DISTRIBUTED INCREMENTAL DATA STREAM MINING FOR WIRELESS SENSOR NETWORK

HAKILO AHMED SABIT

A thesis submitted to

Auckland University of Technology

in fulfilment of the requirements for the degree of

Doctor of Philosophy (PhD)

2013

School of Engineering

CONTENTS

Declaration

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

Auckland, 2013

_____

Hakilo Sabit

LIST OF FIGURES

LIST OF TABLES

Algorithms and Code Listings

LIST OF ABBREVIATIONS

1D	One Dimension
2D	Two Dimensions
3D	Three Dimensions
4D	Four Dimensions
6LoPAN	IPv6 over Low power Wireless Personal Area Networks
AASC	American Association of State Climatologists
AC	Alternating Current
ACK	Acknowledgement
ACKL	Auxiliary Clock
ADC	Analog to Digital Converter
AI	Artificial Intelligence
AM	Active mode
AODV	Ad Hoc On-Demand Distance Vector
AR	Accept Ratio
ATM	Automated Teller Machine
BUI	Build Up Index
CH	Cluster Head
CPU	Central Processing Unit
CSMA/CA	Carrier Sense Medium Access/Collision Avoidance
CVFDT	Concept Adapting Very Fast Decision Tree
DAC	Digital to Analog Converter
DC	Drought Code
DCO	Digitally Controller Oscillator
DMA	Direct Memory Access
DMC	Duff Moisture Code
DSP	Digital Signal processing
DTC	Data Transfer Controller
EDACH	Energy-Driven Adaptive Clustering Hierarchy
EEPROM	Electrically Erasable Programmable Read-Only Memory
EFDT	Efficient-VFDT

EPA	Environmental Protection Agency
FCM	Fuzzy Cluster Means
FDI	Fire Danger Index
FFMC	Fine Fuel Moisture Code
FWI	Fire Weather Index
GIS	Geographic Information System
GSB	Global Stream Base
GSM	Global System for Mobile Communications
I/O	Input/output
IFTF	Internet Engineering Task Force
IPv6	Internet Protocol Version 6
ISI	Initial Spread Index
ISM	Industrial Scientific Medical
ISP	In-System Programming
KB	Kilo Byte
LEACH	Low-Energy Adaptive Clustering Hierarchy
LED	Light Emitting Diode
LF	Low Frequency
LPM	Low Power Mode
LSB	Local Stream Base
LW-WPAN	Low Rate Wireless Personal Area Network
MAC	Medium Access Control
MCLK	Main Clock
MCU	Micro Controller Unit
MIP	Million Instructions Per Second
MPH	Miles Per Hour
NIWA	National Institute of Water and Atmospheric Research
OS	Operating System
PC	Personal Computer
PCA	Principal Component Analysis
QoS	Quality of Service
RAM	Random Access Memory
RH	Relative Humidity
RISC	Reduced Instruction Set Computing
RR	Reject ratio
RTD	Resistance Temperature Detector
SF	Squash Factor
SMCLK	Sub-Main Clock
SoC	System on Chip
SUBFCM	Subtractive Fuzzy Cluster Means
TCP	Transfer Control Protocol
TI	Texas Instruments

VFDT	Very Fast Decision Tree
WLAN	Wireless Local Area Network
WMO	World Meteorological Organization
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
ZC	ZigBee Coordinator
ZED	ZigBee End Device
ZNP	ZigBee Network Processor
ZR	ZigBee Router

ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, praise for the one above all of us, the Almighty Allah, for answering my prayers for giving me the strength to complete this research.

My family, specially my mom for the absolute love and support throughout my life, thank you so much mom.

My utmost gratitude to my Primary Supervisor Professor Adnan Al-Anbuky, Director of Sensor Network and Smart Environment Research Centre (*SeNSe*), School of Engineering, Auckland University of Technology (AUT), whose sincerity and encouragement I will never forget. Professor Adnan Al-Anbuky has been my inspiration as I hurdle all the obstacles in the completion this research work.

My co-supervisor Dr. Hamid Gholamhosseini for his valuable advices throughout the research period.

I would like to thank Auckland University of Technology for the financial support in the form of fellowship. Special thanks to the Administrators of the School of Engineering Auckland University of Technology.

Last but not the least, my colleagues at the Sensor Network and Smart Environment Research Centre (*SeNSe*) for their support and friendship thank you so much.

ABSTRACT

Wireless sensor networks (WSNs) despite their energy, bandwidth, storage, and computational power constraints, have embraced dynamic applications. These applications generate a large amount of data continuously at high speeds and at distributed locations, known as *distributed data stream*. In these applications, processing data streams on the fly and in distributed locations is necessary mainly due to three reasons. Firstly, the large volume of data that these systems generate is beyond the storage capacity of the system. Secondly, transmitting such large continuous data to a central processing location over the air exhausts the energy of the system rapidly and limits its lifetime. Thirdly, these applications implement dynamic models that are triggered immediately in response to events such as changes in the environment or changes in set of conditions and hence, do not tolerate offline processing. Therefore, it is important to design efficient distributed techniques for WSN data stream mining applications under these inherent constraints.

The purpose of this study was to develop a resource efficient online distributed incremental data stream mining framework for WSNs. The framework must minimize inter-node communications and optimize local computation and energy efficiency without compromising practical application requirements and quality of service (QoS). The objectives were to address the WSN energy constraints, network lifetime, and distributed mining of streaming data. Another objective was to develop a novel high spatiotemporal resolution version of the standard Canadian fire weather index (FWI) system called the Micro-scale FWI system based on the framework.

The perceived framework integrates autonomous cluster based data stream mining technique and two-tiered hierarchical WSN architecture to suit the distributed nature of WSN and on the fly stream mining requirements. The underlying principle of the framework is to handle the sensor stream mining process in-network at distributed locations and at multiple hierarchical levels. The approach consists of three distinct

processing tasks asynchronously but cooperatively revealing mining the sensor data streams. These tasks are the sensor node, the cluster head, and the network sink processing tasks. These tasks were formulated by a lightweight autonomous data clustering algorithm called Subtractive Fuzzy C-Means (SUBFCM). The SUBFCM algorithm remains embedded within the individual nodes to analyze the locally generated streams 'on the fly' in cooperation with a group of nodes.

The study examined the effects of data stream characteristics such as data stream dimensions and stream periods (data flow rates). Moreover, it evaluated the effects of network architectures such as node density per cluster and tolerated approximation error on the overall performance of the SUBFCM through simulations. Finally, the QoS or certain level of guaranteed performance that is supported by the WSN architecture for applications utilizing the framework was examined.

The results of the study showed that the proposed framework is stream dimension and data flow rate scalable with average errors of less than 12% and 11% in reference to the benchmarks, respectively. The node density per cluster and local model drift threshold showed significant effects on the framework performance only for very fast streams.

The study concludes that the network architecture is an important factor for the quality of mining results and should be designed carefully to optimally utilize basic concepts of the framework. The overall mining quality is directly related to the combined effect of the stream characteristics, the network architecture, and the desired performance measures. The study also concludes that WSNs can provide good QoS feasible for online distributed incremental data stream mining applications.

Simulations of real weather datasets indicate that the Micro-scale FWI can excellently approximate the results obtained from the Standard FWI system while providing highly superior spatial and temporal information. This can offer direct local and global interaction with a few meter square spaces as against the tens of square kilometers of the present systems.

Chapter 1

1. INTRODUCTION

Wireless sensor networks (WSNs) despite their energy, bandwidth, storage, and computational power constraints, have embraced dynamic applications that generate a large amount of data continuously at high speeds and sourced from distributed locations, known as *distributed data stream*. In these applications, processing data streams on the fly and in distributed locations is necessary mainly due to three reasons. Firstly, the large volume of data that these systems generate is beyond the storage capacity of the system. Secondly, transmitting such large amount of continuous data to a central processing location over the air exhausts the energy of the system rapidly and limits its lifetime. And thirdly, these applications implement dynamic models that are triggered immediately in response to events such as changes in the environment or changes in a set of conditions, hence do not tolerate offline processing. Therefore, designing effective distributed techniques for WSN data stream mining applications under the inherent constraints becomes important. We have developed an efficient online distributed incremental data stream mining for WSNs using a distributed clustering technique.

1.1. BACKGROUND TO THE RESEARCH

Data streaming is an inherent feature of a WSNs [1] and a number of research has been dedicated in mining the data streaming from such networks [2-6]. Data mining is a promising and relatively new technology that is defined as a process of discovering hidden valuable and useful knowledge or information by analyzing large amounts of

data storing in databases or data warehouse using different techniques such as machine learning, artificial intelligence (AI) and statistical. Data mining techniques can discover information that many traditional business analysis and statistical techniques fail to deliver [7]. A variety of real time applications produce distributed continuous data streams and require data stream mining. Data stream mining is the extraction of structures of knowledge that are represented in the case of models and patterns of infinite streams of information [8]. There are several major data mining techniques developed and used in data mining projects recently including association, classification, clustering, prediction and sequential patterns. We will briefly introduce those data mining techniques with example to have a good overview of them followed by a brief description of WSNs and their unique characteristics and the challenges they pose to stream mining techniques.

1.1.1. Association

Association is one of the best known data mining techniques. In association, a pattern is discovered based on a relationship of a particular item on other items in the same transaction. For example, the association technique is used in market basket analysis to identify what products customers frequently purchase together. Based on this data businesses can have corresponding marketing campaigns to sell more products to make more profit. The patterns discovered with this data mining technique can be represented in the form of association rules [9, 10]. The domain experts set the two measures of rule interestingness threshold which are rule support and confidence. The association rules are considered interesting if they satisfy minimum support threshold and minimum confidence threshold.

1.1.2. Classification

Classification is a classic data mining technique based on machine learning. Basically classification is used to classify each item in a set of data into one of predefined set of classes or groups. Classification methods make use of mathematical techniques such as

decision trees, linear programming, neural networks and statistics. In classification, we develop the software that can learn how to classify the data items into groups. For example, we can apply classification in an application that “given all past records of employees who left the company, predicts which current employees are likely going to leave in the future.” In this case, we divide the employees’ records into two groups that are “leave” and “stay”. Then we can ask our data mining software to classify the employees into each group.

1.1.3. Prediction

The prediction as its name implies is one of a data mining techniques that discovers relationships between independent variables and relationships between dependent and independent variables. For instance, prediction analysis technique can be used in sale to predict profit for the future if we consider sale as an independent variable, profit could be a dependent variable. Then based on the historical sale and profit data, we can draw a fitted regression curve that is used for profit prediction.

1.1.4. Sequential Patterns

Sequential patterns analysis is a data mining technique that seeks to discover similar patterns in data transactions over a business period. The discovered patterns are used for further business analysis to recognize relationships among data.

1.1.5. Clustering

Clustering is a data mining technique that makes meaningful or useful clusters of objects that have similar characteristics using an automatic technique. Different from classification, clustering technique also defines the classes and puts objects into them, while in classification objects are assigned into predefined classes. To make the concept clearer, we can take library as an example. In a library, books have a wide

range of topics available. The challenge is how to keep those books in a way that readers can take several books in a specific topic without hassle. Using clustering technique, we can keep books that have some kind of similarities in one cluster or one shelf and label it with a meaningful name. If readers want to grab books in a topic, he or she would only go to that shelf instead of looking through the whole library.

1.1.6. Wireless Sensor Network Background

Advances in recent technologies have allowed the development of low cost small sensors with the capabilities of sensing physical environment, computing, data processing and storing, and communicating wirelessly with other sensors. These sensors can integrate with each other without any fixed or centralized infrastructure to form a network, the WSN, that is able to monitor the environment and transmit detected events to a well equipped node called the Sink. A WSN consists of a large number of sensors [11], each of which are physically small devices, and are equipped with processing capability (one or more microcontrollers, CPUs or DSP chips), multiple types of memories (program, data and flash memories), RF transceiver (usually with a single Omni-directional antenna), a power source (e.g., batteries and solar cells), and various sensors and actuators. Due to size and cost constraints, sensors in WSNs have certain intrinsic constraints on resources such as energy, memory, computational capabilities, and communications bandwidth. WSN deployment consists of spatially distributed autonomous sensors connected via a wireless communication infrastructure to cooperatively monitor, record, and store physical or environmental conditions such as temperature, humidity, light, sound, vibration, pressure, motion or pollutants.

The sensor nodes communicate wirelessly and often self-organize after being deployed in an ad hoc fashion whereby a group of sensor nodes spontaneously form a network without any fixed and central infrastructure. Therefore, they can be deployed in inaccessible locations by aerial drop to form a cooperative monitoring network. When two nodes in a WSN wish to communicate, intermediate nodes are called upon to forward packets to form a multi-hop wireless route. WSNs deploy a sheer number of

sensor nodes and due to the large probability that many of them will be sensing events in close proximity and simultaneously, they enable multi-projection of an event and hence open the door for several unique applications. The WSN technology is exciting with unlimited potential for numerous application areas including environmental, medical, military, transportation, entertainment, crisis management, homeland defense, and smart meters [12].

With the advances in WSNs and their ability to generate a large amount of data, data mining techniques to extract useful knowledge regarding the underlying network have recently received a great deal of attention [11]. However, the stream nature of the data, the limited resources, and the distributed nature of sensor networks bring new challenges for the mining techniques that need to be addressed. These challenges are further amplified when the data they generate is of a continuous stream in nature.

1.2. MOTIVATION

More recently the need to process a large amount of data has motivated the field of data mining whereby ways are investigated to process the static data sets efficiently and algorithms are developed to compute the final static model representing the data sets [13]. However this data mining approach despite handling large data sets does not address the problem of a continuous supply of data. A model that was previously induced cannot be updated as new data arrives. Instead, the entire training process must be repeated with the new examples included. This is undesirable and inefficient for many continuous data streaming systems.

The deployment of pervasive communication infrastructures such as short-range wireless ad hoc sensor networks has enabled the capture of different measurement of data in a wide range of fields. These measurements are generated continuously and at high data rates. Such continuous flows of data grow rapidly over time and are known as data streams. Examples include sensor networks, web searches, phone conversations, and network traffic. Data streams necessitate the need for new

applications that process, analyze, and react to data streams in a near real-time manner.

Consider for instance, a body area network [14] in a health care system monitoring patients' health conditions. The sensor nodes have to continuously take vital signs measurements and feed into a model that alerts the care giver when a health risk condition is detected. Further, consider a WSN deployed to monitor forest Fire Weather Indices (FWI) and alert fire hazard in real-time. The sensor nodes continuously measure weather parameters and feed into the FWI model, which triggers fire hazard alarm whenever high fire risk conditions are detected. These systems naturally do not tolerate offline data analysis. Therefore, the data streaming from such systems has to be processed on the fly and in real-time.

1.3. RESEARCH PROBLEM

Given a distributed sensor system consisting of resource constrained sensor nodes and connected via an underlying wireless network. Each node is tasked with probing its proximity and updating the system regarding its acquired information instantly and periodically along with all other nodes in the system. The nodes continue repeating their task indefinitely in short periods. At each short period, the system is required to find patterns within the update information received from all nodes in real-time and keep up with the continuous periodic update of information arrivals. This research aims at answering the following question: How can update information stream mining tasks for extracting patterns in real-time from the union of all nodes' information be executed in the system with all nodes participating in a collaborative distributed computation such that the energy, computational power, and communication bandwidth resources are efficiently utilized?. The research also aims at answering the following specific questions: How can temporal correlation of dynamic situations dispersed over a given geographic area be continuously captured while efficiently utilizing the scarce WSN system resource? How can the spatial correlation of dynamic situations dispersed over a given geographic area be continuously captured while

efficiently utilizing the scarce WSN system resource? How these temporal and spatial patterns capturing tasks can execute on the fly and in real-time in WSN systems?

1.4. CONTRIBUTIONS

The thesis has systematically studied the limitations of existing stream mining techniques for WSNs in terms of their power consumption, computational power, and communication efficiencies and provided alternative architectures for real life applications of distributed data stream mining. The specific contributions of the thesis are:

1. Proposing an efficient architecture for optimizing power consumption, computational power, and communication bandwidth for distributed incremental data stream mining for resource constrained WSNs.

2. A real-time online distributed data stream mining WSN system framework efficiently utilizing the scarce WSN system resources.

3. A generic real-time online distributed data stream mining WSN system simulation model, which can be used to design and analyze WSN systems for distributed data stream mining applications before building and deploying the actual system.

4. Proposing a high spatio-temporal resolution FWI system (Micro-scale FWI) for a forest fire danger monitoring WSN application utilizing the distributed incremental data stream mining WSN model.

These contributions can be applied to WSN design and deployment for online distributed data stream mining applications utilizing a resource efficient architecture. Furthermore, this research provides a foundation for future investigation of high spatio-temporal resolution forest fire monitoring.

1.5. OUTLINE OF THE THESIS

The thesis is organized into 9 chapters, which include Introduction; Literature Review; Research Materials; Theory of Distributed Incremental Data Stream Mining WSN; Development of SUBFCM algorithm; Modeling and Simulation of the Distributed Incremental Data Stream Mining WSN; Results and Analysis ; Case study; and conclusion and future scope.

Chapter one starts with a brief introduction of WSN, data stream mining techniques and describes the motivation behind this research, states the specific problem addressed, highlights the contributions of the thesis and gives an overview of the structure of the thesis.

Chapter two presents an overview of WSN architecture, multi-level energy conservation strategies, WSN relevant data stream mining techniques, distributed data stream mining framework; it analyses the state-of-the-art in the distributed data stream mining techniques literature for WSN, and highlights the research gaps.

In chapter three, the hardware and software tools and materials used during the research and their capabilities and limitations are examined.

Chapter four presents the theoretical framework of the distributed incremental data stream mining technique for the WSN system. It describes the details of the WSN architecture to support the distributed data stream clustering technique.

Chapter five presents the development of the core stream mining algorithm that is embedded and runs within the cluster head nodes- the subtractive fuzzy cluster means (SUBFCM) algorithm.

Chapter six presents a detailed description of the distributed incremental data stream mining WSN system modeling and simulation. The implementation of the individual modules of the system model is also presented in this chapter.

Chapter seven presents the results, performance evaluation and analysis of the research.

Chapter eight presents the case study for the distributed incremental data stream mining WSN application. The Micro-scale forest fire weather index application of WSN is presented.

Chapter nine concludes this thesis and outlines the directions for future research in distributed data stream mining for resource constrained systems such as WSN.

1.6. PUBLICATIONS

During this study, the following five international peer reviewed publications have been produced that include international journal and conference proceedings.

- Sabit, H., Al-Anbuky, A., and Gholamhosseini, H. (2011). Data stream mining for wireless sensor networks environment: energy efficient fuzzy clustering algorithm. *International Journal of Autonomous and Adaptive Communications Systems*, 4(4):383-397.
- Hakilo Sabit, Adnan Al-Anbuky, Hamid GholamHosseini, Wireless Sensor Network Based Wildfire Hazard Prediction System Modeling, Procedia Computer Science, Volume 5, 2011, Pages 106-114.
- Sabit, H., Anbuky, A. A., and Hosseini, H. G. (2009). Distributed WSN data stream mining based on fuzzy clustering. In *Proceedings of the 2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, UIC-ATC '09, pages 395-400, Washington, DC, USA. IEEE Computer Society.
- Sabit, H., and Al-Anbuky, A. 2011. Sensor Network & Weather Data Stream Mining. Proceedings of Bushfire CRC & AFAC 2011 Conference Science Day' 1 September 2011, Sydney Australia, Bushfire CRC.
- Sabit, H., Al-Anbuky, A., and Gholamhosseini, H. 2011. Micro-scale Forest Fire Weather Index and Sensor Network. Proceedings of Bushfire CRC & AFAC 2011 Conference Science Day' 1 September 2011, Sydney Australia, Bushfire CRC.

1.7. CHAPTER SUMMARY

This chapter presented an introduction to data stream mining and related techniques relevant to WSNs. The chapter also described background to the research giving specific examples to motivate the research, as well as a problem statement and specific research questions. It further provided a list of contributions during the research period and outline of the thesis.

Chapter 2

2. LITERATURE REVIEW

2.1. INTRODUCTION

This chapter introduces and analyses the literature on distributed data stream mining and the state of the art of the field of distributed data stream mining in the context of WSNs. The field of WSN focuses on the design and operation of wireless personal area network (WPAN) based sensor systems consisting of distributed autonomous devices to cooperatively monitor an environment but with very strong constraints on resources such as energy, memory, computational speed and bandwidth. Data stream mining focuses on the design of processes and algorithms that enable computing nodes to extract knowledge structures from continuous, rapid data records. Distributed techniques for WSN data stream mining focuses on designing scalable and reliable stream mining methods to suit the distributed nature of WSN and online stream mining goals for large scale real-time stream mining systems. A review of the foundations of each of these fields is presented below.

