarized important concepts relating to monitoring and decision making in sensor networks. The basic idea underlying monitoring and probing the given network is to improve upon the performance of the system (in this case given network of sensing devices). Our approach was to use human-like reasoning to deal with complex multi-parameter network to characterize the behavior. By exploiting the nature of fuzzy logic controller which efficiently handles uncertainty and nonlinearity in the system, we developed simple rule-base to monitor and thereby update the operating parameters (such as sleep time, power level, etc.) of the network. By doing so, we improved the performance (either in terms of lifetime or node integrity) of the deployed network. Choquet integral introduced in chapter 4 provides a mathematical basis for decision making with multiple interacting criteria. Such decision making is often helpful in planning process to determine the sensor function or usage according to changing situations. The decision making process on what tasks the sensor needs to accomplish depending on the mission plan and situation generally depends on the various criteria involved. Once the behavior pattern for a given application is identified, the state of the sensor network and its performance can be used as feedback for creating training set for learning algorithms such as neural networks.

We also presented a novel approach in designing a reinforcement learning scheme that will guide the sensor to adaptively sleep so as to reduce the network packets as well as conserve on-board battery power. The node sleeps as much as possible but at the same time should handle the stimuli (capture events) from the environment. This complex contradicting requirement is embedded in a reward function that is developed and implemented as chapter 5. The decision making rules (either through fuzzy rule-base or Choquet integral) can be adaptively changed by such reinforcement learning algorithms.

7.2 Suggested Follow-on Work

Solutions were proposed in this thesis with the main aim for practical implementation on available sensor platforms. Several extensions to the proposed work in this thesis can be thought of. Specifically, we have identified the following future work:

1. In the area of data aggregation, a robust mechanism to fuse the data from heterogeneous sensors to a meaningful decision information.
2. Built-in Test methods provide node-level (micro) calibration. A more comprehensive approach is needed to utilize this micro calibration and extend it to network level (macro) calibration. Although, there is some work done on macro-calibration [82], it is generally hard to calibrate when there are functionally heterogeneous network of sensing devices and is an interesting research topic to pursue.
3. Very few directed research has been done in the area of machine learning in sensor networks. Our proposed learning methodology is at node-level. At a network level, an interested learning topic would be to analyze the adaptive behavior of each node by looking into the behavior of neighboring sensor nodes.

4. Resource optimization can be another approach as a continuation of this thesis in the area of sensor network management. Optimization techniques such as Particle Swarm Optimization (PSO) [83] are simple and implementable on sensor platforms.
5. The user interface can be extended to handle message from sensors from different manufacturers, thus providing a unified platform for analyzing pure network of heterogeneous sensing devices.

REFERENCES

[1] Chong, C., Kumar, S., “Sensor Networks: Evolution, Opportunities, and Challenges”, Proc. of the IEEE, Vol. 91, No. 8, 2003

[2] Estrin, D., Govindan, R., Heidemann, J., Kumar, S. , “Next Century Challenges: Scalable Coordination in Sensor Networks”, Proc. of the ACM/IEEE Conference on Mobile Computing and Networking, pp. 263-270, 1999

[3] Raghunathan, V., Schurgers, C., Park, S., Srivastava, M., "Energy Efficient Design of Wireless Sensor Nodes", Wireless Sensor Networks, pp. 51-69, Kluwer Academic Publishers, 2004

[4] Heinzelman,W., Chandrakasan,A., and Balakrishnan,H., "Energy-Efficient Communication Protocol for Wireless Microsensor Networks", Proc. of the 33rd Hawaii International Conference on System Sciences (HICSS '00), 2000

[5] Bandyopadhyay, S., Colye, E., “An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks”, Proc. of IEEE Infocom, pp. 1713-1723, 2003

[6] Chan, Perrig, “ACE: An Emergent Algorithm for Highly Uniform Cluster Formation”, Proc. of the First European Workshop on Sensor Networks (EWSN), Germany, 2004

[7] Madni, Asad M., Costlow, Lynn E., “Common Design Techniques for BEI GyroChip® Quartz Rate Sensors for both Automotive and Aerospace/Defense Markets”, IEEE Transactions on Sensors Journal, Vol 3 No. 5, pp. 569-578, October 2003

[8] Chen, L., Dey, S., Sanchez, P., Sekar, K., Chen, Y., “Embedded Hardware and Software Self-Testing Methodologies for Processor Cores”, 37th Design Automation Conference, pp. 625 - 630, 2000

[9] F. Koushanfar, M. Potkonjak, A. Vincentelli, “Fault Tolerance Techniques for Wireless Ad hoc Sensor Networks”, IEEE Sensors Journal, Vol 2., pp. 1491- 1496, 2002

[10] Elnahrawy, E., Nath, B., “Cleaning and Querying Noisy Sensors”, Proc. of the Workshop on wireless sensor networks and applications, pp. 78 – 87, 2003

[11] Hereford, J., “Fault-Tolerant Sensor Systems Using Evolvable Hardware”, IEEE Transactions on Instrumentation and Measurement, Vol. 55, No. 3, pp. 846-853, June 2006

[12] Culler,D., Estrin,D., Srivastava,M., “Overview of Sensor Networks”, IEEE Magazine – Computers, Vol 37, No. 8, pp. 41-49, 2004

[13] R. R. Brooks, S. S. Iyengar, “Multi-Sensor Fusion: Fundamentals and Applications With Software”, Prentice Hall PTR, 1997

[14] Cohen, O., Edan, Y. , “Adaptive Fuzzy Logic Algorithms for Sensor Fusion Mapping”, Proc. of the IEEE International Conference on Systems, Man and Cybernetics, pp. 2326- 2331, 2004

[15] Kahler, O., Denzler, J., Triesch, J., “Hierarchical Sensor Data Fusion by Probabilistic Cue Integration for Robust 3D Object Tracking”, Proc. of the 6th IEEE Southwest Symposium on Image Analysis and Interpretation, pp. 216-220, 2004

[16] Krishnamachari, B., and S. S. Iyengar, “Distributed Bayesian Algorithms for Fault-tolerant Event Region Detection in Wireless Sensor Networks”, IEEE Transaction on Computers, Vol. 53, No. 3, pp. 241-250, 2004

[17] Hall, D., Llinas, J., "Ab Introduction to Multisensor Data Fusion", Proceeding of the IEEE, Vol. 85, No. 1, 1997

[18] Yager, R.R., “On Ordered Weighted Averaging Aggregation Operators in Multi-Criteria Decision Making”, IEEE Trans. On Systems, Man and Cybernetics, Vol. 18, No. 1, pp. 183-190, 1988

[19] Detyniecki, M., "Mathematical Aggregation Operators and Their Application to Video Querying", Doctoral thesis - research report 2001-002, Laboratoire d'Informatique de Paris 6, 2000

[20] Torra, V., “On Integration of Numerical Information: From Arithmetic Mean to Fuzzy Integrals”, Information fusion in data mining, PhysiinVerlag, 2001

[21] Zhao, J., Govindan, R., Estrin, D., “Residual Energy Scan for Monitoring Sensor Networks”, Proc. of the IEEE Wireless Communication and Networking Conference (WCNC), pp 356-362, 2002

[22] Intanagonwiwat, C., Govindan, R., and Estrin, D., “Directed Diffusion for Sensor Networks", IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 2-16, 2003

[23] Heinzelman, W., Kulik, J., Balakrishnan, H., “Adaptive Protocols for Information Dissemination in Wireless Sensor Networks”, Proc. of the 5th Annual ACM/IEEE Conference on Mobile Computing and Networking, pp. 174-185, 1999

[24] Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W., “TAG: A Tiny Aggregation Service for Ad-hoc Sensor Networks”, Proc. of the Symposium on Operating Systems Design and Implementation, (OSDI), 2002

[25] Elson, J., Girod, L., Estrin, D., “Fine-grained Network Time Synchronization using Reference Broadcasts”, Proc. of the 5th Symposium on Operating Systems design and Implmentation (OSDI), 2002

[26] Solis, I., Obraczka, K., “The Impact of Timing in Data Aggregation for Sensor Networks”, Proc. of the IEEE Conference on Communication, pp. 3640-3645, 2004

[27] Boulis, A., Ganeriwal, S., Srivastava, M., “Aggregation in Sensor Networks: An Energy-Accuracy Trade-off”, Proc. of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003

[28] Nelson, M., Gailly, J-L., “The Data Compression Book”, MIS Press, NY, 1995

[29] Knuth, D., “Dynamic Huffman Coding”, Journal of Algorithms, Vol. 6, pp. 163-180, 1985

[30] Witten, I., Neal, R., Cleary, J., “Arithmetic Coding for Data Compression”, Communications of the ACM, Vol. 30, pp. 520-540, 1987

[31] Hauck, E., “Data Compression using Run Length Encoding and Statistical Encoding”, U.S. Patent No. 4626829, 1985

[32] Ziv, J., Lempel, A., “A Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343, 1977

[33] Ning Xu, “Implementation of Data Compression and FFT on TinyOS”, Embedded Networks laboratory, Computer Science Dept. USC, Los Angeles, http://enl.usc.edu/ningxu/papers/lzfft.pdf

[34] Elson, J., "Sensor Network Software Challenges", The 4th International Conference on Information Processing in Sensor Networks, Tutorial-B, Los Angeles, CA, 2005

[35] Klir, G., Yuan, B., “Fuzzy Sets and Fuzzy Logic: Theory and Applications”, Chapter 15: Fuzzy Decision Making, Prentice Hall NJ, 1995

[36] Jamshidi, M., “Large Scale Systems: Modelling and Control”, North Holland, New York, 1983

[37] Zilouchian, A., Jamshidi, M., “Intelligent Control Systems using Soft Computing Methodologies”, CRC Press, 2001

[38] Madni, Asad M., Wan, L. A., Hansen, R. K., and Vuong,J. B., “Adaptive Fuzzy Logic Based Control System for Rifle Stabilization”, Proc. of the World Automation Congress, Paper No. ISSCI 96, 1998

[39] Wan, C-Y., Eisenman, S., Campbell, A., “CODA: Congestion Detection and Aviodance in Sensor networks”, Proc. ACM Conference on Embedded Network Sensor Systems (SenSys), pp. 266 - 279, 2003

[40] Sridhar, P., Madni, Asad M., Jamshidi, M., “Intelligent Monitoring of Sensor Networks using Fuzzy Logic Based Control”, Proc. of the IEEE International Conference on Systems, Man and Cybernetics, 2006

[41] Crossbow Inc., www.xbow.com

[42] Dixon, W.E., Zergeroglu, E., Fang, Y., Dawson, D.M. , “Object Tracking by a Robot Manipulator: A Robust Cooperative Visual Servoing Approach”, IEEE Conference on Robotics and Automation, 2002

[43] Luo, R.C., “Target Tracking using Hierarchical Grey-fuzzy Motion Decision-Making Methods”, IEEE transactions on SMC – Part A: Systems and Humans, vol. 31, no.3, 2001

[44] Xiao-Rong Li and Vesselin P. Jilkov, “A Survey of Maneuvering Target Tracking III: Measurement Models”, Proc. of SPIE- Signal and Data Processing of Small Targets, 2001