2.2. WIRELESS SENSOR NETWORK INTRODUCTION

Wireless sensor networks are networks consisting of small multimodal sensor devices/nodes capable of limited processing power, short range communication and

limited memory space. In other words, WSNs consist of hundreds or even thousands of resource-constrained nodes. These devices are of low cost, small footprint, and individually unreliable. However, deployed in large quantities and observing in a spatially overlapping areas, they provide a system of high reliability and unprecedented measurement resolution. In recent years, WSNs have found ever increasing and diverse applications particularly in monitoring and control systems. They have been successfully utilized in some of socially and scientifically significant applications: water quality monitoring [15, 16], climate change modeling [17], industrial plants monitoring [18-20], personal health monitoring [21, 22], structural health monitoring [23-25], wild fire detection [26], etc. Source nodes and sink node(s) constitute the basic WSN architecture. The source nodes host multimodal sensors, a low power processor, a radio transceiver, and light weight battery. They are the sources of the network data traffic. The sink contains a processor, a radio transceiver, no sensors, and is usually mains powered. The sink node is the coordinator and collection point for network data traffic. The sink is usually interfaced to the base station PC (data management station) which could be a gateway to another external network (e.g., the internet). Owing to their small footprint and wireless communication capabilities, the sensor nodes can be placed in hostile and inaccessible locations to quantify and transmit the current state of the phenomena under observation. Due to the hostility and inaccessibility of their location, the task of battery replacement is hard if not impossible. Hence, WSNs require an aggressive energy conservation strategy for each battery-powered device in order to operate for a meaningful period of time before dropping out of the network.

Dead node replacement could be a means of restoring normal network operation if there is no minimum uninterruptible operation period requirement by the application and as long as the application can withstand the delay during network reconfigurations. The time and cost of redeployment and reconfiguration are however, prohibitive for most applications. In order to obtain a good network lifetime for a WSN application, a number of energy conservation strategies have to be considered. These energy conservation strategies concern the WSN hardware platforms, the wireless protocol stack, and the application.

2.2.1. Hardware Platform

A WSN consists of spatially distributed sensor nodes. Each sensor node is capable of probing its environment and limited independent processing. The WSN nodes are also capable of short range communication due to their radio transceiver which, allows them to forward their sensed information to a central sink node. Furthermore, WSN nodes can perform local coordination. Among the common sensor node platforms are Crossbow technology MICAz, TMote Sky, Sun Microsystems Sun SPOT, and Texas Instruments CC2530. The sensor nodes' basic hardware components are an embedded processor, a radio transceiver, memories, a power source, and sensors.

The WSN platform energy conservation strategies include efficient sensors energizing, efficient device power mode setting, setting optimal transmission power, duty cycling, etc.

2.2.1.1. Sensor Energizing

Sensor transducers are an integral part of sensor nodes. Sensors, either analog or digital, translate physical phenomena into electrical signals. The sources of energy consumption in sensor transducers can be signal sampling, conversion of physical signals to electrical ones, signal conditioning, and analog to digital conversions [27]. Depending on the nature of their sensing mechanism, sensors consume different amounts of energy. Passive sensors such as resistance temperature detectors (RTDs) and photodiodes, for instance, consume much less energy than active ones such as sonar rangers and strain gauges. The sampling rate also plays a major role in sensor energy consumption. The higher the sampling rate, the higher is the energy consumption [28, 29]. Another strategy in practice is employing low power and higher error rate detector sensors before actually energizing the higher power consuming higher quality sensors [30].

2.2.1.2. Power Mode Settings

The microcontroller unit is at the heart of the sensor nodes controlling the sensors and execution of the communication protocols and signal processing algorithms [27]. The microcontrollers support various operating modes, including active, idle, and multi level sleep modes. Each mode is characterized by a different amount of power consumption. The different power modes are achieved by switching off some functional components of the microcontroller and hence, the power modes have different functional capabilities. Depending on the application requirements, valuable energy can be conserved by switching between the different power modes. For data stream applications, which are the focus of this research, we consider a multi power mode operation whereby all nodes remain in low power mode, briefly wake up, sample their sensors periodically and send the data to their local sink.

2.2.1.3. Optimal Transmission Power Setting

In WSN, transmitting data at unnecessarily high power not only reduces the lifetime of the nodes and the network, but also introduces excessive interference [31]. The transmit power of the nodes determine the connectivity level of ad hoc wireless networks [31]; however, transmitting at excessive power levels increase mutual interference in the shared radio channel and limit the battery power. Therefore, the optimal transmit power with respect to network lifetime and connectivity sufficient to guarantee network connectivity [32, 33] should be a design consideration for efficient WSNs. In [34], the optimal transmit power is derived for a random topology.

2.2.1.4. Duty Cycling

To resolve the conflict between limited energy and application lifetime requirements, it is necessary to reduce node communication and sensing duty cycles [35]. Periodic interval sensing is also used as sensor nodes' energy conserving mechanism where the nodes remain in off mode during the inactive duty cycles [35, 36]. Considering data streaming applications, especially high speed data streaming applications, the amount of time the nodes remain in inactive mode is very low. Therefore, duty cycling

becomes efficient only when the energy consumption of switching between the active and inactive cycles is significantly low.

2.2.2. Operating System

Operating systems (OSs) for WSN nodes are typically simple and less complex than general purpose OSs because of the resource constraints in hardware platforms and also because of specific requirements of WSN applications. In many cases, simply round robin based task scheduling suffices for specific WSN applications. A free and open source component-based operating system and platform targeting wireless sensor networks, TinyOS, is perhaps the first operating system specifically designed for WSNs.

TinyOS programs are built out of software component libraries which include network protocols, distributed services, sensor drivers, and data acquisition tools. The TinyOS component libraries can be customized for application requirements. TinyOs programming is based on an event-driven mode rather than multithreading. TinyOS programs are composed into event handlers and tasks with run-to-completion semantics. The TinyOS system and programs written for TinyOS are written in a special extension of the C programming language called nesC.

Contiki is another highly portable open source OS specially developed for WSNs. Contiki is an event-driven operating system, but it supports multithreading unlike TinyOS. The Contiki operating system provides an IP communication stack, both IPv4 and IPv6, with a very small memory footprint. There are also some new operating systems for WSN such as LiteOS.

2.2.3. WSN Protocol Stack

A WSN is an ad-hoc arrangement of multifunctional sensor nodes in a sensor field, usually to gather information regarding some phenomenon. Sensor nodes can be densely distributed over a large even remote area and can continue to collaborate their efforts to the benefit of the network even if a number of nodes malfunction.

There are two main layouts for WSNs. The first is a star layout where the nodes communicate, in a single hop, directly to the sink whenever possible and peer-to-peer communication is minimal. In the second, information is routed back to the sink via data passing between nodes. This multi-hop communication is expected to consume less power than single-hop communication because nodes in the sensor field are densely distributed and are relatively close to each other.

A sensor network protocol stack is similar to the traditional protocol stack. The WSN protocol stack consists of application, transport, network, data link, and physical layers. The physical layer is responsible for frequency selection, carrier frequency generation, signal detection, modulation and data encryption. The data link layer is responsible for the multiplexing of data streams, data frame detection, medium access and error control. It ensures reliable point-to-point and point-to-multipoint connections in a communication network. The network layer takes care of routing the data supplied by the transport layer. The network layer design in WSNs must consider the power efficiency, data-centric communication, data aggregation, etc. The transportation layer helps to maintain the data flow and may be important if WSNs are planned to be accessed through the Internet or other external networks. Depending on the sensing tasks, different types of application software can be set up and used on the application layer.

2.2.3.1. IEEE 802.15.4/ ZigBee

The IEEE 802.15.4 Standard, introduced in 2003 is designed to address the need for a low cost and low power wireless solutions and has become the foundation for monitoring and control solutions, including ZigBee technology, SynkroRF technology, the WirelessHART specification, WiMi specification as well as numerous other proprietary network stacks.

ZigBee has been the de-facto protocol stack for low-cost, low-power WSN devices. Based on the IEEE 802.15.4 MAC and physical layer standard [37], the ZigBee specification defines an architecture for sensor networks that comprises a network layer, an application support layer, as well as a security managing unit. ZigBee is a

wireless mesh network standard which operates in the industrial, scientific, and medical (ISM) radio bands; 868 MHz in Europe, 915 MHz in the USA and Australia, and 2.4 GHz worldwide. ZigBee supports data transfer rate of 10Kbps, 20Kbps, and 40Kbps at 686 MHz, 915 MHz, and 2.4 GHz bands respectively. The ZigBee network layer natively supports star and tree networks, and generic mesh networks. Every ZigBee network must have one coordinator device tasked with its creation, control and maintenance. The star and mesh configurations allow use of ZigBee routers to extend communication at the network level.

The ZigBee specification consists of three types of devices; the ZigBee coordinator (ZC), ZigBee Router (ZR), and ZigBee End Device (ZED). ZC is a full function device that forms the root of the network. In a ZigBee network, there will only be one ZC. ZR is a fully functional device that can run applications functions, as well as act as an intermediate router passing on data from other devices. A ZigBee network can have multiple ZR devices. ZED is a reduced function device that can only talk to the parent nodes (full function devices) and is not able to relay data from other devices. A typical ZigBee network contains more ZigBee end devices (ZEDs) than ZigBee routers (ZRs).

ZigBee network protocols support beacon and non-beacon enabled networks. In beacon-enabled networks, ZRs transmit periodic beacons to confirm their presence to the other network nodes. Nodes may sleep between beacons, thus lowering their duty cycle and extending their battery life. In non-beacon-enabled networks, an unslotted CSMA/CA channel access mechanism is used. In non-beacon-enabled networks, ZRs have their receiver continuously active, which allows some devices to receive continuously and others to transmit on an external stimulus.

2.2.3.2. 6LoWPAN

More recently, the Internet Engineering Task Force (IFTF) has defined 6LoWPAN standard which enables IPv6 connectivity over Low Power Personal Area Networks. The 6lowpan group has defined encapsulation and header compression mechanisms that allow IPv6 packets to be sent and received over from over IEEE 802.15.4 based networks. The 6LoWPAN concept originated from the idea that "the Internet Protocol

could and should be applied even to the smallest devices," [38] and that low-power devices with limited processing capabilities should be able to participate in the Internet of Things [39].

2.2.4. Protocol Level Energy Conservation

The WSN protocol stack energy conservation concerns optimal topology formation, method of packet routing, and route discovery/rediscovery.

Resource constraints are one of the major drawbacks on sensor networks. Since sensor nodes run on battery, which cannot be replenished, it is vital that it runs very efficiently, in terms of sensing, computation and communication. Sensing and computation activities, compared to communication are very efficient. It is the communication activities of transmitting and receiving which take up most of the energy. So *resource awareness* should be inbuilt in the protocol architecture for efficient communication. Enhancing power efficiency can be achieved in the entire network protocol stack of wireless ad hoc networks [40]; topology formation, MAC protocol, and routing protocol.

Topology formation is an important issue in a WSN. Performance parameters such as energy consumption, network lifetime, data delivery delay, sensor field coverage depend on the network topology [41].

MAC protocols control the communication modes in WSNs and regulate access to the shared wireless medium such that the performance requirements of the underlying applications are satisfied [42-45]. The major sources of energy waste in conventional MAC protocols are Packet collision, idle listening, overhearing, and control overhead [46].

Cluster-based routing protocols such as Low–energy adaptive clustering hierarchy (LEACH) [47], Proxy-based adaptive clustering hierarchy (PEACH) [48], Energy-driven adaptive clustering hierarchy (EDACH) [49] are known to minimize sensors energy consumption.

2.2.5. Application Level Energy Conservation

The application level energy conservation concerns issues such as data acquisition rate, data processing (distributed or central), data aggregation or reduction. The specific WSN applications such as target tracking, event detection, phenomena monitoring, actuation, and etc. determine which of these strategies can be optimal.

In WSN applications, data is acquired at a rate desired by the application process. However, different data acquisition methods can be employed. For instance, in target tracking application, all sensors can collect data about a moving target either at the same rate or different rates. That is sensors at close proximity to the target can capture data at faster rates than those at a distance from the target.

In WSNs, communication among the nodes is the major energy consuming process. A large percentage of the nodes' energy is spent on radio transmissions and receptions [50, 51]. Hence, processing incoming data locally as much as possible and transmitting only when incoming data shows significant variation can contribute to WSN efficiency. Also when a cluster-based topology is in use, data aggregation [52] at cluster heads can significantly enhance energy efficiency.

2.3. DATA STREAM MINING

Nowadays a growing number of applications generate streams of data characterized by massive volume and continuous fast arrival rates. Applications such as performance measurement in network monitoring and traffic management, call detail records in telecommunications, transactions in retail chains, ATM operations in banks, web logs on servers, and sensor networks generate data streams. Data stream mining gained in importance over recent years because it is indispensable for many real applications such as prediction and evolution of weather phenomena; security and anomaly detection in networks; evaluating satellite data; and mining health monitoring streams.

Data stream processing systems are interested in mining patterns, processing queries, and compute statistics on data streams in real-time. Stream mining algorithms must take account of the unique properties of stream data: infinite data, temporal ordering,

concept drifts and shifts, demand for scalability, etc. Due to the nature of data streams, the stream processing systems impose certain unique requirements; each record (stream element) is examined once or a small number of times at most (single pass), there is limited memory for storing summary, and per record processing time must be low (real-time). Therefore, the key issue in mining on data streams is that only one pass is allowed over the entire data. Moreover, there is a real-time constraint, i.e. the processing time is limited by the rate of arrival of instances in the data stream, and the memory and disk available to store any summary information may be bounded [11]. Data stream processing algorithms generally compute approximate answers with deterministic or probabilistic error bounds [53, 54]. There are many stream mining techniques and methods proposed within the technology and knowledge discovery community to overcome the challenges of storing and processing of fast and continuous streams of data [55-57]. Data-based techniques and task-based techniques are the two categories of data stream mining algorithms. Based on these two categories, a number of clustering, classification, and frequency counting and time series analysis have been developed [58]. Data-based solutions focus on stream synopses computation that enables efficient processing of the data stream by the existing mining methods to meet the requirements of data streams. Task-based solutions focus on developing methods to address the computational challenges of data stream processing [59, 60].

There are a number of synopsis data structures in the literature and in existing systems. Examples include uniform and biased random samples, various types of histograms, statistical summary information such as frequency moments, data structures resulting from lossy compression of the data set, etc. Often, synopsis data structures are used in a heuristic way, with no formal properties proved on their performance or accuracy, especially under the presence of updates to the data set [61]. A Variety of techniques can be used for synopsis construction in data streams including sampling, histograms, wavelets, sketches, and micro-cluster based summarization. A survey of these methods construction in data streams can be found in [62]. Task-based techniques include approximation algorithms, sliding windows, and algorithm output granularity [60].

Based on the two data stream mining techniques, data-based and task-based, a number of methods/algorithms have been proposed for extracting knowledge from streaming data. These mining algorithms summarize the whole or part of the incoming stream using data-based techniques such as sampling [63], load shedding [64], sketching [65], synopsis data structures [66], clustering [67], etc. to form the basis for data stream mining. Based on the data stream summaries a number of task-based stream analysis techniques have been employed including frequent pattern mining in data streams [68], multidimensional analysis of streaming data [69], classification analysis of data stream [70], stream clustering [67], stream outlier analysis, rare event detection [71], and so on. Among these, the most frequently applied techniques are described below.

2.3.1. Frequent pattern mining

Frequent pattern mining has become one of the most actively researched topics in data mining and knowledge discovery in databases. The starting point was market basket analysis and especially the task to mine transactional data, which describe the shopping behavior of customers of supermarkets, mail-order companies and online shops, for products that are frequently bought together. For this task, which became generally known as frequent item set mining, a large number of efficient algorithms were developed, which are based on sophisticated data structures and clever processing schemes. Among them, Apriori [72], Eclat [73], and FP-growth [74, 75] are most widely known. Extensions from item sets to item sequences are fairly straightforward, but open up exciting new application areas, like genome mining [76] and temporal pattern extraction from data describing, for instance, alarms occurring in telecommunication networks [77]. Recently, finding frequent patterns from data streams has become one of the important and challenging problems, since capturing the stream content memory efficiently with a single-pass and efficient mining have been major issues [78].

Jiawei et al. [74] developed an efficient Frequent-pattern tree (FP-tree) based mining method, FP-growth, for mining the complete set of frequent patterns by pattern fragment growth. They avoided the costly candidate generation-and-test

drawback of Apriori-like algorithms. The use of FP-growth is, however, restricted to static data sets due to the FP-tree requirements of two database scans and prior threshold knowledge. Leung and Khan [79] proposed a novel tree structure, called DSTree (Data Stream Tree) that efficiently captures important concepts from the data stream. Several frequent pattern mining algorithms in streaming data use the sliding windows approach [80]. Carson and Fan [81, 82] proposed mining algorithms that use the time-fading and landmark models to discover frequent patterns from streams of uncertain data. They designed a tree structure that captures and stores frequent patterns discovered from batches of transactions in dynamic streams for users interested in discovering frequent patterns from a variable-size time window. Several other stream frequent pattern mining algorithms have been proposed [83, 84].

2.3.2. Classification Mining

Classification is a data mining (machine learning) technique used to predict group membership for data instances or is the process of automatically creating a model of classes from a set of records that contain *class labels*. The Classification mining function analyzes records that are already known to belong to a certain class, and creates a profile for a member of that class from the common characteristics of the records. A data mining application tool can then be used to apply this model to new records, that is, records that have not yet been classified. Popular classification techniques include decision trees and neural networks.

In recent years, there have been progressively several decision tree algorithms for data stream classification emerged, such as Very Fast Decision Tree (VFDT) [85] and Concept Adapting Very Fast Decision Tree (CVFDT) [86]. VFDT implements a decision-tree learning system based on the Hoeffding tree *algorithm.* CVFDT is a decision-tree induction system capable of learning accurate models from high speed, concept-drifting data streams. CVFDT is an efficient algorithm for mining decision trees from continuously-changing data streams, based on the ultra-fast VFDT decision tree learner. CVFDT stays current while making the most of old data by growing an alternative subtree whenever an old one becomes questionable, and replacing the old with the new when the new becomes more accurate. Feixiong and Quan [87] have

extended the VFDT system to EVFDT (Efficient-VFDT) in two directions: (i) they present Uneven Interval Numerical Pruning (UINP) approach for efficiently processing numerical attributes. (ii) they use naive Bayes classifiers associated with the node to process the samples to detect the outlying samples and reduce the scale of the trees. From the experimental comparison, the two techniques significantly improve the efficiency and the accuracy of decision tree construction on streaming data.

2.3.3. Outlier Detection

Outlier detection is a branch of data mining concerned with the discovery of data that deviates significantly from other data. An outlier is an observation in a data set which appears to be inconsistent with the reminder of that set of data [88]. Outliers are often considered as an error or noise; however, they may carry important information. Their detections prior to data modeling and analysis is usually a key to coherent analysis and unbiased results. Outlier detection has found application in credit card fraud detection [89], clinical trials [90], data cleansing [91], network intrusion [92], severe weather prediction [93], geographical information systems [94], and others. Several studies have been conducted on outlier detection for large datasets. The early work in outlier detection employs statistical methods on the database assuming a priori knowledge of distribution [95-97]. Clustering algorithms have also been used in outlier detection where objects that are not located within clusters of a dataset are considered outliers [98-101]. Recently, outlier detections for data streams are also studied using techniques as sliding windows [102-104], auto-regressive technique for time series data [95, 105], outlier detection for sensor networks' multiple homogeneous data stream [104, 106], and clustering outlier data stream techniques [107, 108].

2.3.4. Stream Clustering

Clustering in the data stream domain is partitioning of large volumes of data arriving in a stream. The objective is to maintain a consistently good clustering of the sequence observed so far, using a small amount of memory and time. Due to the relevance of new classes of applications involving massive data sets, clustering in the data stream

model has become important. In recent years, a few one-pass clustering algorithms have been developed for the data stream problem [109, 110]. For instance, the algorithm in [110] extends the k-means algorithm to stream based continuous clustering, which maintain a number of cluster centers that change or merge as necessary throughout the execution of the algorithm. Density based clustering algorithms specially designed for data streams have also emerged [111-114]. On the other hand, some algorithms using micro-cluster for saving summary information about the clusters that are not density based are designed for data streams [115, 116]. As the understanding of streaming data mature, more and more commercial stream clustering algorithms are emerging. Among the stream clustering algorithms, STREAM [118], BIRCH [118], and COBWEB [119] are well known.