[45] Tan, J., Kyriakopoulos, N., “Implementation of a Tracking Kalman Filter on a Digital Signal Processor”, IEEE Transaction on Industrial Electronics, Vol. 35, 1988

[46] Tao, H,Sawhney, H.S.,Kumar, R., “Object Tracking with Bayesian Estimation of Dynamic Layer Representations”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002

[47] Chang, C., Ansari, R. Khokhar, A., “Multiple Object Tracking with Kernel Particle Filter”, Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, 2005

[48] Kung, H.T., Vlah, D., "Efficient Location Tracking using Sensor Networks", IEEE Wireless Communications and Networking, Vol. 3, 2003

[49] Yang, H., Sikdar, B., "A Protocol for Tracking Mobile Targets using Sensor Networks", Proc. of the IEEE International Workshop on Sensor Network Protocols and Applications, 11 May 2003

[50] Viswanathan, R., Varshney, P.K., "Distributed Detection with Multiple Sensors: Part I – Fundamentals", Proc. of IEEE, Vol. 85, No. 1, 1997

[51] Liu, P.X., Meng, M., Hu, C., "On-Line Data-Driven Fuzzy Clustering with Applications to Real-Time Robotic Tracking", Proc. of the IEEE Conference on Robotics and Automation, 2005

[52] Pal, N., Pal, K., Keller, J., Bezdek, J.,"A Possibilistic Fuzzy c-Means Clustering Algorithm", IEEE Transactions on Fuzzy Systems, Vol. 13, No. 4., pp. 517-530, Aug 2005

[53] Younis, O., Fahmy, S.,"HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks", IEEE Transactions on Mobile Computing, pp. 366-379, Dec 2004

[54] Ziegler, B., Sarjoughian, H., "Creating distributed simulation using DEVS M&S environments", Proc. of the 2000 winter simulation conference, 2000

[55] Thrishantha Nanayakkara, Keigo Watanabe, Kazuo Kiguchi and Kiyotaka Izumi, "Fuzzy Self-Adaptive RBF Neural Network Based Control of a Seven-Link Industrial Robot Manipulator," in Advanced Robotics, Vol. 15, No. 1, pp. 17-43, 2001

[56] Decision Making Models - web.mit.edu/ist/about/decisionmaking/models.pdf

[57] Belton,V., Steward, T.J., "Multiple Criteria Decision Analysis-An Integrated Approach", Kluwer Academic Publisher*s*, 2002

[58] Marichal., J-L., "An Axiomatic Approach of the Discrete Choquet Integral as a Tool to Aggregate Interacting Criteria", *IEEE Transactions on Fuzzy Systems,* Vol 8, No. 6, pp. 800-807, Dec 2000

[59] Grabisch, M., "The Application of Fuzzy Integral in Multicriteria Decision Making", European journal of operational research, Vol. 89, pp. 445-456, 1995

[60] Denguir-Rekik,A., Mauris,G., Montmain,J.,"Propagation of Uncertainty by the Possibility Theory in Choquet Integral-Based Decision Making: Application to an E-

Commerce Website Choice Support", *IEEE Transactions on Instrumentation and Measurement*, VOL. 55, NO. 3, June 2006

[61] Wang Z., Klir, G.J., Wang, W., "Determining Fuzzy Measures by Choquet Integral", Proc. of the 3rd International Symposium on Uncertainty Modeling and Analysis, pp. 724, 1995

[62] Grabisch, M., "A Graphical Interpretation of the Choquet Integral", IEEE Transactions on Fuzzy Systems, Vol 8, No. 5, pp 627-631, 2000

[63] Klir, G., Yuan, B., "Fuzzy Sets and Fuzzy Logic: Theory and Applications", Chapter 15: Fuzzy Decision Making, Prentice Hall NJ, 1995

[64] Huédé, F., Grabisch, M., Labreuche, C., Savéant, P., "Integration and Propagation of a Multi-criteria Decision Making Model in Constraint Programming", *Journal of Heuristics*, Vol 12 ,Issue 4-5, pp. 329 – 346, 2006

[65] Azarnoush, H., Horan, B., Sridhar P., Madni, A M., Jamshidi, M., "Towards Optimization of a Real-World Robotic-Sensor System of Systems", World Automation Congress, Budapest, Hungary, 2006

[66] Margi, C., Obraczka, K., Manduchi, R., "Energy Consumption Trade-offs in Sensor Networks", Poster - Internetworking Research Group, UC Santa Cruz, http://inrg.cse.ucsc.edu/posters/citris-energy.pdf

[67] Predd, J., Kulkarni, S., Poor, V., "Distributed Learning in Wireless Sensor Networks", IEEE Signal Processing Magazine, pp. 56-69, July 2006

[68] Wang, P., Wang, T., "Adaptive Routing for Sensor Networks using Reinforcement Learning", Proc. of the Sixth IEEE conference on Computer and Information Technology, pp 219, 2006

[69] Martyna, J., "Fuzzy Reinforcement Learning for Routing in Wireless Sensor Networks", Computational Intelligence, Theory and Applications, Part 23, pp. 637-645, 2006

[70] Simic, S., "A Learning-theory Approach to Sensor Networks", Proc. of the IEEE Pervasive Computing, Vol. 2, pp. 44-49, 2003

[71] Nguyen, X., Wainwright, M., Jordan, M., "Decentralized Detection and Classification using Kernel Methods", Proc. of the International Conference on Machine Learning, 2004

[72] Sutton, R., Barto, A., “Reinforcement Learning: An Introduction”, MIT Press, 1998

[73] Haykin, S., “Neural Networks: A Comprehensive Foundation”, Macmillan, 1994

[74] Alpaydin, E., “Introduction to Machine Learning”, chapter 1, MIT Press, 2004

[75] Mustapha, S., Lachiver, G., “A Modified Actor-Critic Reinforcement Learning Algorithm”, Proc. of the Canadian Conference on Electrical and Computer Engineering, Vol. 2, pp. 605-609, 2000

[76] Barto, A., Sutton, R., Anderson, C., “Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems” , IEEE Transactions on Systems, Man, and Cybernetics, Vol. 13, No. 5, pp. 834-846, 1983

[77] Greenstein, B., Levis, P., “TEP 113: Serial Communication”, in TinyOS Extension Proposals, http://www.tinyos.net/tinyos-2.x/doc/txt/tep113.txt

[78] National Instruments, www.ni.com

[79] Stallings, W., “Wireless Communications and Networking”, chapter 8, Pearson Education, 2002

[80] Sensirion SHT11 temperature/humidity sensor data sheet, www.sensirion.com

[81] Rao, B.R., “Making the Most of Middleware” Proc. of the Data Communications International 24,pp. 89-96, 1995

[82] Whitehouse, K., Culler, D., “Macro-calibration in Sensor/actuator Networks”, Mobile Networks and Applications, Kluwer Academic Publishers, pp. 463-472, 2003

[83] Eberhart, R. C., Kennedy, J., “A New Optimizer using Particle Swarm Theory”, Proc. of the Sixth International Symposium on Micromachine and Human Science, pp. 39-43, 1995