2.4. DISTRIBUTED TECHNIQUES FOR WSN DATA STREAM MINING

The main focus of a vast majority of research in the WSN field is on energy efficiency and network lifetime maximization. The dominance of communication power consumption over computation power consumption in WSNs has motivated research into a communication-computation tradeoff strategy for energy efficiencies and network lifetime maximization. The high data rate of sensor nodes in these networks has further raised the issue of data processing model efficiency. Distributed sensor data stream mining systems have emerged as a result, to address both the challenges of energy efficiency or network lifetime maximization and high data rates. This section briefly reviews some of the existing distributed data stream mining methods with particular emphasis on distributed sensor data stream clustering WSN systems.

To design effective distributed techniques for WSN data stream mining applications under their inherent constraints, the above general mining techniques have to be crafted to suit the distributed nature of WSNs and satisfy online stream mining goals. Due to the high computational burden of analyzing such streams, distributed stream mining systems have been recently developed [120]. It has been shown that distributed stream mining systems transcend the scalability, reliability, and performance objectives of large-scale, real-time stream mining systems [121-123].

Clustering is probably the most frequently used data mining algorithm, used as exploratory data analysis technique [124]. The general goal of a clustering technique is to decompose or partition data sets into groups such that both intra-group similarity and inter-group dissimilarity are maximized [125]. For the WSN environment to achieve significant energy conservation, clustering has to be performed in distributed fashion within the network due to the inherent constraints. There are several recent research works on distributed clustering.

When data is being produced at multiple locations, as in a WSN, two major clustering frameworks are apparent in the wider literature. The first framework consists of a process that gathers data to a central location and analyzes the stream at the central location. The second framework consists of two level-processing; level one clusters data at the individual sources and level two compiles the results at a central location and defines the final clusters based on the clusters transmitted by the individual sources. The former framework is obviously resource inefficient and inapplicable to WSN systems. The latter framework has attracted several research works. The cluster ensemble approach [126], for instance, follows this framework.

Distributed data mining appears to have the necessary features to apply clustering to streaming data produced on sensor networks [127]. Although few works were directly targeted at data clustering on sensor networks, some distributed techniques are obviously relevant.

Continuous clustering algorithms over distributed data streams have recently attracted the attention of the clustering research community. In [128] the authors present a distributed majority vote algorithm, which can be seen as a primitive to monitor a k-means clustering over peer-to-peer networks. The k-means monitoring algorithm has two major parts: monitoring the data distribution in order to trigger a new run of k-means algorithm and computing the centroids actually using the k-means algorithm. The monitoring part is carried out by an exact local algorithm, while the centroid computation is carried out by a centralization approach. The local algorithm raises an alert if the centroids need to be updated. At this point data is centralized, a new run of k-means is executed, and the new centroids are shipped back to all peers.

A different strategy to achieve the same goal, with local and global computations, in order to balance communications costs has been proposed in [129]. They considered techniques which give an approximation for the radius and diameter of clusters with guaranteed cost of two times the cost of the optimal clustering based on the furthest point algorithm [130].

Kargupta et al. presented a collective principal component analysis (PCA), and its application to distributed cluster analysis [131]. In this algorithm, each node performs PCA, projecting the local data along the principal components, and applies a known clustering algorithm on this projection. Then, each node sends a small set of representative data points to the central site, which performs PCA on this data, computing global principal components. Each site projects its data along the global principal components, which were sent back by the central node to the rest of the network, and applies its clustering algorithm. However, these techniques can easily overload the system when the sensors are required to react to a query.

Klusch et al. proposed a kernel density based clustering method over homogeneous distributed data [125], which, in fact, does not find a single clustering definition for all data sets. It defines local clustering for each node, based on a global kernel density function, approximated at each node using sampling from signal processing theory. These techniques present a good feature as they perform only two rounds of data transmission through the network. Other approaches using the K-Means algorithm have been developed for peer-to-peer environments and sensor network settings [132].

Considering the lack of resources usually encountered on sensor networks, Gaber & Yu proposed Resource-Aware Clustering [133] as a stream clustering algorithm for clustering that can adapt to the changing availability of different resources. The system is integrated in a generic framework that enables resource-awareness in streaming computation, monitoring main resources like memory, battery and CPU usage, in order to achieve scalability in distributed sensor networks, by adapting the parameters of the algorithm. Data arrival rate, sampling and number of clusters are examples of parameters that are controlled by this monitoring process.

Previous works concentrate on clustering of data at local sites and compiling the local results at the central cite to define global clustering. Some of the previous works also considered relaying the global clusters to the local sites so that the local sites can fine tune their local clustering to achieve better overall global cluster models. However, none of them considered detecting and suppressing the local site computations whenever the data acquired at these sites show no significant changes from their previous acquisition, which is a common occurrence in dynamic systems. This strategy reduces the global clustering computation at the central site whenever there are no significant changes observed at the local sites. Further, few of the previous works consider producing mining results in an on-line fashion with the exception of [134, 135].

Qi et al. [134] proposed a suite of communication efficient algorithms for computing approximate k-median clustering over distributed data streams under different topology settings. Their algorithm basically considers a tree structure whereby each node computes local summaries and refines the summaries aggregating data from their child nodes along the path to the root. Though this guarantees low error bounds in final summaries while significantly reducing communication, it involves multiple merge and compression computations to achieve the final summary and hence it is not able to handle online clustering of fast streams. This algorithm does not feedback the global summaries into the network and hence local summaries have to be computed continuously. Therefore, it suffers from poor computation scalability.

Maria and Iordanis [135] proposed an online data clustering method suitable for distributed streaming data processing and for capturing their dynamically changing characteristics using belief propagation techniques. They considered a set of distributed nodes that communicate directly with a central location. At each time slot the node level identifies a set of representative data items (exemplars) based on certain similarity matrices and sends the exemplars to the central location. The central location computes global exemplars and feeds back to the nodes with appropriately modified weights which reflect their importance in global clustering. Their algorithm does not apply for multi-hop networks where some nodes cannot directly reach the central location. Besides, the appropriate cluster weights feedback is not possible where there is no prior knowledge of desired global exemplars. Their work, however, is

focused on capturing changing characteristics and does not consider resource constraints and WSN characteristics.

Distributed clustering of streaming data under the framework of two-level processing -- where level-one clusters data at the individual sources and level-two compile the results at a central location and defines the final clusters based on the clusters transmitted by the individual sources -- has been targeted by researchers to cope with high-speed production of data streams. Meanwhile, a gap has been left in adapting the distributed stream clustering for the on-line mining framework in the context of WSNs. We propose a distributed incremental stream mining framework for WSNs consisting of a multilevel processing architecture.

2.5. CHAPTER SUMMARY

In this chapter we have first presented the architecture of WSNs. The WSN functional components have hardware platforms, WSN operating systems, wireless protocol stacks and applications are described. The energy conservation strategies that can be useful at those components are also described. Following the WSN description, the data stream mining techniques, specifically those relevant to the WSN framework are reviewed. We have analyzed the distributed data stream mining framework based on the literature in the context of WSN systems and identified a gap in this research area.

Chapter 3

3. RESEARCH TOOLS AND MATERIALS

This chapter introduces general tools and materials that are used in this research to model and simulate the concept of distributed incremental data stream mining WSNs and implementation of a small scale prototype system. These tools and materials enable building models representing real life scenarios and simulate various events to derive meaning and draw conclusions from data obtained. Analysis of the details from the simulation provides concise descriptions and guidelines for optimal real system counterpart development. The main tools and materials used in this research are:

1. TrueTime simulator.
2. TI's CC2530ZNP-Mini kit.
3. Sensirion's SHT1x Humidity/Temperature sensors.
4. Sparkfun's SEN-08942 Weather meter.

3.1. TRUETIME SIMULATOR

TrueTime is a MATLAB/Simulink based simulator [136] for real-time networked and embedded control systems. It facilitates co-simulation of the temporal behavior of a multitasking real-time kernel, network transmissions, and continuous model dynamics. The tasks are processes that are modeled as ordinary continuous-time Simulink blocks.

TrueTime also makes it possible to simulate models of standard MAC layer network protocols, and their influence on the network. Further details can be found in the TrueTime kernel Reference Manual [137].

TrueTime has been studied thoroughly by researchers in the embedded networked control systems society. Its reliability has been validated in different studies, including simulation of computer nodes and communication networks interacting with continuous time dynamics of the real world [138], time-triggered and event-based networked control and AODV routing in wireless ad-hoc networks [139], and WirelessHART communication system clock drift, delay, and packet loss [140].

TrueTime is MATLAB-based and requires MATLAB 7.0(R14) with Simulink 6.0 (R14) or later. TrueTime has been tested under Linux, Windows, and Mac operating systems. The TrueTime simulator, as shown in Figure 3.1, contains a block library with, TrueTime Kernel block, TrueTime Network block, TrueTime wireless network block, TrueTime ultrasound network block, TrueTime Battery block, and TrueTime Send and Receive standalone blocks. The blocks are variable-step, discrete, MATLAB S-functions written in C++ [141]. The user writes code functions to configure and initialize these blocks for a specific simulation. The code functions for tasks and the initialization commands may be written either as C++ functions or as MATLAB M-files. During the simulation, User defined tasks and interrupt handlers representing, e.g., I/O tasks, process algorithms, and network interfaces are executed on the kernel block according to a user defined scheduling policy. The TrueTime blocks are event-driven where the executions are determined by events (both internal and external). Internal events correspond to events such as scheduled timer interrupts, message transmission completione, etc. External events correspond to arrivals of over the air messages, sensor readings, etc.

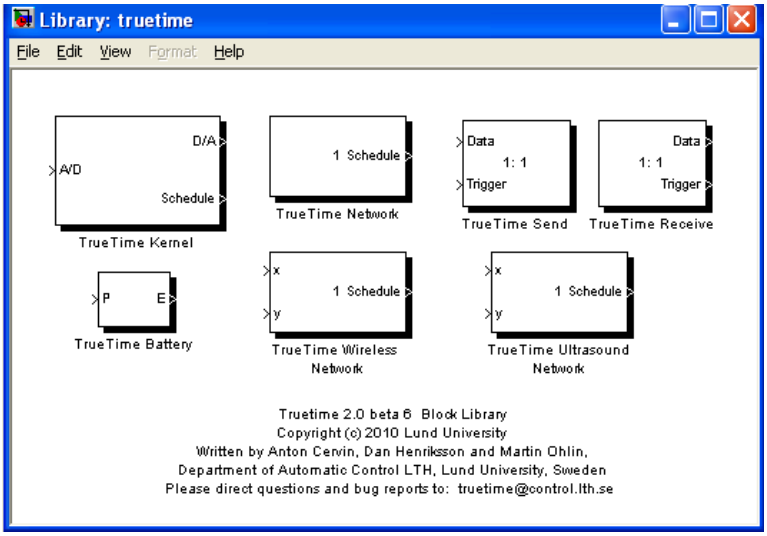


Figure 3.1: TrueTime Block Library.

3.1.1. The TrueTime Kernel

TrueTime implements a complete real-time kernel with a ready queue for tasks ready to execute, a time queue for tasks waiting to be released and waiting queues for monitors and events. Queues are manipulated by the kernel or by calls to kernel primitives. The simulated kernel is ideal such that no interrupt latency and no execution time associated with real-time primitives, however possible to specify a constant context switch overheads. TrueTime utilizes the Simulink zero-crossing function to enable event-based simulation.

The TrueTime kernel block S-function simulates a computer or controller with a simple flexible real-time kernel, including A/D and D/A converters, network interface, and external interrupt ports. The kernel abstracts several data structures that are commonly found in a real-time kernel such as ready queue, time queue, tasks records, interrupt handlers, monitors and timers created for simulation. The execution of tasks and interrupt handlers is defined by code functions, written in C++ or MATLAB code. Process algorithms may be defined graphically using ordinary discrete Simulink block diagrams.

The kernel is configured through the block mask dialog (Figure 3.2) with parameters; Init function, Init function argument, Battery, Clock drift, and Clock offset.

A single run-time parameter to configure the kernel on the fly is also available. At the moment dynamic CPU scaling and energy consumption can be set through the run-time configuration command.

The init function parameter defines the name of the initialization script which must be on the same path as the simulation model file. The Init function argument is an optional argument to the initialization script. The Battery parameter sets whether the kernel should depend on a power source. The Clock drift defines the desired time drift between the local time and the actual simulation time. The Clock offset sets a constant time offset from the nominal time.

In this research, the kernel block simulates a sensor node platform i.e. the controller that hosts the sensor interface library, wireless protocol stack and data stream mining algorithm.

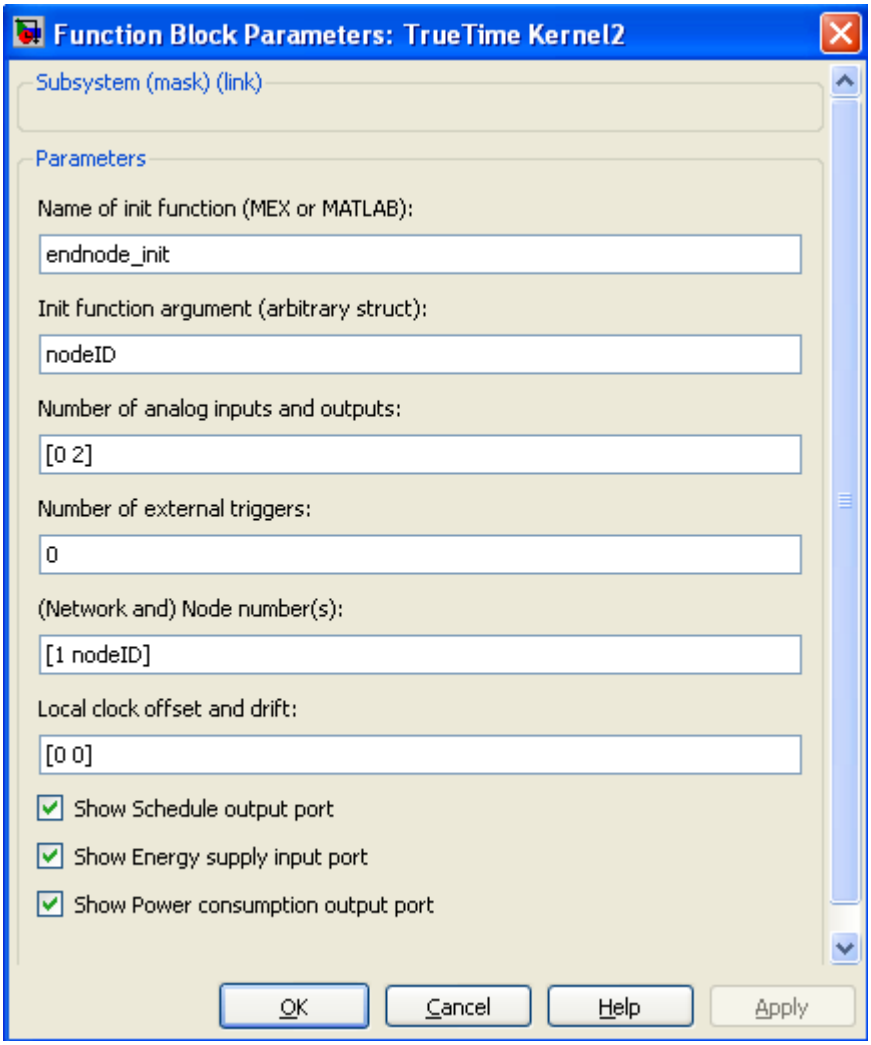


Figure 3.2: The TrueTime Block Mask Dialog.

3.1.1.1. Tasks

In TrueTime, tasks are used to model the execution of user codes. The release of task instances (jobs) may be periodic or aperiodic. For periodic tasks, the jobs are created by an internal periodic timer. For aperiodic tasks, the jobs (e.g. to respond to set interrupts) must be created by the user. When there are multiple jobs of the same task, pending jobs are queued. Each job has an execution-time budget. Dynamic task attributes such as release time, absolute deadline, and execution time are updated by the kernel as simulation progresses. However, static task attributes such as period, priority, and relative deadline are kept constant unless explicitly changed by the user. Communications between tasks is supported by mailboxes. A finite ring buffer is used to store incoming messages.

3.1.1.2. Code Function

The user task code is represented by a *code function* in the format shown in Equation 3.1. Discrete Simulink blocks may be called from within the code functions. Block states are stored in the kernel between calls.

$$[exectime, data] = function\ mycode(segment, data) \quad (3.1)$$

where *data* is an input/output argument representing local memory of the task. *Segment* is an input argument representing the program counter, and *exectime* is an output argument representing the execution time of the current code segment.

3.1.1.3. Code Segments

A code segment models a number of statements that are executed sequentially as shown in Figure 3.3. Multiple code segments are required to simulate input-output delays, self-suspensions, waiting for events or monitors, and loops or branches.

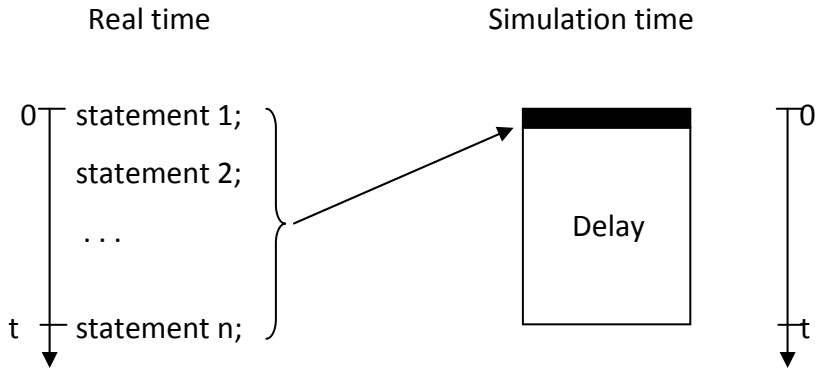


Figure 3.3: Code Segment.

The execution time t must be supplied by the user and it may be constant, random or data-dependent. A return value of -1 for exectime means that the job has finished. All statements in a segment are executed sequentially; non-preemptively, in zero

simulation time, and no local variables are saved between segments. Only the delay can be preempted by other tasks.

3.1.1.4. Configuring Simulation

Each kernel block is initialized in a script (block parameter) as in listing 3.1 below. The scheduling policy of the kernel is defined by a priority function, which is a function of task attributes. Pre-defined priority functions exist for fixed-priority, rate-monotonic priority, deadline-first priority, and earliest-deadline-first scheduling.

```
nbrInputs = 3;
nbrOutputs = 3;
ttInitKernel(nbrInputs, nbrOutputs, prioFP);
periods = [0.01 0.02 0.04];
code = myCtrl;

for k = 1:3
     data.u = 0;
     taskname = [Task  num2str(k)];
     offset = 0; % Release task at time 0
     period = periods(k);
     prio = k;
     ttCreatePeriodicTask(taskname,offset,period,prio,
code,data);
end
```

Listing 3.1: The Kernel Block initialization script.

3.1.1.4. Scheduling Hooks

Scheduling hook is code that is executed at different stages during the execution of a task that facilitates implementation of arbitrary scheduling policies, such as server-based scheduling. TrueTime supports the following six scheduling hooks; Arrival hook, Release hook, Start hook, Suspend hook, Resume hook, and Finish hook. The Arrival hook is executed when a job is created. Release hook is executed when the job is first inserted in the ready queue. Start hook is executed when the job executes its first segment. Suspend hook is executed when the job is pre-empted, blocked or voluntarily

goes to sleep. The resume hook is executed when the job resumes execution. The Finish hook is executed after the last code segment.

3.1.1.5. Data Logging

A number of variables may be logged by the kernel as the simulation progresses and are written to the MATLAB workspace when the simulation terminates. TrueTime provides automatic logging for response time, release latency, sampling latency, task execution time, and context switch instances. User variables may be logged as required in the given scope.

3.1.1.6. Monitors

Monitors are used to model mutual exclusion between tasks that share common data. Tasks waiting for monitor access are arranged according to their respective static or dynamic priorities. The implementation supports standard priority inheritance to avoid priority inversion.

3.1.2. The TrueTime Wireless Network

The TrueTime Wireless network block simulates medium access and packet transmission in a wireless network. The network blocks dispatch messages between kernel blocks according to a preferred model of a wireless network. The network block contains a discrete-event simulator that reads incoming messages, handles the medium access and resolves collisions, simulates the actual data transmission, and writes outgoing messages.