APPENDIX – A

Source Code

```
/* Program that runs on each sensor node that is deployed in the
environment
*/

module TestSensor{
  provides interface StdControl;
  uses {

      //communication
      interface StdControl as CommControl;
      interface SendMsg as Send;
      interface ReceiveMsg as Receive;

// Battery
    interface ADC as ADCBATT;
    interface StdControl as BattControl;

//Accels
    interface StdControl as AccelControl;
    interface I2CSwitchCmds as AccelCmd;
    interface ADC as AccelX;
    interface ADC as AccelY;

//Intersema
    interface SplitControl as PressureControl;
    //interface StdControl as PressureControl;
    interface ADC as IntersemaTemp;
    interface ADC as IntersemaPressure;
    interface Calibration as IntersemaCal;

//Sensirion
    interface SplitControl as TempHumControl;
    interface ADC as Humidity;
    interface ADC as Temperature;
    interface ADCError as HumidityError;
    interface ADCError as TemperatureError;
//Taos
    interface SplitControl as TaosControl;
    interface ADC as TaosCh0;
    interface ADC as TaosCh1;

    interface Timer;
    interface Leds;


  }

}


implementation
{
```

```
#define TIMER_PERIOD 2000           // timer period in msec
#define TIMER_INC_PERIOD 6000
#define TIMER_DEC_PERIOD 500


#define SAMPLE_SIZE 5
#define TEMP_SAMPLE 2

  char count;

  uint16_t calibration[4];           //intersema calibration words
  norace uint8_t  state;                   //
  uint8_t  sensor_state;             //debug only



  TOS_Msg msg_buf;
  TOS_MsgPtr msg_ptr;


 norace uint8_t valueFrmUART; //Added Prasanna
 bool built_in_test;

/*****************************************************/
/*  For Temporal Difference Learning  ******/

uint16_t calib_batt;
uint16_t temperature[TEMP_SAMPLE];

uint16_t sleepTime=1000;


uint8_t cnt=0;
float epsilon = 0.05;
float reward[4];
float V1[2];
float theta[4];

/*********************************************************************
*****
 * Task to send uart and rf message

*********************************************************************
****/

task void setSleepTime()
{

      call Timer.stop();
      call Timer.start(TIMER_REPEAT, sleepTime);

return;
}
    task void send_msg(){
            if (sending_packet) return;
            atomic sending_packet = TRUE;
      pack->xSensorHeader.board_id  = SENSOR_BOARD_ID;
//    pack->xSensorHeader.packet_id = iNextPacketID;
      pack->xSensorHeader.packet_id = valueFrmUART;  //added prasanna
      pack->xSensorHeader.node_id   = TOS_LOCAL_ADDRESS;
//    pack->xSensorHeader.rsvd    = 0;

            call Leds.yellowOn();
            if (IsUART) {
                  if(call Send.send(TOS_UART_ADDR,sizeof(XDataMsg)-
1,msg_ptr)!=SUCCESS)
                        {
                              atomic sending_packet = FALSE;
                              call Leds.greenToggle();
                        }
            }
            else {
                  if(call Send.send(TOS_BCAST_ADDR,sizeof(XDataMsg)-
1,msg_ptr)!=SUCCESS)
                        {
                              atomic sending_packet = FALSE;
                              call Leds.greenToggle();
                        }
                  }
            return;
    }


/*********************************/

task void evalCritic()
{

      uint8_t i;


      // V1[0] will hold the new value at each iteration

      V1[1] = V1[0];
      V1[0] = 0.0;

      for(i=0;i<4;i++)
      {
            V1[0] = V1[0] + (reward[i]*theta[i]);
      }

      if((V1[0]-V1[1]) > 0.0)
      {
            sleepTime = sleepTime-200;   //if newer V is greater than
old V, we need to sleep less to capture more
      }
      else
      {
            sleepTime = sleepTime+500;
      }
      if(sleepTime > 10000 || sleepTime < 200)
            sleepTime = 1000;

      pack->xData.data1.cal_wrod1 = reward[3];
      pack->xData.data1.cal_wrod2 = sleepTime;

return;
}

task void evalReward()
{

      reward[cnt] =
(calib_batt/3.0)*(pow((sleepTime/1000),(epsilon*abs(temperature[0]-
temperature[1]))));
            cnt++;

      if(cnt == 4)
            {
                  cnt=0;
                  post evalCritic();
            }
return;
}

task void populateBattery()
{
      calib_batt = (1252352/pack->xData.data1.vref)/1000;
      return;
}

task void populateTemp()
{
      uint16_t calib_temperature;
      uint8_t front,tail,i;

      front = 0;
      tail = TEMP_SAMPLE-1;
      for(i=tail;i>front;i--)
            temperature[i] = temperature[i-1];

      calib_temperature = -38.4 + (0.0098 * pack-
>xData.data1.temperature);
      temperature[front] = calib_temperature;


return;
}

task void caliberateTempBIT()
{
      uint16_t calib_temp;
      uint16_t return_temp;
      calib_temp = -38.4 + (0.0098 * pack->xData.data1.temperature);
      if(calib_temp>30)
      {
            calib_temp = calib_temp * pow(2,(-1*calib_temp*0.01));
            return_temp = (calib_temp+38.4)/0.0098;
            pack->xData.data1.temperature = return_temp;
      }
return;
}



/*********************************/

  command result_t StdControl.init() {
      uint8_t i;

        atomic {
        msg_ptr = &msg_buf;

        sending_packet = FALSE;
        WaitingForSend = FALSE;
       built_in_test = FALSE;     //added Prasanna
       headptr = 0;               //added Prasanna
       head = 0;                    //added Prasanna
      }
      pack = (XDataMsg *)msg_ptr->data;

      // usart1 is also connected to external serial flash
      // set usart1 lines to correct state
      TOSH_MAKE_FLASH_OUT_OUTPUT();             //tx output
      TOSH_MAKE_FLASH_CLK_OUTPUT();             //usart clk

      call BattControl.init();
      call CommControl.init();
      call Leds.init();

      call TaosControl.init();
      call AccelControl.init();      //initialize accelerometer
      call TempHumControl.init();    //init Sensirion
      call PressureControl.init();   // init Intersema

      for(i=0;i<SAMPLE_SIZE;i++)
            battery[i] = 3;

      for(i=0;i<TEMP_SAMPLE;i++)
            temperature[i] = 0;
      return SUCCESS;
  }

  command result_t StdControl.start() {
      call HumidityError.enable();                //in case Sensirion
doesn't respond
      call TemperatureError.enable();             // same as above

      call CommControl.start();
      call BattControl.start();
      atomic state = START;
      atomic sensor_state= SENSOR_NONE;

      IsUART = TRUE;
      call Timer.start(TIMER_REPEAT, TIMER_PERIOD);    //start up sensor
measurements



      return SUCCESS;
  }

  command result_t StdControl.stop() {
      call BattControl.stop();

      call Timer.stop();
      call CommControl.stop();
      return SUCCESS;
  }


/**********************************************************************
*****
 * Battery Ref  or thermistor data ready

**********************************************************************
****/
  async event result_t ADCBATT.dataReady(uint16_t data) {
      pack->xData.data1.vref = data ;
      post populateBattery();

      atomic state = BATT_DONE;
      return SUCCESS;
  }

      return SUCCESS;
  }
  async event result_t Temperature.dataReady(uint16_t data) {
      pack->xData.data1.temperature = data ;

      post populateTemp();

      if(built_in_test==TRUE)
            post caliberateTempBIT();             // Added Prasanna

      post stopTempHumControl();
      return SUCCESS;
  }

 event result_t Send.sendDone(TOS_MsgPtr msg, result_t success) {

    call Leds.yellowOff();

      if(IsUART){
            msg_ptr = msg;
            IsUART = !IsUART;        // change to radio send
            WaitingForSend = TRUE;   // uart sent, issue radio send
            sending_packet = FALSE;
      }
      else
      {
            IsUART = !IsUART;  // change to uart send
      atomic {
                  WaitingForSend = FALSE;  // both uart and radio sent,
done for current msg
                sending_packet = FALSE;
       }
    }
//post setSleepTime();

   return SUCCESS;
  }


task void receive_task()
{
      //Increase Sleep Time to 6 seconds
      if(valueFrmUART==0x05)
      {
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, TIMER_INC_PERIOD);
      }

      //Decrease Sleep Time to 0.5 seconds
      if(valueFrmUART==0x06)
      {
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, TIMER_DEC_PERIOD);
      }


      // Reset Sleep Time to 1 second
      if(valueFrmUART==0x07)
      {
            call Timer.stop();
            call Timer.start(TIMER_REPEAT, TIMER_PERIOD);
      }

      // Enable/Disable Built In Test for Sensors
      if(valueFrmUART==0x03)
      {
            atomic built_in_test = TRUE;
      }
      if(valueFrmUART==0x04)
      {
            atomic built_in_test = FALSE;
      }


      return;
```

```
}


/************************************************************************
*****
* Process packets recived from UART
************************************************************************
****/
  event TOS_MsgPtr Receive.receive(TOS_MsgPtr data) {
      /*************** Additions *****************/
      TOS_MsgPtr pBuf=NULL;
      XUARTDataMsg *pack_uart;

      pBuf = data;          // Update the pointer, same as saying
copying the received data onto 'mess'
      pack_uart = (XUARTDataMsg *)pBuf;
      if(pBuf)
      {
                  //Does this packet belong to me?
                  if(pack_uart->uartData[5]==TOS_LOCAL_ADDRESS)
                  {
                        valueFrmUART = pack_uart->uartData[6];  //Action
                  type set by our protocol
                  post receive_task();
                  }
      }

      /*******************************************/
      return data;
  }

}
```