It takes into account the path-loss of the radio signal through x and y inputs that specify the true location of the nodes. The network protocols supported are limited to IEEE 802.15.4 (ZigBee) and IEEE 802.11b/g (WLAN) at the moment. The radio model in use includes support for: Ad-hoc wireless networks, isotropic antenna,

inability to send and receive messages at the same time, path loss of radio signals modeled as $\frac{1}{d_a}$ where d is the distance in meters and a is a parameter chosen to model the environment, and interference from other terminals.

The wireless network block is configured through the block mask dialog (Figure 3.4) with parameters; Network type, Network number, Number of nodes, Data rate, Minimum frame size, Transmit power, Receiver signal threshold, Path-loss exponent, ACK timeout, Retry limit, and Error coding threshold.

The Network type parameter determines the MAC protocol to be used (either IEEE 802.15.4 or IEEE 802.11b/g). The network number parameter specifies the number of network blocks in use. Number of nodes specifies the number of nodes connected to the network block. Data rate determines the speed of the network in bits per second (bits/s). Minimum frame size determines the minimum message size in bits including protocol overhead. A message or a frame shorter than minimum frame size will be padded to give minimum length. Transmit power determines the strength of the radio signal and hence its reach. Receiver signal threshold determines the received signal energy threshold above which the medium is classed as busy. Path-loss exponent models the radio signal path loss of the environment. ACK timeout is the time the sending node will wait for ACK (acknowledgment) before retransmitting. Retry limit is the maximum number of times a node will try to retransmit a message before giving up. Error encoding threshold defines the percentage of block errors based on the signal-to-noise ratio in a message that the coding can handle.

3.1.3. The TrueTime Battery

The battery block enables simulation of battery-powered devices. The initial power of a node is set using the battery configuration mask. The battery constitutes a simple integrator model, so that it can be both charged and discharged. The power drains such as kernel computation, radio transmissions, sensors and actuators must be connected to the battery input to simulate the node's power consumption. If the kernel is configured to use battery and the energy input to the kernel is zero, it will not

execute any code. The dynamic voltage scaling scheme along with the battery block allows simulations with changing CPU speed and proportional energy consumption scenarios.

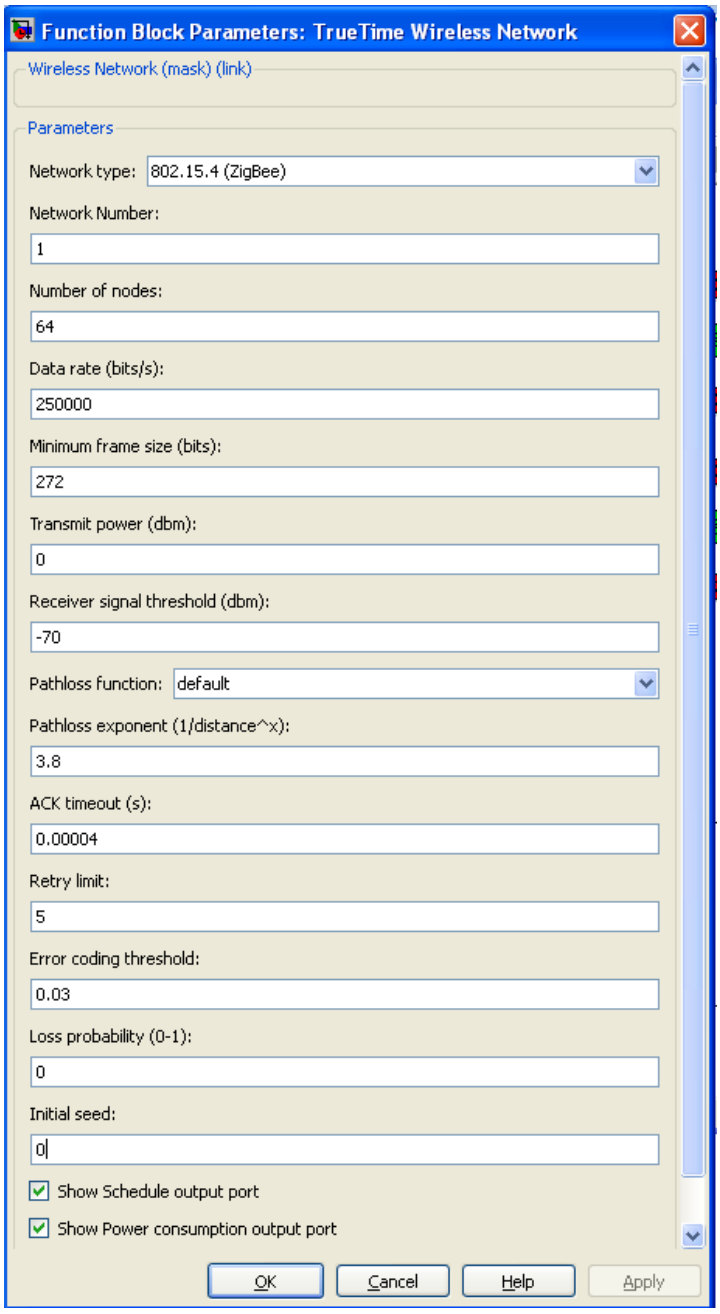


Figure 3.4: The TrueTime Wireless Network Block mask dialog.

3.1.4. Limitations of TrueTime

TrueTime cannot express tasks and interrupt handlers directly using production code. The code is modeled using TrueTime MATLAB code or TrueTime C code and there is no automatic translation. Further code execution times or distributions are assumed to be known. There is no built-in support for network and transport layer protocols such as TCP and AODV, however, these two are implemented as an example and included in the package. The code segments in the code function are non pre-emptive, as MATLAB does not allow functions to be pre-empted/resumed.

3.2. TI's CC2530 ZNP-MINI KIT

The CC2530ZNP-Mini kit is Texas Instruments (TI) ZigBee network development kit. The CC2530ZNP-Mini kit is the successor to the EZ430-RF2480 and uses the ZigBee network processor (ZNP) firmware on the CC2530 system-on-chip (SoC). The CC2530ZNP allows separating the ZigBee stack from the application processor. The ZigBee Network Processor development kit (Figure 3.5) is a typical introduction to ZigBee WSNs. The hardware consists of a CC2530 ZigBee device programmed with ZigBee software and an MSP430F2274 microcontroller that controls the ZigBee device. The kit enables existing applications to add a serial interface to a ZigBee processor that takes care of all protocol handling for ZigBee communication. The kit sensor boards include an accelerometer, temperature sensor, and light sensor that can be used in conjunction with LED lights and push buttons to develop simple demo applications. The Kit board is shown in Figure 3.6 below.

3.2.1. CC2530

TI's CC2530 is a true system-on-chip solution tailored for IEEE 802.14.5, ZigBee applications. The CC2530 combines a fully integrated, high-performance RF transceiver with an 8085 MCU, 8 KB of RAM, 32/64/128/256 KB of Flash memory, and powerful supporting features and peripherals. Combined with TI's low power microcontroller, MSP430F2274, CC2530ZNP provides a very easy way of deploying and testing low

power sensor networks. The block diagram of the CC2530 is shown in Figure 3.7 below. Refer to datasheet [143] for more details of the CC2530. The kit provides some typical features for low power WSN based on the ZigBee standard;

- 2.4-GHz IEEE 802.15.4 Compliant RF transceiver.
- Excellent Receiver Sensitivity and Robust to Interference.
- Programmable Output Power up to 4.5 dBm
- High-performance and Low-Power 8085 Microcontroller core with Code Prefetch.
- Low Power
    - Active mode RX (CPU Idle): 24 mA.
    - Active mode TX at 1 dBm(CPU Idle): 29 mA.
    - Power mode 1 (4 μs wake-up): 0.2 mA.
    - Power mode 2 (Sleep Timer Running): 1 μA.
    - Power mode 3 (External Interrupt): 0.4 μA.
    - Wide Supply-Voltage Range (2 V – 3.6 V).
- ISP communication to host controller

Figure 3.5: CC2530ZNP-Mini Kit.

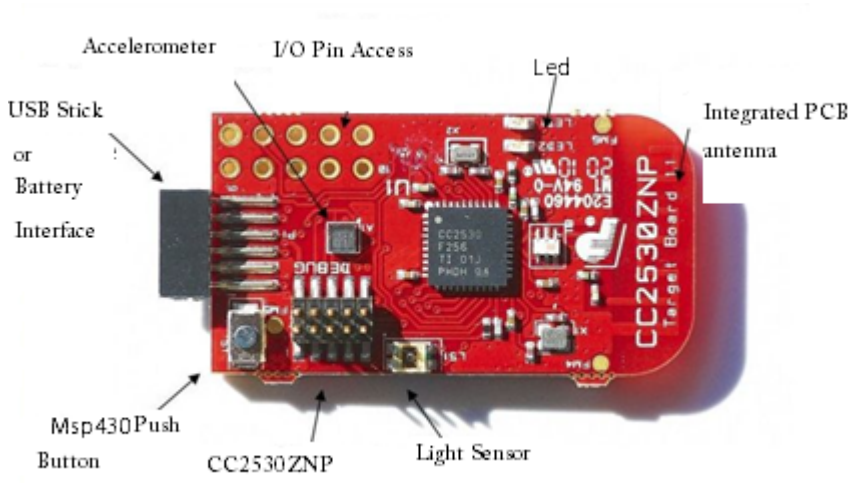


Figure 3.6: CC2530ZNP board.

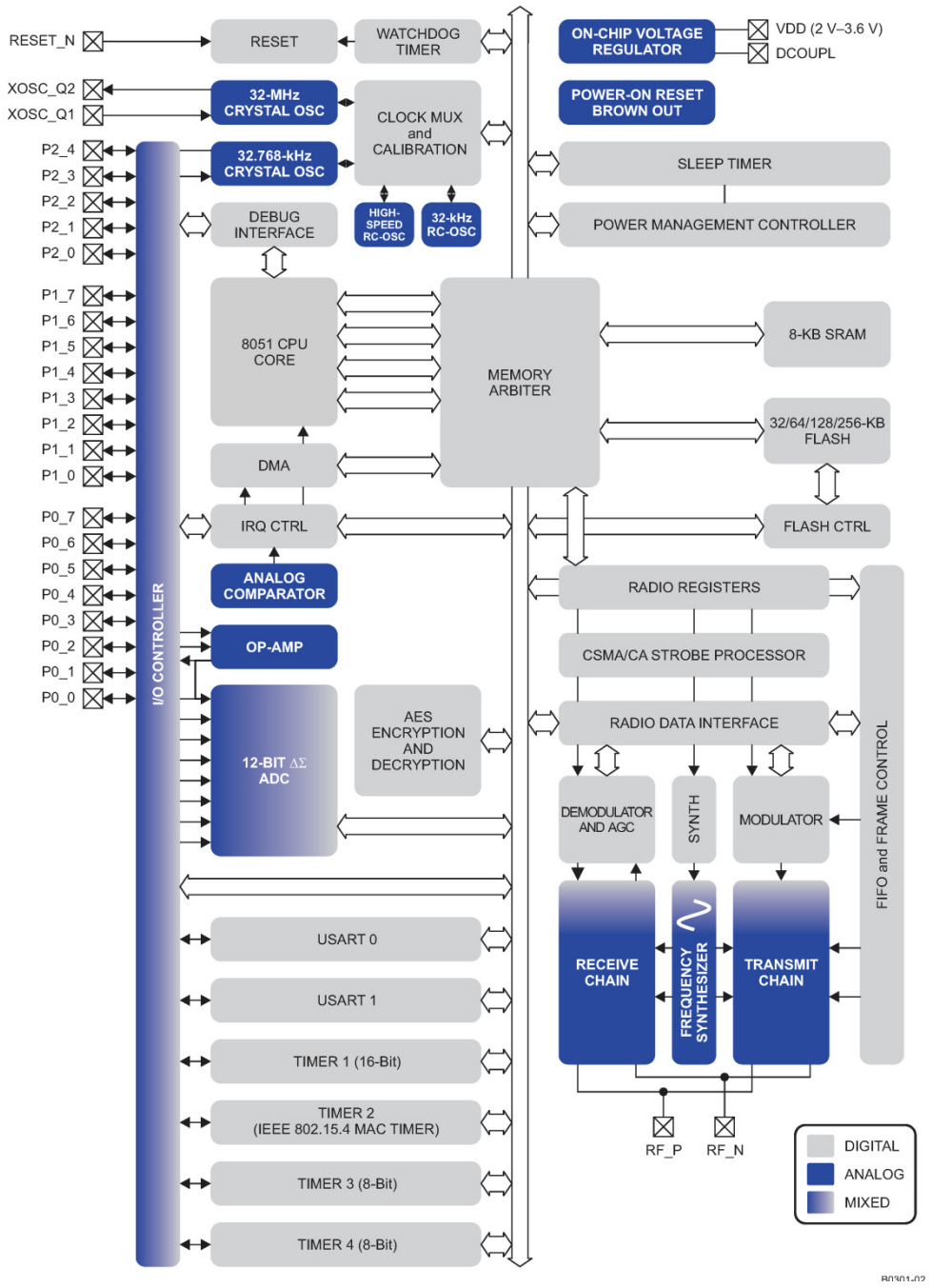


Figure 3.7: CC2530 ZNP Block Diagram.

3.2.2. MSP430F2274

The Texas Instruments MSP430F2274 is an ultra-low-power mixed signal microcontroller. It features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The MSP430F2274 peripherals include two built-in 16-bit timers, a universal serial communication interface, 10-bit A/D converters with integrated reference and data transfer controller (DTC), two general-purpose operational amplifiers, and 32 I/O pins. The architecture combined with five low-power modes is optimized to achieve extended battery life in portable applications. Refer to the datasheet [142] for more information. The functional block diagram of the device is shown in Figure 3.8.

The clock system of the MSP430F2274 is supported by the basic clock module that includes support for a 32768-Hz watch crystal oscillator, an internal very-low-power low-frequency oscillator, an internal digitally-controlled oscillator (DCO), and a high-frequency crystal oscillator. The basic clock module is designed to meet the requirements of both low system cost and low power consumption. The internal DCO provides a fast turn-on clock source and stabilizes in less than 1 μs. The basic clock model provides the following clock signals:

- Auxiliary clock (ACKL), sourced from a 32768-Hz crystal, a high-frequency crystal, or the internal very-low-power LF oscillator
- Main clock (MCLK), the system clock used by the CPU
- Sub-main clock (SMCLK), the sub-system clock used by the peripheral modules

The MSP430F2274 microcontroller has an active mode and five software-selectable low-power modes of operation. An interrupt event can wake up the device from any of the five low-power modes, service the request, and restore back to the low-power mode on return from the interrupt programme.

The operation modes of the MSP430F2274 are:

- Active mode (AM)
    - All clocks are active.
- Low-power mode 0 (LPM0)

    - CPU is disabled.
    - ACLK and SMCLK remain active. MCLK is disabled.
- Low-power mode 1 (LPM1)
    - CPU is disabled ACLK and SMCLK remain active. MCLK is disabled.
    - DCO dc-generator is disabled if DCO not used in active mode.
- Low-power mode 2 (LPM2)
    - CPU disabled.
    - MSCL and SMCLK are disabled.
    - DCO dc-generator remains enabled.
    - ACLK remains active.
- Low-power mode 3 (LPM3)
    - CPU is disabled.
    - MCLK and SMCLK are disabled.
    - DCO dc-generator is disabled.
    - ACLK remains active.
- Low-power mode 4 (LPM4)
    - CPU is disabled.
    - ACLK is disabled.
    - MCLK and SMCLK are disabled.
    - DCO dc-generator is disabled.
    - Crystal oscillator is stopped.

The MSP430F2274 has four 8-bit I/O ports implemented—ports P1, P2, P3, and P4. Only three I/O pins are implemented from port P2, therefore bits [5:1] of all port P2 registers read as 0 and write data is ignored.

- All individual I/O bits are independently programmable.
- Any combination of input, output, and interrupt condition is possible.
- Edge-selectable interrupt input capacity for all eight bits of port P1 and P2.
- Read/write access to port-control registers is supported by all instructions.
- Each I/O has an individually programmable pull-up/pull-down resistor.

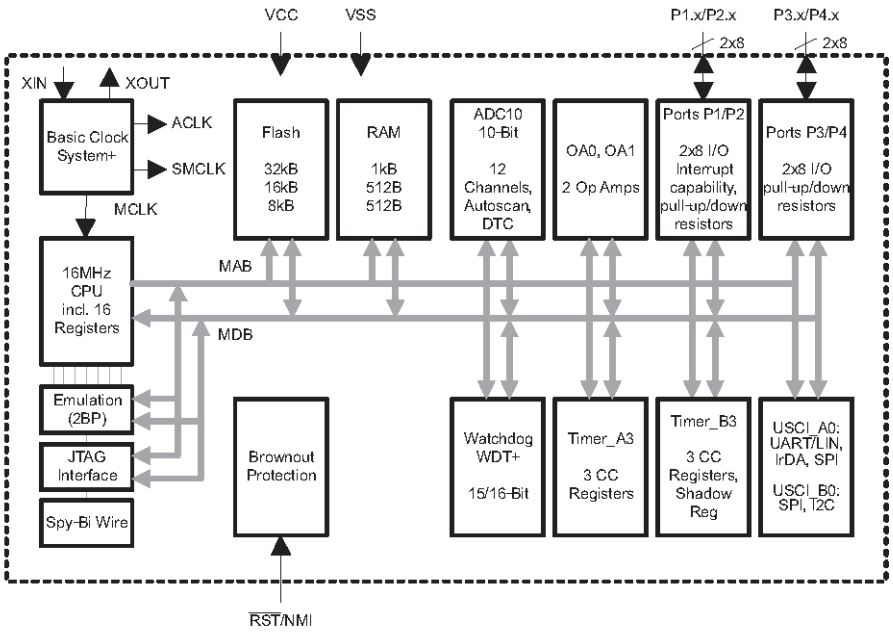


Figure 3.8: MSP430F2274 Functional Block Diagram.

3.3. SHT15 – DIGITAL HUMIDITY SENSOR (RH&T)

SHT15 digital humidity and temperature sensor integrates sensor elements plus signal processing on a tiny footprint (Figure 3.9) and provides a fully calibrated digital output. A capacitive sensor element is used for measuring relative humidity while temperature is measured by a band-gap sensor. Both sensors are seamlessly coupled to a 14-bit analog to digital converter (ADC) and a serial interface circuit. It is calibrated with its own calibration coefficients saved on the sensor's own EEPROM. The two-wire serial interface and internal voltage regulation allows for easy and fast system integration.

The SHT15 features;

1. Energy consumption: 800uW (at 12-bit, 3V, 1 measurement/s)
2. RH operating range: 0 -100 % RH
3. T operating range: -40 - +125 ℃ (-40 - +257 ℉)
4. RH response time: 8 sec (τ 63%)
5. Output: digital (2-wire interface)

where τ is the time for reaching 63% of a step function, valid at 25℃ and 1m/s airflow.

Relative humidity from 0% to 100% can be measured in typical steps of 2 % RH. Temperature sensor has a range of -40 degree Celsius to 123.8 degree Celsius with +-0.03 degree Celsius resolution.

The relative humidity (RH) sensor is non-linear. For compensating the non-linearity of the humidity sensor and obtaining the full accuracy of the sensor it is recommended to convert the humidity readout (SO_{RH}) with the following formula (3.2) with coefficients given in Table 3.1.

$$RH_{linear} = C_1 + C_2 * SO_{RH} + C_3 * {SO_{RH}}^2 \text{ (\%RH)} \quad (3.2)$$

Table 3.1: Humidity conversion coefficients.

SO_{RH}	C_1	C_2	C_3
12 bit	**-2.0468**	**0.0367**	**-1.5955E-6**
8 bit	**-2.0468**	**0.5872**	**-4.0845E-4**

The band-gap proportional to the absolute temperature is very linear by design. Equation 3.3 should be used to convert digital readout (SO_T) to temperature value, with coefficients given in Table 3.2.

$$T = d_1 + d_2 * SO_T \quad (3.3)$$

Table 3.2: Temperature conversion coefficients.

VDD	d_1 (°C)	d_1 (°F)
5V	-40.1	-40.2
4V	-39.8	-39.6
3.5V	-39.7	-39.5
3V	-39.6	-39.3
2.5V	-39.4	-38.9

SO_T	d_2 (°C)	d_2 (°F)
14 bit	0.01	0.018
12 bit	0.04	0.072

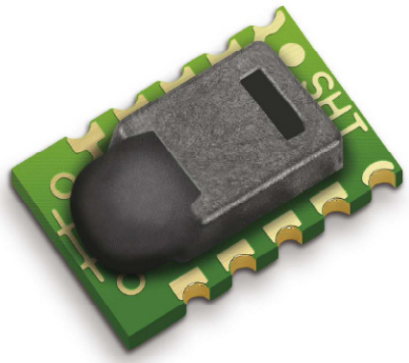


Figure 3.9: SHT15 Digital Humidity Sensor.

3.4. SEN-08942 WEATHER METER

SEN-08942 is Sparkfun Electronics' weather station that includes an anemometer, wind vane, and rain gauge (Figure 3.10). The sensors contain no active electronics, instead using sealed magnetic reed switches and magnets to take measurements. A voltage must be supplied to each instrument to produce an output. The anemometer uses a reed switch, so simple frequency detection can be used to measure wind speed. The wind vane uses a potentiometer to detect wind direction. The rain gauge acts as a switch that closes at measured increments.

The rain gauge is a self-emptying tipping bucket type. Each 0.011'' (0.2794mm) of rain causes one momentarily contact closure that can be recorded with a digital counter or microcontroller interrupt input.

The cup-type anemometer measures wind speed by closing a contact as a magnet moves past a switch. A wind speed of 1.492 MPH (2.4 km/h) causes the switch to close once per second.

The wind vane has eight switches, each connected to a different resistor. The vane's magnet may close two switches at once, allowing up to 16 different positions to be indicated. An external resistor can be used to form a voltage divider, producing a voltage output that can be measured with an analog to digital converter.

Figure 3.10: SEN-08942 Weather meter.

3.5. CONCLUSION

TrueTime is capable of investigating behavior of time or event-triggered processes (such as control loops) subject to sampling jitter, input-output latency, and lost samples caused by real-time scheduling and networking effects. It is also capable of investigating the performance of various scheduling methods, and wired or wireless MAC protocols. It can further simulate scenarios involving battery-powered and mobile nodes communicating using wireless ad hoc networks.

The CC2530ZNP-Mini kit is a perfect tool to add low-power wireless capability to an existing system with minimum porting as it provides separate application processor and ZigBee communication protocol processor.

The SHT15 Humidity and Temperature Sensor provides stable and high resolution digital output, which is ideal for applications that require long term untethered operation.

SEN-08942 weather meter is simple to implement, cost-effective, and low-power, well suited for energy constrained WSNs.

Chapter 4

4. DISTRIBUTED INCREMENTAL DATA STREAM MINING WIRELESS SENSOR NETWORK FRAMEWORK

4.1. INTRODUCTION

This chapter describes the theory of distributed incremental data stream mining WSNs based on the hybrid fuzzy clustering technique. The proposed framework enables mining of continuously streaming WSN data on the fly and in-network with limited resource requirements, thus expands the scope of applications for WSNs. The basic concept is to develop a distributed sensor data stream mining algorithm that minimizes inter-node communications, maximizes local computation and energy efficiency without compromising practical application requirements and quality of service (QoS).

WSNs consist of spatially distributed autonomous sensor nodes equipped to sense specific information and hence can be considered as distributed data sources (database). In several WSN applications, physical variables such as temperature, relative humidity, and light are generated in continuous streams. In such applications the WSN can be modeled as distributed data stream base and different distributed data stream management techniques can be utilized for analysis of the WSN.

In several WSN applications, physical variables such as temperature, relative humidity, and light are monitored continuously along the network operation. WSNs usually generate data continuously in an online fashion as time progresses. Thus, data arrival to the sink is more or less continuous and unordered. Data with such features

are commonly referred to as data streams [57], which are also known as sensor streams for data streams generated by sensor networks [144].

WSN nodes, besides being data stream sources, are also capable of limited processing, storing, and transmitting their data short distances wirelessly. This work leverages these limited capabilities of sensor nodes and their distributed nature to implement a distributed sensor stream mining system. The framework aims to achieve energy-efficiency, communication-efficiency, and computation-efficiency as a result of the incremental in-network distributed data stream clustering before transmission-commonly known as the computation-communication tradeoff [145].

4.2. DISTRIBUTED INCREMENTAL DATA STREAM MINING

Most WSN applications envisage large deployments of wireless sensor nodes at high redundancy to account for the unreliability of individual nodes. In order for large deployments to be cost-effective, sensor nodes are resource-constrained in terms of energy capacity, radio transmission, processing capabilities, and memory storage [146]. Transmitting data to a certain distance results in consuming several orders of larger energy than processing. Therefore, distributed local processing can offer tremendous advantages to WSNs in general. However, WSN nodes are limited in processing capability to individually accomplish computational requirements of certain applications on the acquired sensor data. Further, certain applications require simultaneous acquisition and computation of data from several nodes at distributed locations. Under these circumstances, distributed and organized cooperative processing is required.

4.3. NETWORK ARCHITECTURE FOR DISTRIBUTED INCREMENTAL DATA STREAM MINING

The proposed distributed incremental data stream mining system is coupled to a hierarchical two-tiered communication architecture of WSN. Hence, the data stream

clustering algorithm assumes that the network nodes or sensor nodes are organized in distinct hierarchical clusters of nodes with each cluster under a predefined cluster head (CH). WSN nodes under this scheme organize themselves in clusters and cooperate to perform an assigned task autonomously without intervention.

We assume a two-tiered communication architecture. The first tier consists of sensor nodes to cluster heads communication. The sensor nodes are only able to communicate two ways with their cluster heads and no sensor to sensor communication is assumed. The sensor nodes are sources of the streaming data. The second tier consists of cluster heads to Sink communication. Here, the cluster heads can communicate to one another besides communication with the Sink. Cluster heads can send data packets multi hop to the Sink via other cluster heads, therefore the network reach is extended. The cluster heads do not any data, but are purely tasked with computations and communications of sensor node data. The network architecture of the distributed incremental data stream mining system is shown in Figure 4.1.

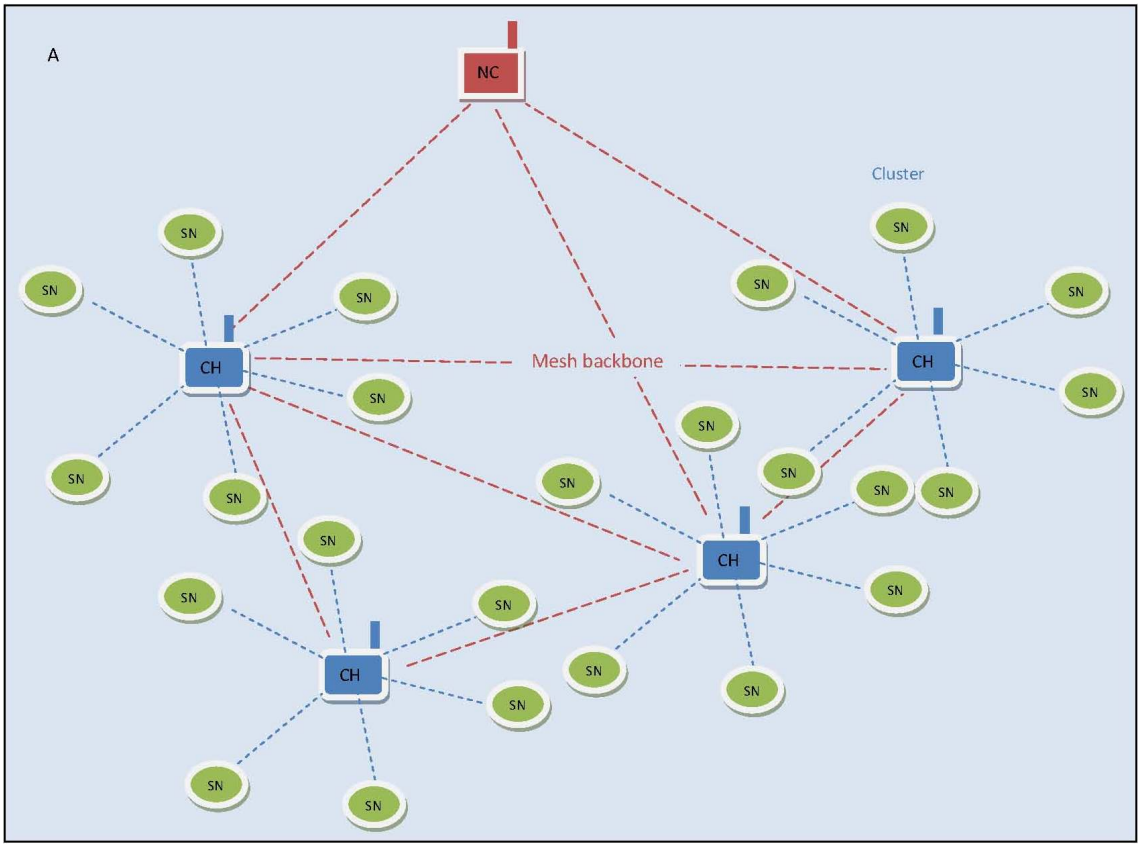


Figure 4.1: The hierarchical two-tiered WSN architecture for the distributed incremental data stream mining framework.

4.4. DISTRIBUTED INCREMENTAL DATA STREAM MINING FRAMEWORK

A detailed description of the distributed incremental data stream mining framework is presented in this section. The underlying principle of distributed incremental stream mining is to handle the sensor stream mining process in-network at distributed locations and incrementally at multiple hierarchical levels. This involves starting from simple local processing at sensor nodes to fair regional mining at intermediate nodes (CHs) and through to complete global mining at the network sink. The approach is such that as the sensor streams traverse up the network from sensor nodes via intermediate nodes and finally to the network sink, the stream processing complexity increases while the total amount of transmitted bits decreases.

This approach consists of three distinct stream processing tasks asynchronously but cooperatively revealing the underlying structure in distributed sensor data streams. These tasks are the sensor nodes, the cluster heads, and the network sink processing tasks.

4.4.1. Sensor Nodes Processing

The basic idea is for the sensor nodes to process the incoming stream locally and minimize data transmission as much as possible without compromising the accuracy of the information hidden in the stream.

Under this scheme, the sensor nodes initially transmit an item of their stream (tuple) to their respective cluster heads and wait for their cluster head's response. The Cluster heads respond to each sensor node by transmitting cluster prototypes that are computed from the received tuple and tuples received from other members of the group. Following this initial transmission, the sensor nodes continuously compare their incoming tuples to the received local cluster prototype. If their input tuples fall within their local cluster prototype with a deviation less than a predetermined threshold, from now on referred to as *local the model drift threshold*, then the sensors categorize the tuples as belonging to the local cluster prototype and avoid transmission. However, if the new input tuples deviate significantly from the local model drift threshold, then they transmit the new tuple to their cluster head and wait to receive

the new local cluster prototype. The sensor nodes, following the initial transmissions, only transmit their input tuples whenever there is significant drift in their acquired stream information.

Consider a general model of a data stream where data values are generated as stream of tuples(x_i). Let $S = (x_1, \dots, x_n)$, $n = 1,2,3, \dots \infty$ be the data stream that is continually generated at sensor nodes as time progresses, where n is the stream tuple number. Let (x_1) be the first tuple of a data stream. For a cluster of M member nodes, there will be $\{S_1, S_2, S_3, \dots, S_M\}$ data streams being generated simultaneously within the cluster.

We assume unordered, unaggregated model (cash register) of data arrivals i.e. the general case where data arrives unordered and the same value may appear multiple times within the stream [66]. The processing of massive data streams requires the use of a more restricted model of computation where data streams must be processed with the demand that each tuple in the stream must be processed completely and discarded before the next is received [147]. In this model, once a tuple has been seen, it cannot be retrieved unless it is explicitly stored in the main memory which is extremely limited for WSN nodes.

During initial transmission, the first tuple of all streams $(x_1^i, i = 1,2,3, \dots, M)$ is sent to the cluster head. The cluster head will use the *SUBFCM* (described in Chapter 5) algorithm to partition the first tuples into cluster prototypes $(c_i, i = 1,2,3, \dots, C)$ of "similar" tuples, where C is the number of cluster prototypes at the moment. The word "Similar" is context specific (i.e. two tuples are considered similar when the measure of their distance metric taken in all dimensions is a minimum). Hence each sensor node will have a cluster structure (c_i) that its stream currently belongs to. The tuple that is then generated (x_n) by a sensor node, will be compared to the received local cluster structure(c_i). If the deviation of tuple (x_n) is within a given local model drift threshold(Th), then the stream is considered in line with the local cluster structure and its transmission will be suppressed. However if its deviation exceeds the *local model drift threshold* (Th) given, then the tuples will be transmitted to the cluster head and local cluster structure update is requested. Graphical depiction of the data stream and sliding window is shown in Figure 4.2.

The sensor nodes compute the deviation of their current stream from the local cluster prototype as:

$$d_{(x_n,c_i)} = \|x\_n - c\_i\|_D \text{^} \tag{4.1}$$

where D is the dimension of the tuple.

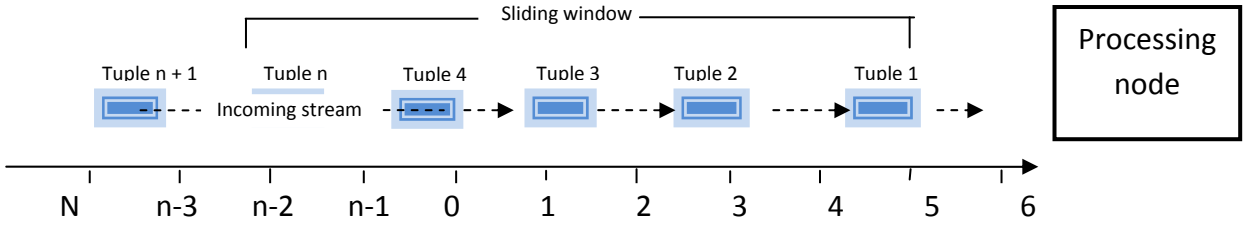


Figure 4.2: Graphical depiction of data stream and sliding window.

Algorithm 4.1: Sensor processing algorithm.

1	Input stream: $S = (x_1, x_2, x_{3,\ldots,} x_\infty)$, set threshold = Th
2	$i = 1$
3	$n = 1$
4	$Read\ \ first\ stream\ x_{n=}x_1$
5	$Send\ \ x_1\ \ to\ CH$
6	WHILE $cluster\ prototype\ (c_i)\ not\ recieved$
7	Wait to receive c_i
8	ENDWHILE
9	WHILE (1) //loop forever
10	$n = n + 1$
11	$Read\ x_n$
12	$d_{(x_n,c_i)} = \|x_n - c_i\|^D$ //D is dimension of the input stream elements
13	WHILE $d_{(x_n,c_i)} \geq Th$
14	$Send\ x_n\ to\ CH$
15	$Request\ c_i\ \ update$
16	$i = i + 1$
17	ENDWHILE
18	ENDWHILE

4.4.2. Cluster Head Processing

The basic idea of stream processing at this level is to compute local cluster prototypes of the streams generated by all member nodes. Processing here assists sensor nodes to optimize communication by sending them their local cluster prototypes so that sensors will only transmit new input streams when they detect significant deviation in their streams and request update. Cluster head processing also optimizes CH-to-Sink communication by transmitting cluster summary of their local streams to the Sink rather than the whole local stream.

Cluster heads form and maintain a short table of Tuples of their member nodes- from now on known as *Local Stream Base* (LSB). Initially the CHs cluster the Tuples in the Local Stream Base using the SUBFCM algorithm and multicast the local cluster prototypes to the associated member nodes. CHs further send local cluster prototypes and associated node ID's to the Sink for computation of global cluster prototypes for location based event cluster mapping. During subsequent stages, if CHs receive a stream Tuple from member nodes (i.e. a local cluster prototype update is requested), then they update the LSB. If certain number of set update requests, $\in$, are received, then the CHs re-compute local cluster prototypes and transmit updated local models to each member node and the Sink as well.

Consider a single cluster of the network under consideration which contains M member sensor nodes each generating a data stream. At every time instance, the cluster head, CH, receives M streams, one from each of its members. The CH maintains a sliding window of size LSB which updates randomly as local tuples are received from member nodes which also represent local cluster prototype update requests.

Initially, the sliding window will be filled by the first tuples of all member nodes as:

$$(x_1^i, i = 1,2,3, \dots, M) \tag{4.2}$$

The tuples in (4.2) are the first entry for streams being generated at M geographic locations within the cluster and are given as:

$$\{S_1, S_2, S_3, \dots, S_M\} \tag{4.3}$$

Subsequently, the sliding window will be updated only when the member nodes detect changes in their local stream concept and send their current tuples. Therefore the sliding window update is randomly based on the local stream concept drift. The subsequent sliding window content can be represented as:

$$(x_n^i, i = 1,2,3, \dots, M) \tag{4.4}$$

Upon receiving a set local cluster prototype update requests, $\in$, the CH, recomputes local cluster prototypes, $(c_i, i = 1,2,3, \dots, C)$, using the SUBFCM algorithm and sends the updated local models to each member nodes and the Sink.

Algorithm 4.2: Cluster heads processing algorithm.

```
1   INPUT: initialize LSB, set maximum update request ∈
2   n = 1
3   i = 1
4   update request = 0                    //update request is no. of update requests
    received
5
    WHILE  i < M                          // M is no. of active cluster members
6
7           IF    x_n^i is received       //x_n^i  is the n^th stream item from i^th  cluster
8   member
9                 LSB[i] = x_n^i
                  i = i + 1
10          ENDIF
11
    ENDWHILE
12
13  compute cluster prototype (c_i) from LSB
    send c_i to corresponding member nodes
14  send c_i to sink
15
    i = 1
16
    WHILE (1)                     // loop forever
17      n = n + 1
18      IF  x_n^i is received
19            LSB[i] = x_n^i
20            update request = update request + 1
21
22              IF  update request = ∈
23                    recompute c_i from LSB
24                    send c_i to corresponding member nodes
25                    send c_i to sink
                      update request = 0
                ENDIF
26
27        ENDIF

    ENDWHILE
```

4.4.3. Sink Processing

The network sink as being the most capable node undertakes the most computationally intensive task. Sink processing is mainly to extract the global stream cluster prototypes based on the local cluster prototypes received so far and present the sensor stream mining results and facilitate sensor stream analysis for a user.

The sink maintains a table of stream clusters and associated node IDs of all previous transactions- now on known as Global Stream base (GSB). On every communication with CHs the Sink updates its GSB and performs global stream clustering.

Assume the network contains a total of N sensor nodes organized under M clusters. As described above, at each time instance, every cluster head sends the local cluster prototypes to the Sink given as:

$$(c_i, i = 1,2,3, \dots, C) \tag{4.5}$$

Each of the local cluster prototypes represent the M data streams given in Equation 4.3. Therefore the GSB at any time instance contains M such local cluster prototypes given as

$$\{(c_i, i = 1,2,3, \dots, C)_m, m = 1,2,3, \dots M\} \tag{4.6}$$

The Sink computes global cluster prototypes using the SUBFCM algorithm every time there is an update to its GSB entry. The sink node possesses abundant computational, energy, and memory resource due to its physical connection to a base station. Therefore, besides handling the global cluster prototype model computations, it also stores all the local prototypes for historical analysis. Due to the same reason of abundance of resource, the Sink computational, communication complexities and energy consumption are not analyzed.

Chapter 5

5. DEVELOPMENT OF SUBTRACTIVE FUZZY CLUSTER MEANS (SUBFCM) ALGORITHM

This chapter describes the Subtractive Fuzzy Cluster Means (SUBFCM) algorithm developed for distributed incremental data stream mining in a resource-limited environment such as WSN. The SUBFCM algorithm is designed to be embedded and run on resource (such as computation, memory and power) limited nodes of WSNs. It builds a base for cluster mining techniques. SUBFCM combines subtractive clustering with fuzzy c-means algorithms to achieve clustering without the need of the number of partitions within the data space to be known a priori. SUBFCM is implemented as an autonomous unsupervised learning algorithm that feeds on distributed streaming data within a WSN. Its primary purpose is to minimize energy consumption of individual WSN nodes and consequently extend network lifetime. Following the strategy of computation-communication trade-off, the algorithm performs local pattern discovery within individual nodes and sends only the necessary information over the network.

5.1. CLUSTERING

Clustering has been one of the most widely studied topics in the data-mining field. K-means and fuzzy C-means clustering algorithms have been two of the most popular algorithms in this field. Fuzzy c-means (FCM) is a method of clustering developed by Dunn [148] and later improved by Bezdek [149]. FCM allows a piece of data to belong

to two or more clusters with varying degree of memberships. In real applications there is very often no sharp boundary between clusters so that fuzzy clustering is often better suited for the data. Membership degrees between zero and one are used in fuzzy clustering instead of crisp assignments of the data to clusters. The most prominent fuzzy clustering algorithm is the fuzzy c-means, a fuzzification of k-means. The Fuzzy C-Means (FCM) algorithm is the most widely used clustering algorithm in the field of data mining. It allows one piece of information to belong to two or more clusters. One of the drawbacks of FCM in exploratory data analysis is that it requires the number of clusters within the data space to be known beforehand. When the purpose of clustering is to automatically partition multivariate data coming from a dynamic source, the number of partitions in the data space is typically unknown. Hence in this research, subtractive clustering and FCM algorithms are combined to implement an algorithm that does not require prior information about the number of clusters in the data space. The proposed algorithm is called **S**ubtractive **F**uzzy **C**luster **M**eans algorithm (SUBFCM) [150], and is described below.