/* Matlab Code for Simulating Built-In Test and Data aggregation on simulated data set */

```
clear all; clc;

gWinMin =  20;
gWinMax =  120;

uWinMin =  10;
uWinMax =  150;


s1 = [20 30 80 100 18 15 12 10 120 123 125 130 135 138 140 145 147 155];

[sr sc] = size(s1);
r1 = zeros(1,sc);
Pr = ones(1,sc);
time = 1:sc;

for i=1:sc
    if s1(i) >= gWinMin & s1(i) <= gWinMax
        w1(i) = 1;
        b1(i) = 1;
    else
      if s1(i) < uWinMin | s1(i) > uWinMax
        w1(i) = 0;
        b1(i) = 0;
      else
        w1(i) = exp(-(0.01*s1(i)/2));

        b1(i) = 0;
      end
   end
  Pr(i) = 1-w1(i);
  r1(i) = w1(i) * s1(i);
  rb(i) = b1(i) * s1(i);
end
w1
plot(time,s1,'ko-');
hold on;
plot(time,r1,'r*-');
hold on;
plot(time,rb,'g+-');
figure;
plot(time,Pr)
```

```
clear all; clc;

gWinMin =  20;
gWinMax =  120;

uWinMin =  10;
uWinMax =  150;


s1 = [20 30 80 100 18 15 12 10 120 123 125 130 135 138 140 145 147 155];
s2 = [120 140 150 180 180 180 180 180 180 180 180 180 180  180 180 180
180 180];
s3 = [20 25 30 40 40 40 40 40 40 40 50 40 50 40 40 40 50 40];

[sr sc] = size(s1);
r1 = zeros(1,sc);
Pr = ones(1,sc);
time = 1:sc;

for i=1:sc
    if s1(i) >= gWinMin & s1(i) <= gWinMax
        w1(i) = 1;
        b1(i) = 1;
    else
      if s1(i) < uWinMin | s1(i) > uWinMax
        w1(i) = 0;
        b1(i) = 0;
      else
        w1(i) = exp(-(0.01*s1(i)/2));

        b1(i) = 0;
      end
   end
  Pr(i) = 1-w1(i);
  r1(i) = w1(i) * s1(i);
  rb(i) = b1(i) * s1(i);
end

for i=1:sc
    if s2(i) >= gWinMin & s2(i) <= gWinMax
        w2(i) = 1;
    else
      if s2(i) < uWinMin | s2(i) > uWinMax
        w2(i) = 0;
      else
        w2(i) = exp(-(0.01*s2(i)/2));
      end
   end
  r2(i) = w2(i) * s2(i);
end

for i=1:sc
    if s3(i) >= gWinMin & s3(i) <= gWinMax
        w3(i) = 1;

    else
```

```
      if s3(i) < uWinMin | s3(i) > uWinMax
        w3(i) = 0;

      else
        w3(i) = exp(-(0.01*s3(i)/2));
      end
   end

  r3(i) = w3(i) * s3(i);
end
plot(time,s1,'k+--');
hold on;
plot(time,s2,'k*--');
hold on;
plot(time,s3,'kx--');
hold on;
avg = (s1 + s2 + s3)/3;
plot(time,avg,'r*-');
savg = (r1 + r2 +r3)/3;
hold on;

plot(time,savg,'go-');
```

```
clear all; clc;

load H11;

r1 = H11(1:100,2);      %2,3 and 7th sensors are closely deployed
r2 = H11(1:100,3);
r3 = H11(1:100,7);


w1(1) = 1;
w2(1) = 1;
w3(1) = 1;

[r c] = size(r1);
k = 2;                   % num of neighboring sensors
eps = 1;              % Epsilon

for i=1:r

    agg(i) = (r1(i)*w1(i) + r2(i)*w2(i) + r3(i)*w3(i))/3;

    t1(i) = ((r2(i) + r3(i))/k) - r1(i);
    t2(i) = ((r1(i) + r3(i))/k) - r2(i);
    t3(i) = ((r1(i) + r2(i))/k) - r3(i);

    dw1(i) = abs(t1(i)) * eps;
    dw2(i) = abs(t2(i)) * eps;
    dw3(i) = abs(t3(i)) * eps;

    max = dw1(i);
    flag = 1;
    if (dw2(i) > max)
            max = dw2(i);
            flag = 2;
    end
    if (dw3(i) > max)
            max = dw3(i);
            flag = 3;
    end

    if(flag == 1)
        w1(i+1) = w1(i) - dw1(i);
        w2(i+1) = w2(i) + dw2(i);
        w3(i+1) = w3(i) + dw3(i);
    end
    if (flag == 2)
        w1(i+1) = w1(i) + dw1(i);
        w2(i+1) = w2(i) - dw2(i);
        w3(i+1) = w3(i) + dw3(i);
    end
    if (flag == 3)
        w1(i+1) = w1(i) + dw1(i);
        w2(i+1) = w2(i) + dw2(i);
        w3(i+1) = w3(i) - dw3(i);
    end
```

```
end

plot(w1(1:70),'b-*');
hold on;
plot(w2(1:70),'r-+');
hold on;
plot(w3(1:70),'g-o');

magg = (r1+r2+r3)/3;
%plot(agg,'b*-')
% hold on;
% plot(magg,'ro-')


e = abs(agg' - magg);

%plot(e)
polyfit(r1)
x = 1..70
x = [1:70]
polyfit(x,r1)
polyfit(x',r1(1:70))
polyfit(x',r1(1:70),70)
x
x = x'
y = r1(1:70)
plot(x,y,'0:')
plot(x,y,'O:')
pcoeff = polyfit(x,y,1)
xp = 0:1:70
xp = 1:1:70
yp = polyval(pcoeff,xp)
plot(x,y,'O',xp,yp,'m')
plot(xp,yp,'m')
plot(x,y,'O',xp,yp,'m')
plot(x,y,'O-',xp,yp,'m')
pc2 = polyfit(x,r2(1:70),1)
yp2 = polyval(pc2,xp)
plot(xp,yp2)
figure
plot(xp,yp2)
hold on
plot(xp,yp)
pc3 = polyval(x,r3(1:70))
clear pc3
pc3 = polyfit(x,r3(1:70))
pc3 = polyfit(x,r3(1:70),1)
yp3 = polyval(pc3,xp)
hold on;
plot(xp,yp3)
figure
plot(xp,yp,'b*-')
hold on;
plot(xp,yp,'r*-')
hold on;
plot(xp,yp2,'bo-')
hold on;
```

```
plot(xp,yp3,'k+-')
plot(xp,yp,'r*-')
hold on;
plot(xp,yp2,'bo-')
hold on;
plot(xp,yp3,'k+-')
```