5.2. DATA STREAM MINING ALGORITHM

Data stream mining is the process of extracting knowledge structures from continuous data streams. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only a small number of times using limited computing and storage capabilities. Examples of data streams include, among others, computer network traffic, phone conversations, ATM transactions, web searches, and sensor data. In data mining we are interested in techniques for finding and describing structural patterns in data as a tool for helping to explain that data and make predictions from it [151]. One of the popular data mining techniques in a centralized environment is data clustering. The general goals of a clustering technique is to decompose or partition data sets into groups such that both intra-group similarity and inter-group dissimilarity are maximized [125].

WSNs can benefit a great deal from stream mining algorithms in terms of energy conservation and efficient services. However, for WSNs to achieve significant

energy conservation, the data stream mining has to be distributed within the network due to their resource constraints [150, 152].

Data mining applications place special requirements on clustering algorithms including: the ability to find clusters embedded in subspaces of high dimensional data, scalability, available to the WSN nodes. The algorithm is also required to produce frequent summaries of the corresponding inputs from the network sensor nodes. In stream mining [153, 154], WSN data mining applications further place strict requirements on the underlying algorithm. Collecting data generated in a WSN to a central location and performing data mining is undesirable due to the energy and bandwidth limitations. Therefore, the data mining algorithm has to perform in-network and autonomously on limited-resource applications, and the algorithm has to converge as fast as possible over the limited data sets so that the processor can take on the next set of streams.

5.3. SUBTRACTIVE CLUSTERING METHOD

The subtractive clustering method was developed by Chiu [155]. It is a modification of mountain clustering [156] with improved computational complexity. The subtractive clustering method assumes that each data point is a potential cluster center (prototype). A data point with more neighboring data will have a higher potential to become a cluster center than points with fewer neighboring data. In the subtractive clustering method, the computation is proportional to the number of data points and is independent of the dimension of the problem. Subtractive clustering considers a data point with the highest potential as a cluster center and penalizes data points close to the new cluster center to facilitate the emergence of new cluster centers. Based on the density of surrounding data points, the potential value for each data point is calculated as follows:

$$pot_i = \sum_{j=1}^{n} e^{-\alpha \left\| u_i - u_j \right\|^2} \tag{5.1}$$

where u_i, u_j are data points and $\alpha = \frac{4}{r_a^2}$, r_a is a positive constant defining a neighborhood. Data points outside this range have little influence on the potential.

Following the potential calculation of every data point, the point with the highest potential is chosen as the first cluster center. Let u_k be the location of the first cluster center and pot_k be its potential value. The potential of the remaining data points u_i is then revised by

$$pot_i = pot_i - pot_k e^{-\beta \|u_i - u_k\|^2} \tag{5.2}$$

where $\beta = \frac{4}{r_b^2}$ and r_b is a positive constant ($r_b > r_a$).

Thus, the data points near the first cluster center will have greatly reduced potential, and therefore are unlikely to be selected as the next cluster center. The constant $r_b$is the neighborhood defining radius and will have significant reduction in potential. r_b is set to be greater than $r_a$to avoid closely spaced centers. The ratio between r_a and r_b is called the Squash factor (SF) which is a positive constant greater than one.

The potential update process (5.2) will continue until no further cluster center is found. The parameters known as acceptance ratio (AR) and rejection ratio (RR) together with the influence range and squash factor set the criteria for the selection of cluster centers. The accept ratio and reject ratio are upper acceptance threshold and lower rejection threshold, respectively and they take a value between zero and one. The accept ratio should be greater than the reject ratio.

First criterion: if the potential value ratio of the current data point to the original first cluster center is larger than the acceptance ratio, then the current data point is chosen as a cluster center.

Second criterion: if the potential value falls in between that of the acceptance and rejection ratios, then the compensation between the magnitude of that potential value and the distance from this point to all the previously chosen cluster centers (relative distance) is taken into consideration. If the sum of the potential value and the ratio of the shortest distance between the current data point and all other previously found cluster centers to the influence range is greater than or equal to one, then the current data point is accepted as a cluster center.

Third criterion: If the sum of the potential value and the ratio of the shortest distance between the current data point and all other previously found cluster centers to the influence range is less than one, then the current data point is rejected as a cluster center.

Fourth criterion: if the potential value ratio of the current data point to the original first cluster center is less than the rejection ratio, then the potential value of the current data point is revised to zero and the data point with the next highest potential is tested.

Summary:

Algorithm 5.1: Subtractive clustering algorithm.

Initialize parameters; r_a, r_b , AR, *and* RR

Calculate potential of each data point pot_i *using equation (5.1)*

Set the maximum potential as pot_k

Choose data point corresponding to pot_k *as the cluster center candidate*

If $pot_i > AR * pot_k$, *then accept* u_i *as a cluster center*

    update the potential of each point using equation (5.2) and continue

else if $pot_i < RR * pot_k$, *then reject* u_i *as a cluster center*

else

    Let d_r *be relative distance*

If $\frac{d_r}{r_a} + \frac{pot_k}{pot_i} \geq 1$ *accept* u_i *as a cluster center*

    update the potential of each point using equation (5.2) and continue

else

---

12 *reject u_i and set the potential $pot_i = 0$*

13 *select the data point with the next highest potential as the new candidate and re-test*

14 *endif*

15 *endif*

5.4. FUZZY C-MEANS CLUSTERING

Fuzzy clustering algorithms are based on minimization of the fuzzy c-means objective function formulated as:

$$J_o = \sum_{c=1}^{C} \sum_{i=1}^{m} (v_{ci})^{\theta} \left\| u_i - v_i \right\|^2 A \tag{5.3}$$

where v_{ci} is a fuzzy partition matrix of u,

$$v = [v_1, v_2, \ldots, v_c] \tag{5.4}$$

is a vector of cluster centers, which have to be determined,

$$d_{ciA}{}^2 = \left\| u_i - v_i \right\|^2{}_A = (u_i - v_i)^T A (u_i - v_i) \tag{5.5}$$

is a squared inner-product distance norm, and

$$\theta \in [1, \infty) \tag{5.6}$$

is a parameter which determines the fuzziness of the resulting clusters.

The conditions for a fuzzy partition matrix are given as:

$$v_{ci} \in [0,1], \qquad 1 \leq c \leq i, \qquad 1 \leq i \leq n \tag{5.7}$$

$$\sum_{i=1}^{c} v_{ci} = 1, \qquad 1 \le i \le n \tag{5.8}$$

$$0 < \sum_{i=1}^{n} v_{ci} < N, \quad 1 \le i \le c \tag{5.9}$$

The value of the objective function (5.3) can be seen as a measure of the total variance of u_i from v_i.

The minimization of the objective function (5.3) is a nonlinear optimization problem that can be solved by iterative minimization, simulated annealing or genetic algorithm methods. The Simple iteration method through the first-order conditions for stationary points of (5.3) is known as the fuzzy c-means (FCM) algorithm.

The stationary points of the objective function (5.3) can be found by adjoining the constraint (5.8) to J_o by means of Lagrange multipliers:

$$J = \sum_{c=1}^{C}\sum_{i=1}^{n} (v_{ci})^{\theta} d^2{}_{ciA} + \sum_{i=1}^{n} \lambda_i \left[\sum_{c=1}^{C} v_{ci} - 1\right] \tag{5.10}$$

By setting the gradient of J with respect to the fuzzy partition matrix u, the vector of cluster matrix v, and λ to zero.

Now if $d_{ciA}^2 > 0, \forall i, c$ and $\theta > 1$, then (u, v) may minimize the objective function (5.3) only if

$$v_{ci} = \frac{1}{\sum_{j=1}^{C} \left(d_{ciA}/d_{cjiA}\right)^2 \ /(\theta-1)}, 1 \le j \le C, \ \ 1 \le i \le n \tag{5.11}$$

and

$$v_c = \frac{\sum_{i=1}^{n} (v_{ci})^{\theta} u_i}{\sum_{i=1}^{n} (v_{ci})^{\theta}}; \qquad 1 \le c \le C \tag{5.12}$$

This solution also satisfies the constraints (5.7) and (5.9). Equations (5.11) and (5.12) are the first-order necessary conditions for stationary points of the objective function (5.3). The FCM algorithm iterates through (5.11) and (5.12). Sufficiency of the necessary conditions (5.11) and (5.12) as well as the convergence of the FCM algorithm is proven in [157].

Before using the FCM algorithm, the parameters: number of clusters,C, fuzziness exponent, θ, termination tolerance, ε, the norm-inducing matrix, A, and the fuzzy partition matrix, u, must also be initialized suitably. Note that the FCM algorithm converges to a local minimum of the objective function (5.3) if these parameters are not initializationed suitably. Therefore different initializations may lead to different performance results.

Summary:

Algorithm 5.2: Fuzzy C-Means clustering algorithm.

Initialize parameters: C, θ, ε, A, *and* u

Repeat for $l = 1,2,3, \ldots$

  Compute cluster centers (prototypes):

  $v_c = \frac{\sum_{i=1}^{n}(v_{ci})^{\theta} u_i}{\sum_{i=1}^{n}(v_{ci})^{\theta}}; \qquad 1 \le c \le C$

  Compute distances:

  $d_{ciA}^2 = (u_i - v_c)^T A(u_i - v_c), \quad 1 \le c \le C, \quad 1 \le i \le n$

  Update the partition matrix:

  For $1 \le i \le n$

    If $d_{ciA} > 0$ *for all* $c = 1,2, \ldots, C$

      $v_{ic} = \frac{1}{\sum_{j=1}^{C}(d_{ciA}/d_{cjA})2/(\theta-1)}$

    else

      $v_{ci} = 0$ *if* $d_{ci} > 0$, *and* $v_{ci}\varepsilon[0,1]$ *with* $\sum_{c=1}^{C} v_{ci} = 1$

Until $\left\| v_{ci}^{(l)} - v_{ci}^{(l-1)} \right\| < \varepsilon$

5.5. THE SUBFCM ALGORITHM

Embedding an autonomous cluster mining algorithm in WSN nodes requires that the algorithm take data sets as input and generate output without data preprocessing. In applications where the number of clusters in a data set must be discovered, the FCM algorithm cannot be used directly. For clustering WSN data autonomously, the number of cluster prototypes (categories) has to be determined from the data sets. Hence in this research, Subtractive clustering and FCM algorithms are combined to implement an algorithm that determines the number of clusters in the data space from the input data sets- Subtractive Fuzzy Cluster Means (SUBFCM).

The SUBFCM algorithm uses a subtractive clustering approach to determine the number of cluster prototypes C and the prototype centers c. The algorithm then partitions the stream into C fuzzy clusters using the prototype centers from the above step as initial fuzzy cluster centers.

Initially, the SUBFCM algorithm assumes each D-dimensional data point $u_i, i = 1,2,3,...,n$ as a potential cluster center with a measure of potential (pot) of data points in the stream as;

$$pot_i = \sum_{j=1}^{n} e^{-\alpha \|u_i - u_j\|^2} \tag{5.13}$$

where $\alpha = \frac{4}{r_a^2}$ and r_a is a positive constant defining cluster radius. A large value of r_a results in fewer large clusters, while smaller values result in a greater number of smaller diameter clusters. $\|\ \|$ Denotes the Euclidean distance, which defines the distance between two points $u_1(x_i, y_{1,} z_1)$ and $u_2(x_2, y_{2,} z_2)$ as being equal to the length of vector $\|X_1 - X_2\| = \sqrt{(x_1 - x_2)^2 + (y - y_2)^2 + (z_1 - z_2)^2}$ where $X_1 \equiv \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$ and $X_2 \equiv \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$

The measure of potential for a given data point is a function of its distances to all other points. A data point with many neighboring points will have a high potential value. After computing the potential for every point, the point u_k with the highest potential pot_k will be selected as the first cluster center c_1. The potential for every other point is then updated by (5.14):

$$pot_i = pot_i - pot_k e^{-\beta \|u_i - u_k\|^2} \tag{5.14}$$

where $\beta = \frac{4}{r_b^2}$ and r_b is a positive constant that can be set to a value which is greater than r_a. After the first cluster center is determined, the value of r_b determines the potential of data points becoming subsequent cluster centers. Setting r_b > r_a reduces the potential of data points close to the first cluster center and hence avoids closely spaced cluster centers [158]. The parameter $\varepsilon$is a stopping criterion and should be selected within (0, 1) [159]. $\alpha\varepsilon$ with a value close to zero will result in a large number of hidden centers whereas $\alpha\varepsilon$ close to one leads to a small network structure.

Following the update process, the data point with the highest remaining potential is selected as the next cluster center c_2, and the process repeats until a given threshold ε for the potential is reached and C such centers are computed. SUBFCM then uses the clustering criterion of squared distance ${d_{ci}}^2$ between the $i-th$ stream sample and the $c-th$ prototype and defines the objective function (J_0) as:

$$J_o = \sum_{c=1}^{C} \sum_{i=1}^{n} {v_{ci}}^{\theta} {d_{ci}}^2 \tag{5.15}$$

where the squared distance function is given as:

$$d_{ci}^{\ 2} = \left\| u_i - c_c \right\|^2 \tag{5.16}$$

where v_{ci} represents the membership degree of the $i-th$ stream sample to the $c-th$ cluster.

Membership is determined under the conditions:

$$v_{ci} \in [0,1],\ \ c = 1,2,3,...,C,\ i = 1,2,3,...,n \tag{5.17}$$

$$\sum_{c=1}^{C} v_{ci} = 1, \qquad i = 1,2,3,...,n \tag{5.18}$$

where θ is a weighing exponent or fuzziness measure. If $\theta = 1$, the clustering model is reduced to the hard K-means model. The larger θ, the fuzzier the memberships is. θ is usually set to 2 [149].

The stream partitioning takes place by optimizing the criterion function (5.15) through iteration updating the cluster prototype centers c_j and the membership function v_{ci} as (5.19) and (5.20) respectively:

$$c_j = \frac{\sum_{i=1}^{n} \left(v_{ij}\right)^{\theta} u_i}{\sum_{i=1}^{n} \left(v_{ij}\right)^{\theta}} \tag{5.19}$$

$$v_{ci} = \left(\sum_{l=1}^{C} \left(\frac{\|u_i - c_j\|}{\|u_i - c_l\|} \right)^{2/(\theta-1)} \right)^{-1} \quad (5.20)$$

The iteration should stop when;

$$\max \quad = \left\{ \left| v_{ci}^{(l+1)} - v_{ci}^{(l)} \right| \right\} < \varepsilon \quad (5.21)$$

where ε is the termination criterion, $0 < \varepsilon < 1$ and l is the iteration step.

SUBFCM takes the fuzzy radius (r_a) and fuzziness measure (θ) as inputs and autonomously reveals the structures in the data stream space. The parameter r_a determines the granularity of the structures. The smaller it is, the higher the resolution of the structures and the more computation overhead. The SUBFCM Algorithm steps are shown in Algorithm 3.

Algorithm 5.3: Subtractive fuzzy C-Means algorithm.

Step0: Specify fuzzy radius r_a, α, ε, and fuzziness measure θ

Step1: *For each data object* u_i*;*

*{**1.1** Calculate potential measure -eq (5.13)*

1.2 *Choose max (* pot_i *) as first cluster Center* cj

1.3 *Revise the potential measures –eq (5.14)*

1.4 *If max* $pot_i > \alpha$ *, j=j+1, go to **1.2***

Else end; Set $C = j$

}

Step2: *Calculate* d_{ci} *-eq (5.16) ad Initialize*

$v_{ci} = d_{ci}$*;* *Set iteration no. to* l

Step3: *Increment* l *(*$l = l + 1$*)*

3.1 *Calculate Cluster center* c_j *-eq (5.19)*

Set centers to $c_{(l+1)} = c_j$

3.2 *Calculate membership* v_{ci} *-eq (5.20)*

Set membership to $v_{ci(l+1)} = v_{ci}$

Step4*: If* $v_{ci(l+1)} - v_{ci(l)} > \varepsilon$

go to ***step3***

else end; *Output [*c_{ci}*], [*c_j*]*

5.6. IMPLEMENTATION OF SUBFCM

The SUBFCM algorithm was initially implemented in MATLAB to understand and investigate its characteristics. MATLAB provides toolbox functions readily available for clustering analysis. It also provides visual output of the clustering statistics for user to validate and debug the algorithm. The SUBFCM algorithm's response to continuously streaming data has been analyzed. The main constraints of low power wireless sensor nodes, computation complexity, processing time and energy requirements were considered before porting the SUBFCM algorithm as an embedded task suitable for WSN nodes.

Given the desired fuzzy radius r_a for the particular application under consideration and fuzziness measure θ, the algorithm reveals clusters hidden in the streaming data. As soon as a certain number of elements from the stream enter the buffer of the processing task, the subtractive clustering method will be invoked calling the *subclust* function from the MATLAB toolbox. The *subclust* function takes as a minimum, the set of vector data X to be clustered and fuzzy radius r_a. It returns the Cluster centers C, and range of influence of the cluster centers S in each of the data dimensions. The cluster centers returned by the *subclust* function are used to initialize the fuzzy clustering method. The MATLAB FCM function performs the fuzzy clustering of the input vector data X. The FCM function takes inputs data X and C as initial

cluster centers and produces fuzzy cluster centers c, and the fuzzy membership matrix U as well as the values of the objective function Obj_fun during the iterations.

The MATLAB toolbox subclust function format:

Inputs: $[X, r_a]$ Outputs: $[C, S]$

[C,S] = subclust (X, r_a)

The MATLAB toolbox FCM function format:

Inputs: $[X, C]$ Outputs: $[c, U, Obj\_fcn]$

$[c, U, Obj\_fcn\ ]$ = fcm(X, C)

A database consisting of 200 instances each containing weather parameters were clustered using SUBFCM algorithm. Each instance of the database has its classification of fire weather index (FWI) rating. There are five FWI ratings in the instances of the database; low, moderate, high, and extreme. Each of the FWI ratings present in the database has 50 samples that in total summarize the 200 instances of the database. Table 5.1 below shows a small sample of the instances contained in the weather database. The number of iterations required to partition the sample weather data sets is shown In Figure 5.1.

Table 5.1: Sample of weather database.

Instance	Temperature	Rel. Humidity	Wind Speed	Rain fall	FWI	class
1	26	50	11	0	13.65	M
2	14.1	89	14.9	0	6.938	M
3	26.9	37	16.4	0	19.57	H
4	15.6	78	18.5	0	12.55	M
5	26.1	32	39.8	0	57.63	E
6	10.9	76	12.9	5.2	0.629	L
7	18.4	55	28.8	0	10.64	M
8	20.4	43	45.3	0	42.23	E
9	26.8	40	35	0	42.05	E

The results obtained by running the SUBFCM on the database are:

Cluster centers:

C1	13.9	76.62	4.1	3.97
C2	17.53	64.38	21.37	0.435
C3	21.28	46.69	21.36	0.136
C4	24.05	33.55	32.26	0.334

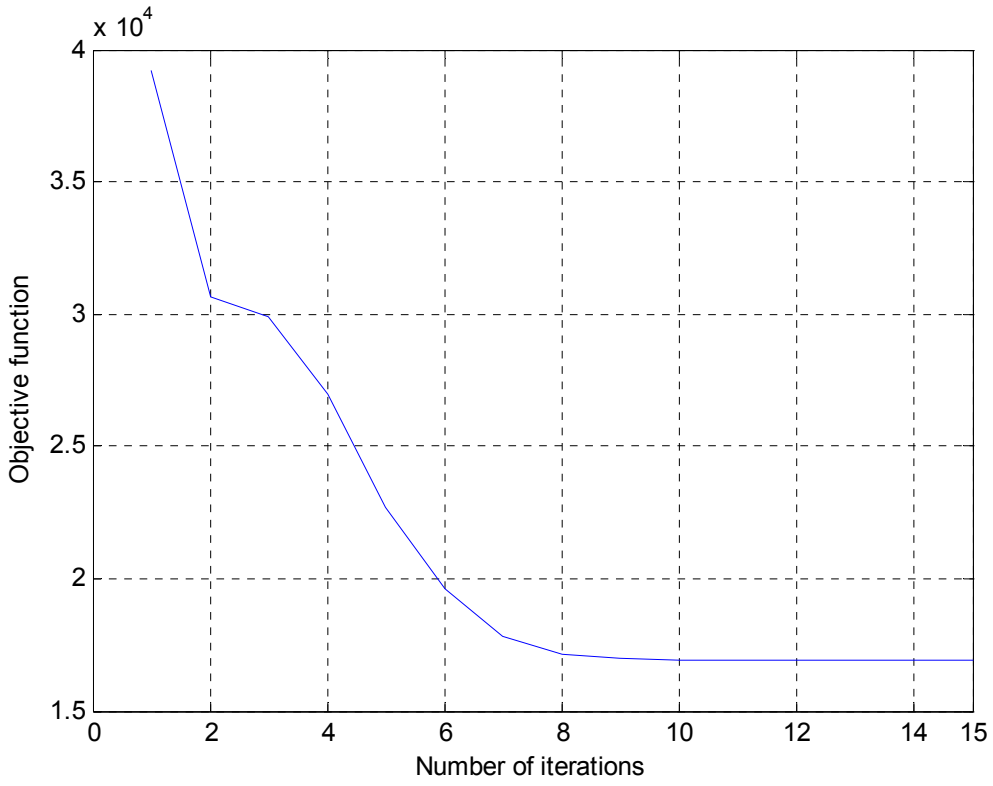


Figure 5.1: The objective function minimization.