/* C code for reading packets from gateway node through serial interface – read and write to serial port*/

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <termios.h>


#ifdef __CYGWIN__
#include <windows.h>
#include <io.h>
#endif

static const char *g_device= "COM4";
static unsigned g_baudrate = B57600;



unsigned char buffer[39];
unsigned int write_flag = 0;


int port_open()
{
    /* open serline for read/write */
    int serline;
    const char *name = g_device;
    unsigned long baudrate = g_baudrate;

    serline = open(name, O_RDWR | O_NOCTTY);
    if (serline == -1) {
        fprintf(stderr, "Failed to open %s\n", name);
        perror("");
        fprintf(stderr, "Verify that user has permission to open
device.\n");
        exit(2);
    }
    printf("%s input stream opened\n", name);

#ifdef __CYGWIN__
    /* Cygwin requires some specific initialization. */
    HANDLE handle = (HANDLE)get_osfhandle(serline);
    DCB dcb;
    if (!(GetCommState(handle, &dcb) &&
        SetCommState(handle, &dcb))) {
      fprintf(stderr, "serial port initialisation problem\n");
      exit(2);
    }
#endif

    /* Serial port setting */
    struct termios newtio;
    bzero(&newtio, sizeof(newtio));
    newtio.c_cc[VMIN] = 1;
    newtio.c_cflag = CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNBRK | IGNPAR;
    cfsetispeed(&newtio, baudrate);
    cfsetospeed(&newtio, baudrate);
    tcflush(serline, TCIFLUSH);
    tcsetattr(serline, TCSANOW, &newtio);

    return serline;
}

void read_port(int serline)
{
    int cnt;
    int i,count;



    tcflush(serline,TCIOFLUSH);

    printf("Reading buffer ****\n");
    while(i<39) {
      unsigned char c;
      cnt = read(serline, &c, 1);
      if (cnt < 0) {
            perror("error reading from serial port");
          exit(2);
      }
        if (cnt == 1) {

            buffer[i] = c;
            i++;
            printf("%02x", c);
        }
    }
 printf("\n");
}

void write_port(int line)
{
      int flag;
      int k;


      flag = write(line,&buff,sizeof(buff));
      if(flag < 0)
            printf("Write Fail");
      printf("\n");

}




int main()
{
      int k,serline;
    setlinebuf(stdout);
    setlinebuf(stderr);

     serline = port_open();

      while(1)
      {
            read_port(serline);
            write_port(serline);

      }


}
```

/* Reward Function and Critic function evaluation for Reinforcement Learning in Matlab*/

```
clear all;
clc;
% col1  volt  temp col4 col5 sleep

a = load('write.txt');
voltage = a(:,2)/1000;
temperature = a(:,3);
sleep = a(:,6);
epsilon = 0.05;

data = [];
for i=1:length(a) - 1
    r = (voltage(i)/3.0)*
((sleep(i)/1000)^(epsilon*abs(temperature(i+1)-temperature(i))));
    data = [data; [i r voltage(i) sleep(i) abs(temperature(i+1)-
temperature(i))]];
end


gamma = 0.6;
reward = data(:,2);

q = 1;
AIC = [];
savedata.theta = 1;
savedata.error_hist = 1;
savedata.P = 1;
for m = 2:7
    theta = randn(m,1);
    k = 1;
    P = 5*eye(m);
    old_error_hist = [];
    for j = 1:500
        critic = [];
        error_hist = [];

        for i = 10:400
            c_val = reward(i) + gamma*reward(i+1) + gamma^2*reward(i+2)
+ gamma^3*reward(i+3) + gamma^4*reward(i+4) + gamma^5*reward(i+5) +
gamma^6*reward(i+6);
            critic = [critic;[i c_val]];
            psi = [reward(i)];
            for n = 1:m-1
                psi = [psi; reward(i - n)];
            end
            c_val_est = psi'*theta;
            error = c_val - c_val_est;
            error_hist = [error_hist;error^2];
            m_factor = P*psi*error;
            theta = theta + m_factor;
            P = P - P*psi*inv(1+psi'*P*psi)*psi'*P;
            k = k + 1;
        end
        old_error_hist = [old_error_hist;error_hist'];
    end
    error_hist = [sum(old_error_hist)/500]';
    er = log(sum(error_hist))
    AIC = [AIC;[m er]];
    %AIC = log(sum(error_hist(:,2))/5);

    savedata(q).theta = theta;
    savedata(q).error_hist = error_hist;
    savedata(q).P = P;
    q = q + 1;
end

save savedata;
save AIC;

figure;
plot(error_hist);
title('Error history');

critic = [];
for i = 1:length(data) - 6
    c_val = reward(i) + gamma*reward(i+1) + gamma^2*reward(i+2) +
gamma^3*reward(i+3) + gamma^4*reward(i+4) + gamma^5*reward(i+5) +
gamma^6*reward(i+6);
    critic = [critic;[i c_val]];
end
figure;
subplot(4,1,1),plot(data(:,1),data(:,2)); title('Reward');
subplot(4,1,2),plot(critic(:,1),critic(:,2)); title('Critic');
subplot(4,1,3),plot(data(:,1),data(:,3));title('Voltage');
subplot(4,1,4),plot(data(:,1),data(:,4));title('Sleep');