The graphs below show the classification of each instance by comparing each pair of attributes present in the database. The SUBFCM algorithm generated classes are shown in different colors; blue for low FWI class, red for moderate FWI class, green for high FWI class, and magenta for extreme FWI class. The database described above contains four attributes (dimensions); temperature, relative humidity, wind speed and rainfall. The different clusters do not seem to be well separated in 2D depictions. However, they are clearly separate clusters in 3D graph.

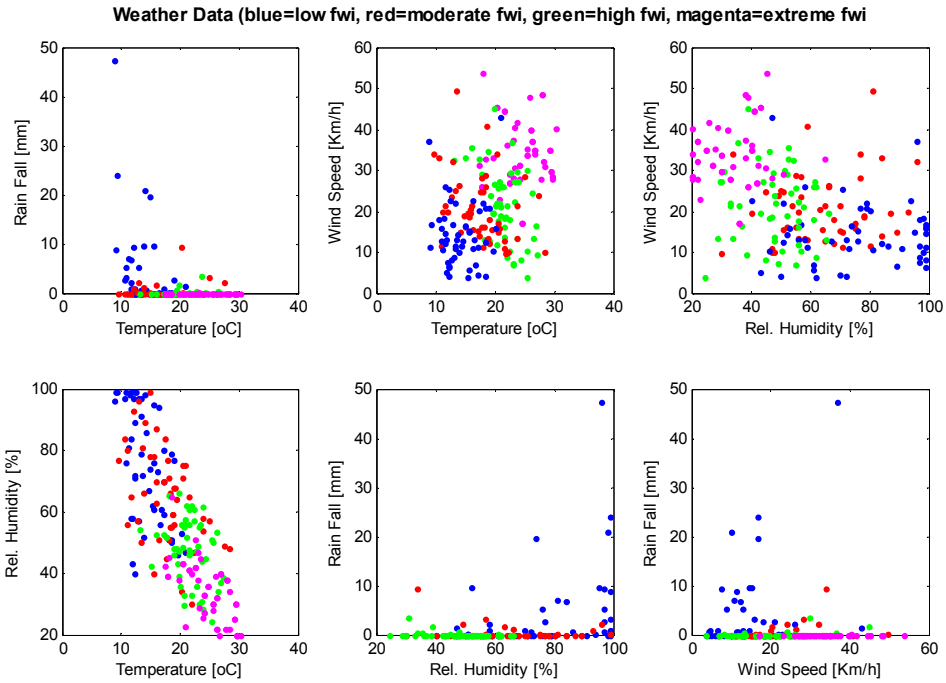


Figure 5.2: 2D classification of the data points from weather database.

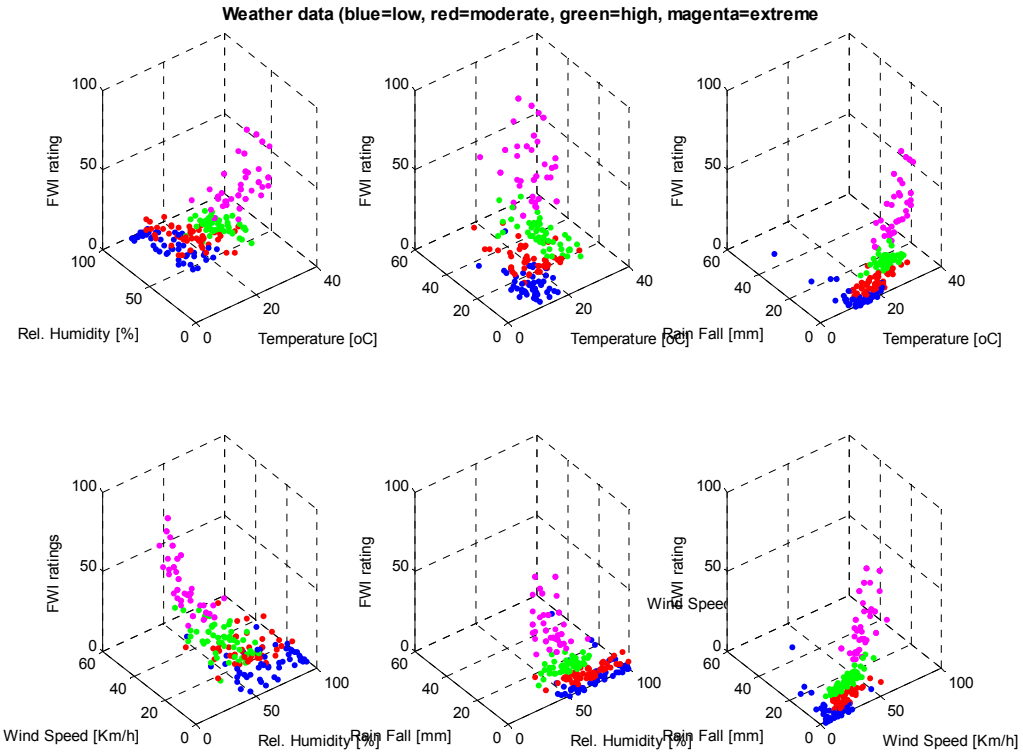


Figure 5.3: 3D classification of the data points from weather database.

5.6.1. Computational Complexity of the SUBFCM Algorithm

The computational complexity of the iterations of the SUBFCM algorithm is assumed to be constant as it computes the distance calculations and centroid recalculations. Distance calculations require approximately $(3ncd + nc + nd + cd)$ floating point operations (flops) per iteration [160], where n denotes the number of total data objects, c denotes the number of clusters, and d denotes the dimension of data objects. Each centroid recalculation requires approximately (cd) flops per iteration. Therefore, we can estimate the computational complexity of the SUBFCM algorithm as:

$$T_{comp} = (3ncd + nc + nd + cd)IT_{flop} \tag{5.22}$$

where T_{comp} is the computational time of implementing the algorithm, I is the number of iterations, and T_{flop} denotes the time for each floating point operation. Under the condition that n is large compared to both c and d, T_{comp} reduces to:

$$T_{comp} = (3ncd)IT_{flop} \tag{5.23}$$

In our experiments, fixed point arithmetic is used instead of floating point arithmetic for all computations without compromising the accuracy significantly. Therefore, the complexity of the algorithm is:

$$T_{comp} = (3ncd)IT_{fixop} \tag{5.24}$$

where T_{fixop} is the time for each fixed point operation.

In Figure 5.1, the number of iterations required to partition a typical weather data sets is shown to be very low. Hence, for a wireless sensor node with reasonable amount of millions of instructions per second (MIP) this is a fairly light operation.

5.6.1.1. Sensor node Computational Complexity

The Computational complexity of the sensor node processing task described above can be expressed as the combination of complexities of sensor stream acquisition, similarity calculation, and determining send or suppress operations. Sensor stream acquisition takes the form of linear time complexity which may be expressed as:

$$O(n) \tag{5.25}$$

where the tuples dimension D determines the upper and lower bounds of n.

The similarity calculation task goes through sequential comparison of the newly generated tuples to a constant value. The dimensionality of the tuple influences the complexity of this task. For a given tuple dimension the complexity of this task is linearithmic time expressed as:

$$O(n \log n) \tag{5.26}$$

The determination to send or suppress the incoming sensor stream average is a constant time complexity given as:

$$O(1) \tag{5.27}$$

Therefore, the time complexity $T(n)$ of the sensor node processing is:

$$T(n) = O(n\log(n)) + O(n) + O(1) \tag{5.28}$$

5.6.1.2. Sensor node Communication Complexity

The communication complexity of the sensor nodes processing task obviously grows with the dynamics of the situation under observation. When the newly incoming stream tuples are more frequently dissimilar to the current tuples the algorithm directs more send operations than suppress. This will have an effect of increasing the computational complexity of the next level of processing tasks. The worst case communication complexity of this task reduces the sensor nodes process to a simple acquire-and-transmit mode.

5.6.1.3. Sensor node Energy Consumption

The sensor nodes processing task energy consumption analysis is based on the simplified microcontroller unit model of [161] of equation (5.29).

$$E_p = N_{cyc} * C_{avb} * V^2 + V(I_o e^{\frac{V}{nV_t}})\frac{N_{cyc}}{f} \tag{5.29}$$

where $N_{cyc}$is the average number of clock cycles needed for the task, C_{avg} is the average capacitance switched per cycle, V is supply voltage, I_o is the leakage current, and f is the clock frequency.

The second term of the Equation 5.29 is eliminated as the leakage current is insignificant at high clock frequencies.

For the similarity calculation task, if the number of clock cycles required for the subtraction operation is s, and the number of clock cycles required for multiplication operation is ml, then the total energy consumption of this task can be calculated as:

$$E_p = N_{cyc} * \ C_{avg} * \ V^2 * D \ (s + ml^2) \tag{5.30}$$

The task to determine sending or suppressing the current average tuple is an obvious compare operation given as:

$$E_p = N_{cyc} * \ C_{avg} * \ V^2 * D * c \tag{5.31}$$

Where c is the number of clock cycles for the compare operation.

Therefore the total energy consumption of the sensor nodes task are dominated by the components of Equation 5.30 and Equation 5.31 as given by:

$$E_p = N_{cyc} * \ C_{avg} * \ V^2 * D \ (c + s + ml^2) \tag{5.32}$$

5.6.1.4. Cluster Head Computational Complexity

The computational complexity of the cluster head processing task is that of the SUBFCM algorithm given in section 5.6.1 with an added term of constant time complexity

$O(1)$n in the worst case. Computational complexity of the cluster head processing task is directly proportional to the communication complexity of the sensor nodes processing task.

5.6.1.5. Cluster Head Communication Complexity

The communication complexity of the cluster head processing task also grows with the dynamics of the situation being observed. The communication complexity of this task is expected to increase whenever the change in situation observed spans multiple sensor nodes. This task updates both the sensor nodes and the sink about the current state of the situation and hence the worst case scenario communication complexity of this task is double that of the sensor nodes' processing task.

5.6.1.6. Cluster Head Energy Consumption

Energy consumption of the cluster head processing task can also be derived from the fundamental equation of [161] given in Equation 5.29. Energy consumption of the cluster head processing task is dominated by that of the SUBFCM algorithm's number of iterations required to converge to optimal number of cluster prototypes.

$$E_p = N_{cyc} * C_{avg} * V^2 * D * (C + h + g) \quad (5.33)$$

where C is the number of cluster prototypes, h is the number of iterations it takes to determine the number of prototypes C, and g is the number of iterations to converge to the optimal cluster prototypes.

5.7. DISCUSSION

The SUBFCM algorithm is a light unsupervised method for the analysis of data and construction of models from data. It has been shown that a systematic combination of subtractive clustering and fuzzy c-means clustering algorithms-SUBFCM is a light data clustering algorithm computationally suited for low resource systems like WSN. Its implementation in MATLAB environment has been discussed.

Chapter 6

6. MODELLING AND SIMULATION OF THE DISTRIBUTED INCREMENTAL DATA STREAM MINING WSN

This chapter presents a detailed description of the distributed incremental data stream mining WSN system modeling and simulation. The various components of the distributed incremental data stream mining approach for WSN systems were theoretically analyzed in chapter 4. The data stream sources, the data stream mining algorithm, the algorithm processing units, and the communication between the processing units are individually expressed in terms of mathematical and simulation models to characterize and analyze their various features. The characteristics of the models would be used to explain the overall system performance and integration. Results from the integration of the models into a system and simulations should provide design guidelines for an actual WSN system design to enable distributed incremental data stream mining applications and determine the system capacity bounds.

6.1. DATA STREAM ACQUISITION (SOURCES)

Data streams represent input data that comes at a very high rate that stresses communication and computing infrastructures and storage infrastructures to some extent. Data streams have features that describe the nature of their source, which influence the complexity of the process handling them. Typical data stream features

are rate of arrival, dimensionality of its instances, and the model used to describe the underlying source.

In WSNs, the data sources are either sensor probes onboard the nodes or over-the-air data packets received from other nodes of the network. The way in which the nodes are set to access their onboard sensor probes and/or set to receive network packets over-the-air, therefore, determines the data stream feature rate of arrival. The number of onboard sensor probes and the number of physical phenomena of interest the node is subscribed to receive over-the-air also determines the dimensionality of the data stream instances. Moreover, when the system is serving applications such as continuous environmental monitoring, the data stream model can be time series.

Here, for the purpose of facilitating the system simulations, the data stream sources are modeled as processes that access stored data files in a specific manner. Two types of data stream source model processes are defined. One process representing the sensor nodes data stream source model and the other representing the cluster heads data stream source model.

The sensor nodes data stream source process periodically accesses a data file to form a periodic time series. The data files, one for each sensor node, contain multidimensional data captured from a continuous stream source for a specific time period to acquire a sufficient number of instances that can last for the duration of the simulations. Starting from the first instance in the file, the process accesses the data instances each period sequentially.

The sensor nodes data stream source model generates a stream as:

$$U: [u_i, u_{i+1}, u_{i+2}, u_{i+3}, \dots \infty] \tag{6.1}$$

where for each period, T, the i^{th} instance from the data file is inserted into the input data stream. The sensor data stream source process chart is shown in Figure 6.1.

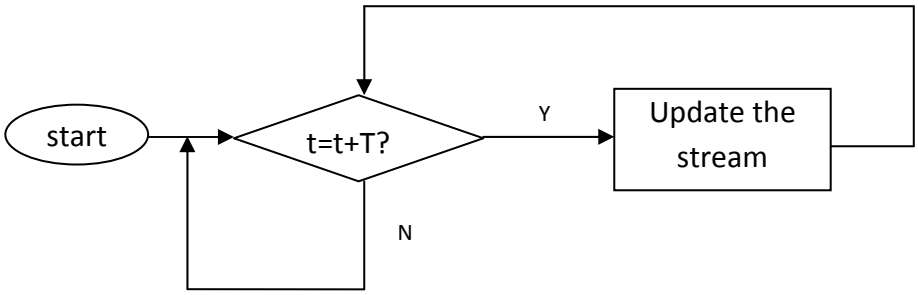


Figure 6.1: Flow chart of the sensor data stream source process.

Figure 6.2 below shows the Simulink model of the sensor node data stream source. The Logical operator block "AND" combines the Pulse Generator and the Step blocks to form a periodic timer that fires every specified period. The counter block increments an index value every time the periodic timer event occurs as long as the end of file, specified by the Constant block, is not reached. The Embedded MATLAB Function block implements a set of low-level MATLAB native file (.mat) read functions. The Embedded MATLAB Function block reads data sets from the From File block pointed to by the index variable each timer event and continually updates the sensor node's input data buffer. This Model forms a data stream source for the simulation model. The data stream rate is determined by the timer period.

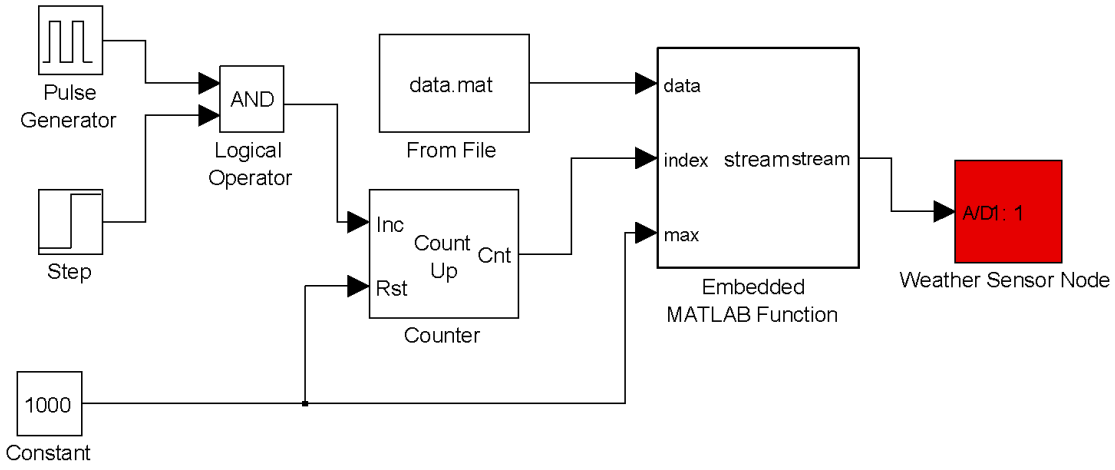


Figure 6.2: The sensor node data stream source model.

The cluster head data stream sources are the sensor nodes that subscribe to membership of the cluster head. The cluster head data stream source process handles the network interrupts for incoming over-the-air data packets. Each network interrupt for incoming data packet is an instance of the stream. This process is therefore modeled as a Callback function to handle the interrupts and form a data stream that the cluster head node acts upon. The data streams at the cluster heads form a sliding window model where a function of interest is computed over a fixed-size window in the stream. As time progresses, instances (items) from the end of the window are removed from consideration while new instances from the stream take their place. Therefore, only the last W (window size) items to have arrived are considered relevant at any moment.

The cluster heads data stream source model generates a stream as:

$$U: [u_i, u_{i+1}, u_{i+2}, u_{i+3}, \dots, u_W], u_{i+W}, u_{i+1+W}, u_{i+2+W}, \dots . \infty \quad (6.2)$$

where u_i is the most recent item and u_W is the oldest item of the sliding window. At every instance i with the arrival of a new item, the oldest item will be displaced from the sliding window. Flow chart of the cluster head data stream model is shown in Figure 6.3.

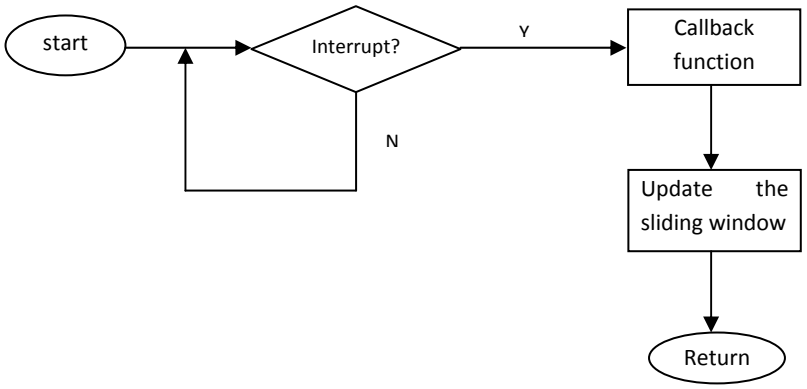


Figure 6.3: Flow chart of the cluster head data stream source process.

The cluster head data stream source model is implemented in TrueTime. The model subscribes to receive an interrupt upon over-the-air data packet arrival event using the TrueTime functions; ttCreateHandler('nw_handler', priority, 'codefcn') and ttAttachNetworkHandler('nw_handler'). The function ttCreateHandler('nw_handler', priority, 'codefcn') creates a TrueTime interrupt handler named ‘nw_handler’ with a priority ‘priority’ and callback function ‘codeFcn’. The function ttAttachNetworkHandler('nw_handler') attaches the interrupt handler to a network interface so that any network event triggers the callback function. The call back function ‘codeFcn’ handles the network event; if the network event is data packet then it pushes the packet received into the sliding window and updates the position of each element of the sliding window. The code snippet in Listing 6.1 below shows the TrueTime modeling of the cluster head data stream source.

```
% subscribe to network event interrupt
ttCreateHandler('nw_handler', 1, 'codeFcn');
ttAttachNetworkHandler('nw_handler');

% an interrupt that notifies msg has been received over a network
function [exectime, data] = codeFcn(seg, data)
global temp

temp = ttGetMsg;
ttTryPost('chReceiveBox', temp);
sensor_data = [sensor_data temp];

end
```

Listing 6.1: The TrueTime model of cluster head data stream source.