% subplot(4,1,1), plot(r,'g.-')
% subplot(4,1,2), plot(t,'m.')
% subplot(4,1,3), plot(v,'b.')
% subplot(4,1,4), plot(s_time,'k.')
```

```
clear all;
clc;
% col1  volt  temp col4 col5 sleep

a = load('write.txt');
voltage = a(:,2)/1000;
temperature = a(:,3);
sleep = a(:,6);
epsilon = 0.05;

data = [];
for i=1:length(a) - 1
r = (voltage(i)/3.0)* ((sleep(i)/1000)^(epsilon*abs(temperature(i+1)-
temperature(i))));
data = [data; [i r voltage(i) sleep(i) abs(temperature(i+1)-
temperature(i))]];
end

critic_data = [];
gamma = 0.6;
reward = data(:,2);
load savedata;
load AIC;
[min_val ind] = min(AIC(:,2));

theta = savedata(ind).theta;

m = length(theta);
    for i = 7:400
        c_val = reward(i) + gamma*reward(i+1) + gamma^2*reward(i+2) +
gamma^3*reward(i+3) + gamma^4*reward(i+4) + gamma^5*reward(i+5) +
gamma^6*reward(i+6);
       psi = [reward(i)];
            for n = 1:m-1
                psi = [psi; reward(i - n)];
            end
            c_val_est = psi'*theta;
            error = c_val - c_val_est;
        c_val
        c_val_est
        error
        critic_data = [critic_data; [i c_val c_val_est error]];
    end


figure;
plot(critic_data(:,1),critic_data(:,2),'-
',critic_data(:,1),critic_data(:,3),'--
',critic_data(:,1),critic_data(:,4),'.');
legend('Total expected reward','Critic','Error');

% subplot(4,1,1), plot(r,'g.-')
% subplot(4,1,2), plot(t,'m.')
% subplot(4,1,3), plot(v,'b.')
% subplot(4,1,4), plot(s_time,'k.')
```

```
/*   Sample program to implement CRC so that packets can be sent from PC to gateway
node using serial interface (UART)*/


#include<stdio.h>


char buffer[37];
int length;
int crc;


calByte(char b)
{
        int i;
        crc = crc^(int)b<<8;
        for(i=0;i<8;i++)
        {
                if((crc & 0x8000)== 0x8000)
                        crc = crc << 1 ^ 0x1021;
                else
                        crc = crc << 1;
        }
}



calc()
{
        int k=0;
        while(length>0)
        {
                calByte(buffer[k]);
                length--;k++;
        }
        crc = crc & 0xffff;
        printf("%x",crc);
}

main()
{
buffer[0] = 0x7e;
buffer[1] = 0x42;
buffer[2] = 0xff;
buffer[3] = 0xff;
buffer[4] = 0x00;
buffer[5] = 0x7d;
buffer[6] = 0x5d;
buffer[7] = 0x1d;
buffer[8] = 0x85;
buffer[9] = 0x01;
buffer[10] = 0x03;
buffer[11] = 0x01;
buffer[12] = 0xa6;
buffer[13] = 0x01;
buffer[14] = 0x1d;
buffer[15] = 0x03;
buffer[16] = 0x53;
buffer[17] = 0x19;
buffer[18] = 0x80;
buffer[19] = 0xb0;
buffer[20] = 0x18;
buffer[21] = 0xb3;
buffer[22] = 0xe3;
buffer[23] = 0x99;
buffer[24] = 0x9d;
buffer[25] = 0xb4;
buffer[26] = 0xf2;
buffer[27] = 0x67;
buffer[28] = 0x7a;
buffer[29] = 0x47;
buffer[30] = 0xfb;
buffer[31] = 0x00;
buffer[32] = 0x00;
buffer[33] = 0x00;
buffer[34] = 0x15;
buffer[35] = 0x00;
buffer[36] = 0xf0;

length=35;


calc();

}
```

/* Middleware Design */

TOSBaseM.nc (program on Gateway/Base-station node)

1. RECEIVE:

Radio: - Recieve From other motes
When packets arrive, the TOS_Msg is encapsulated in a Frame.
So p->data[9] p->data[8] for example represents Temperature reading
p->data[26] p->data[27] represents Accel_X value.
The array structure is represented in TestMTS400M.nc

UART: - Will recieve from Computer
- Need for CRC in each packet, otherwise, the gateway node rejects packet

2. SEND:

UART: - Send UART message to Computer.
- Any radio packet received is forwarded to computer.
- Takes TOS_Msg and encapsulates it inside Frame and sends it.
- At the computer end, read COM port to buffer.
- 39 bytes of data is read.
- buffer[34] = p->data[27]

Radio: - Broadcast any UART packet to all motes
- Transmit TOS_Msg

MTS400M.nc (program on each sensor node)

1. RECEIVE:
UART: - No UART receive

Radio: - Receive TOS_Msg
- pack_uart->uartData[0] = buffer[3] ===> unframe the packet

2. SEND:
UART: - No UART send

Radio: - Send all sensor values broadcast
Sample packet:
7e42ffff007d5d1d8501030125021f04ca19ffff18ffffffffb4ff7f6547ff0000001400f603