6.2. WIRELESS SENSOR NODE MODEL

A wireless sensor node is composed of various specialized hardware and software components. The hardware components are the microcontroller, the sensors, the radio, and the energy sources. The microcontroller is a microprocessor along with some specific purpose peripherals such as timers, analogue to digital converters (ADC), digital to analogue converters (DAC), serial I/O controllers, direct memory access (DMC) etc. The microcontroller is the main host for the application software and firmware that determine its operational purpose. The microcontroller component of the sensor node can be modeled to study its operational behavior through simulation. The sensors are the sources of real world physical quantity measurement data input to the sensor nodes. There are several different types of sensors for probing the same physical quantity. The system modeling here therefore considers a general mathematical model of data input sources described in the previous section. The radio component of the node is a device used to link the network nodes over a wireless radio communication channel. The energy source of a wireless sensor node is usually a battery and occasionally an energy harvesting unit, e.g. a solar cell, is used to support energy demanding applications.

The wireless sensor node hardware components are modeled in TrueTime by the Kernel block, and the Battery block. In this study, TrueTime Kernel block, and the TrueTime Battery block are used to model a sensor node comprising a TI's MSP430 processor, a Chipcon's CC2530 radio transceiver, and a generic dry cell battery, respectively.

The TI's MSP430 based sensor node uses an event-driven programming model with interrupt handlers for handling timer interrupt, network interrupt, etc., in which a single non-terminating task acts as the main program and the event-handling is performed in interrupt handlers. The TrueTime Kernel block implements this programming model. The TrueTime Kernel block also configures a fixed number of analogue/digital inputs and a fixed priority task scheduling policy. Further, the Kernel block models the Chipcon's CC2530 transceiver, initializes the send/receive buffers and network event handlers used by the transceiver. Figures 6.4, 6.5, and 6.6 show the TrueTime model of the wireless sensor node model, cluster head node model and Sink

node model respectively. The energy source of the node is modeled by the TrueTime Battery block. The initial energy available to each node at the start of the system is provided by the Battery block to the Kernel Block through the energy input port of the kernel. The sink node is interfaced to the Base station PC or Laptop, therefore it is considered to be supplied from mains AC.

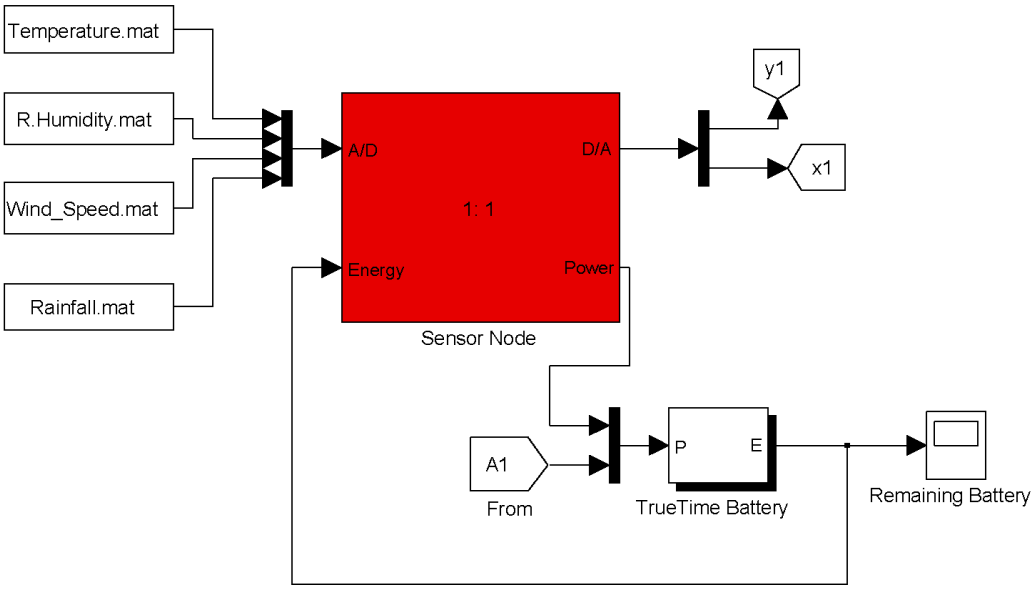


Figure 6.4: The TrueTime wireless sensor node model.

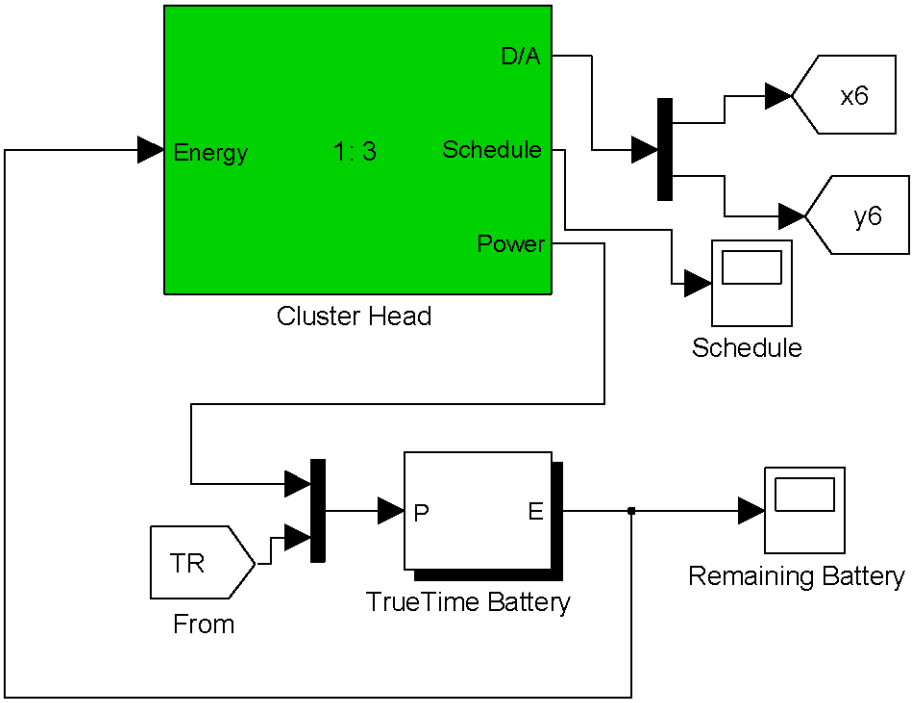


Figure 6.5: The TrueTime cluster head node model.

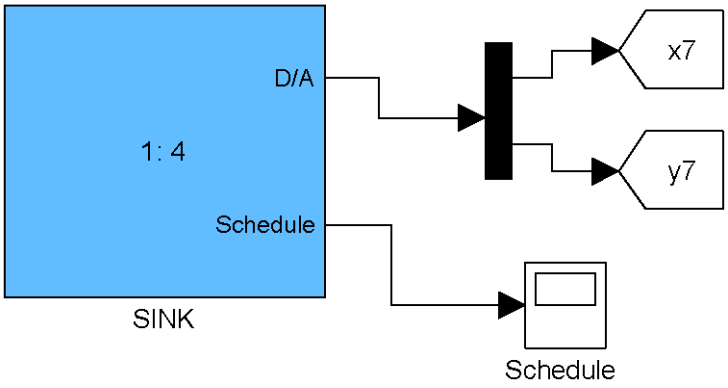


Figure 6.6: The TrueTime sink node model.

6.3. THE WIRELESS NETWORK MODEL

The wireless network linking the sensor nodes for the distributed incremental data stream mining is based on the ZigBee wireless network protocol. ZigBee specifies a high level communication protocol using small, low-power digital radios based on the IEEE 802.15.4 standard for low-rate wireless personal area networks (LR-WPAN). The wireless network model for the purpose of this application is built based on the TrueTime ZigBee model. The wireless network model linking the sensor nodes therefore includes definitions for the CSMA/CA medium access protocol based ad-hoc wireless network, isotropic antenna, half-duplex communication, interference from other nodes, and signal path-loss ($1/d^{a}$), where d is distance and a is path-loss exponent.

The true location of the nodes is specified through the x and y inputs of the model to take into account the path-loss of the radio signal. Communication power drain of individual nodes is also specified through the wireless network model port p. The TrueTime Wireless Network block is shown in Figure 6.7 below.

The Higher layer communication protocol, specifying the wireless network architecture suitable for the distributed incremental data stream mining application, is built on top of the TrueTime IEEE 802.15.4 protocol. The higher layer communication protocol organizes the network into a two-tiered architecture where the first tier forms clusters of sensor nodes in a simple star network and the second tier consists of the cluster heads forming a mesh network. In each cluster of the first tier there is a prefixed node serving as the cluster head. In this architecture, a single sink node serves as the coordinator of the whole network. The Data packets generated at the cluster members are always directed to the respective cluster heads; however, cluster head generated packets are directed to their members or to the sink or both. The data or control packets from the sink are always directed to the cluster heads, which forwards them to the member nodes. Table 6.1 below shows the model parameter settings.

Table 6.1: The TrueTime wireless network parameter settings.

No	Parameter	Value	no	Parameter	value
1	Network type	IEEE802.15.4/ZigBee	**6**	Receiver signal threshold	-90 dBm
2	No. of nodes	226	**7**	Path-loss exponent	2.1
3	Data rate	250000 Kbit/s	**8**	ACK timeout	.0003 ms
4	Minimum frame size	256 bit/s	**9**	Retry limit	5
5	Transmit power	0 dBm	**10**	Error coding threshold	.05

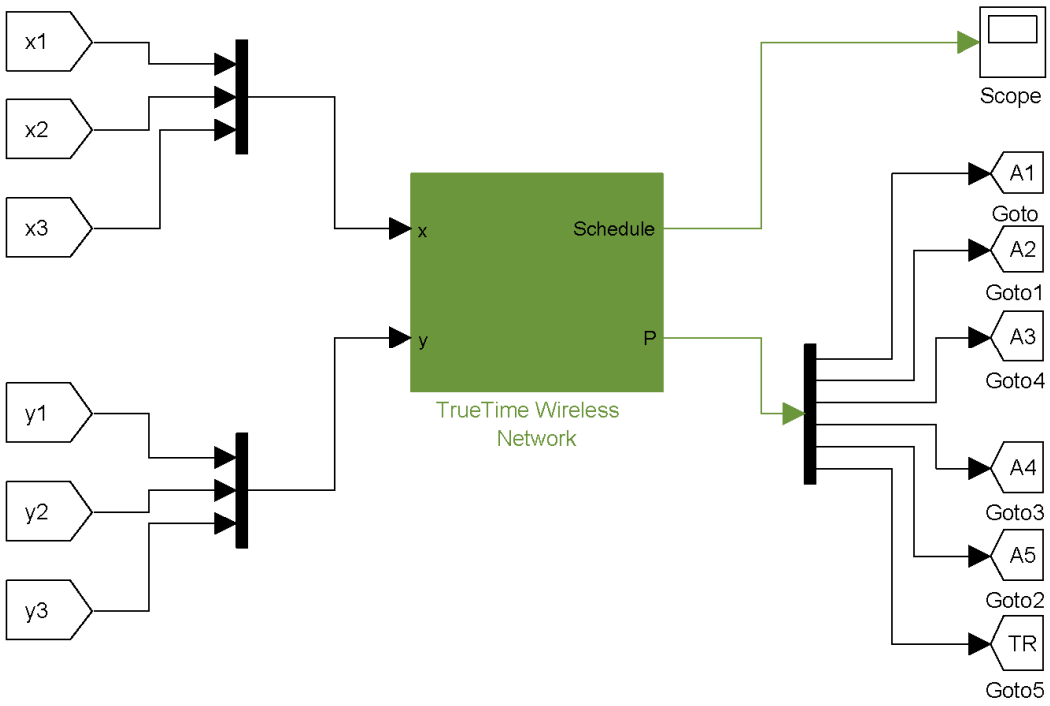


Figure 6.7: The TrueTime Wireless Network Model.

6.4. DATA STREAM MINING TASK MODELS

The distributed incremental data stream mining task is approached through the collaboration of three distinct sub-tasks running on different network nodes; namely, the sensor nodes sub-task, the cluster heads sub-task, and the sink sub-task. These sub-tasks share information over the wireless network to cooperatively implement the data stream mining task utilizing the limited resource available at individual nodes. The

MATLAB/TrueTime implementation of the functions that model these sub-tasks will be described in this section.

We assume that each sensor node sub-task running within a senor node only communicates with the cluster head sub-task running within the cluster head of its host sensor node. The cluster head sub-tasks communicate either to their member node's sub-tasks or to the sink sub-task.

6.4.1. Sensor Nodes sub-task Model

The sensor node sub-task first initializes the host kernel, constants (deviation threshold,Dt), sensors (data stream sources), mail boxes, stream buffer, radio transceiver, radio receive/send buffers, and timers. It then subscribes to network and timer interrupts, defines the handlers of the interrupts, and enters themain programme.

Let u_1 be the first instance of the current data stream of a sensor node. The main programme of the sensor sub-task starts by sending this instance of the current stream to the cluster head sub-task and enters low-power model till it receives a local cluster structure, c_i. When it receives the local cluster structure, the sensor node sub-task starts a stream sampling timer and enters low-power mode until it is woken up by either a timer interrupt or a network interrupt to which is has subscribed. If it is woken up by the timer interrupt then it will call the timer interrupt handler and return back to the low-power mode. Else if it is woken up by the network interrupt then it will call the network interrupt handler and return back to the low-power mode. Figure 6.8 shows the sensor node sub-task model flowcharts.

The timer interrupt handler samples the sensors at a specified rate and records the current instance u_i into the data stream buffer and computes the current instance deviation, $d_{(u_i,c_i)}$ between the current instance, u_i and the local cluster structure, c_i as (6.3).

$$d_{(u_c-c_i)} = \|u_c - c_i\|_D \tag{6.3}$$

where || || is Euclidean metric distance between the two instances, and D is the dimension of the two instances. The dimensions of the two instances must be same.

The handler then determines if the current instance deviation, $d_{(u_i,c_i)}$ is greater than the deviation threshold, Dt as (6.4):

$$d_{(u_i,c_i)} \leq Dt \tag{6.4}$$

If (6.4) returns False (1) then the handler writes the instance, u_i to the send buffer and schedules it for transmission. Upon successful transmission or if (6.4) returns True (0), the handler returns from interrupt. The timer interrupt handler model script is shown in Listing 6.2.

The network interrupt handler upon call gets the message from receive buffer, checks if the message is a non-empty data packet, if so, sets the flag “rcv”, and posts the message to the mailbox of the node. The handler then updates the local structure,c_i, with the received data from the mailbox and starts periodic timer. Listing 6.3 and Listing 6.4 show the network interrupt notifier and the network interrupt handler model scripts, respectively.

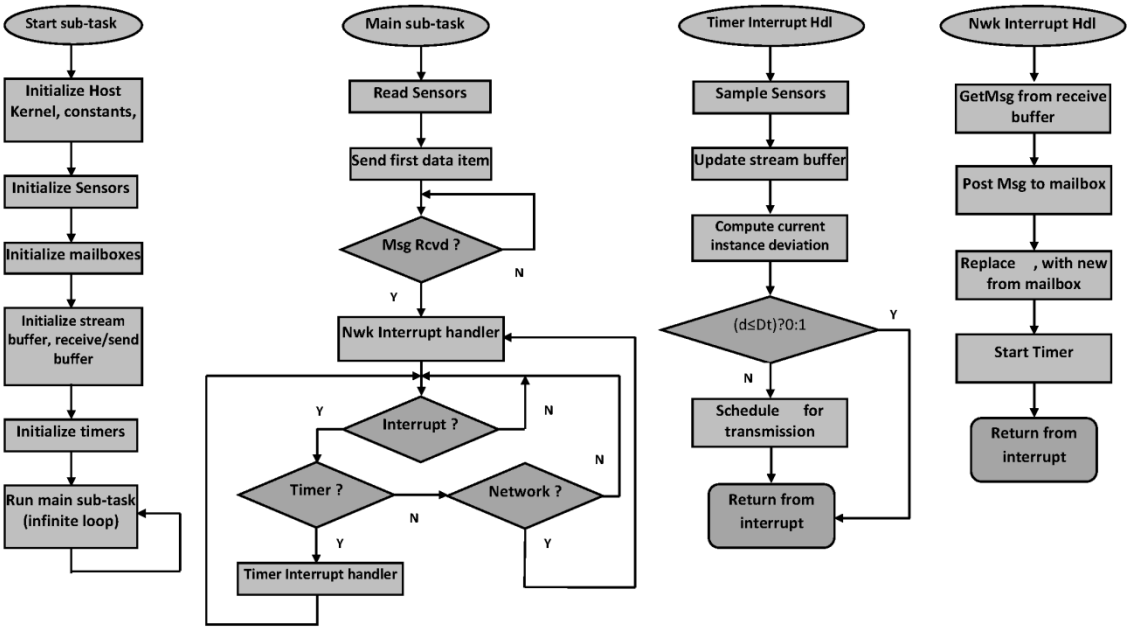


Figure 6.8: The sensor node sub-task model flowcharts.

```
%Timer interrupt handler
function [exectime, data] = TimerIntrpt(seg, data)
global indx index inx indxx u[] d Dt
index + 1;
switch seg,

  case 1,
   u(indx,1) = ttAnalogIn(1);
   u(indx,2) = ttAnalogIn(2);
   u(indx,3) = ttAnalogIn(3);
   u(indx,4) = ttAnalogIn(4);
   exectime = 0.0001;

  case 2,   %Keep the sensor_stream updated periodically
   inx = 2
   if index <= 5
    sensor_stream(index) = u(index);
```

```
    else
     for indxx = inx:index -1
      sensor_stream(indxx -1) = sensor_stream(indxx);
     end
     sensor_stream(indxx) = u(index)
     inx +1;
    end
    exectime = 0.00015;

  case 3,
    d = [sqrt(((u(indx,1) - c(1,1))^2) + ...
      ((u(indx,2) - c(1,2))^2) + ...
      ((u(indx,3) - c(1,3))^2) + ...
      ((u(indx,4) - c(1,4))^2)];
    exectime = 0.0008;

  case 4,
    if d > Dt
     ttCreateJob('snsendTask');
    end
    exectime = -1;
end
```

Listing 6.2: Timer interrupt handler model script for sensor node sub-task.

```
%Network interrupt notifier
function [exectime, data] = NwkIntrpt(seg, data)

global temp
```

```
switch seg,

  case 1,
  %Call the network interrupt handling task
  ttCreateJob('NwkIntrptHndlr');
  exectime = -1;

end
```

Listing 6.3: Network interrupt notifier model script for sensor node sub-task.

```
%Network interrupt handler
function [exectime, data] = NwkIntrptHndlr(seg, data)
global temp temp2 strt_timer

switch seg,

  case 1,
  temp = ttGetMsg;
  if(isfeild(temp, 'data'))
    if(isempty(temp.msg.data))
      else
      ttTryPost('snReceiveBox', temp);
      rcv =1;
    end
  end
  exectime = 0.0018;

  case 2,
  temp2 = ttTryFetch('snReceiveBox');
  c1 = temp2.msg.data; %set received data as local cluster structure
```

```
    ttCreateJob('TimerstartTask'); % start periodic timer
    exectime = -1;
end
```

Listing 6.4: Network interrupt handler model script for sensor node sub-task.

6.4.2. Cluster Heads sub-task Model

The cluster head sub-task initializes the host kernel, constants, counters, mailboxes, sliding window buffer, receive/send buffer, and tasks. It then subscribes to the network interrupt handler and enters the main programme loop. In the main programme loop, counter1 is set to zero and counter2 is set to the number of member nodes. The main programme then enters sleep mode or low-power mode until a network interrupt wakes it up, upon which it calls the network interrupt handler and returns back to sleep mode. The network interrupt handler services most of the cluster head sub-task functions; transfers the received message from the receive buffer to the node's receive mailbox after inspecting the received message integrity, increments counter1 on every successful message reception, updates sliding window content with the received data, and evaluates a conditional statement, counter1 == counter2'. If the evaluation results in false then do nothing and return from interrupt. Otherwise if the evaluation results in true, which means a message from all the member nodes has been received, then calls SUBFCM algorithm model code (function), schedule the returned cluster structures for transmission to member nodes and the sink and return from interrupt. The cluster head sub-task model flowcharts are shown in Figure 6.9.

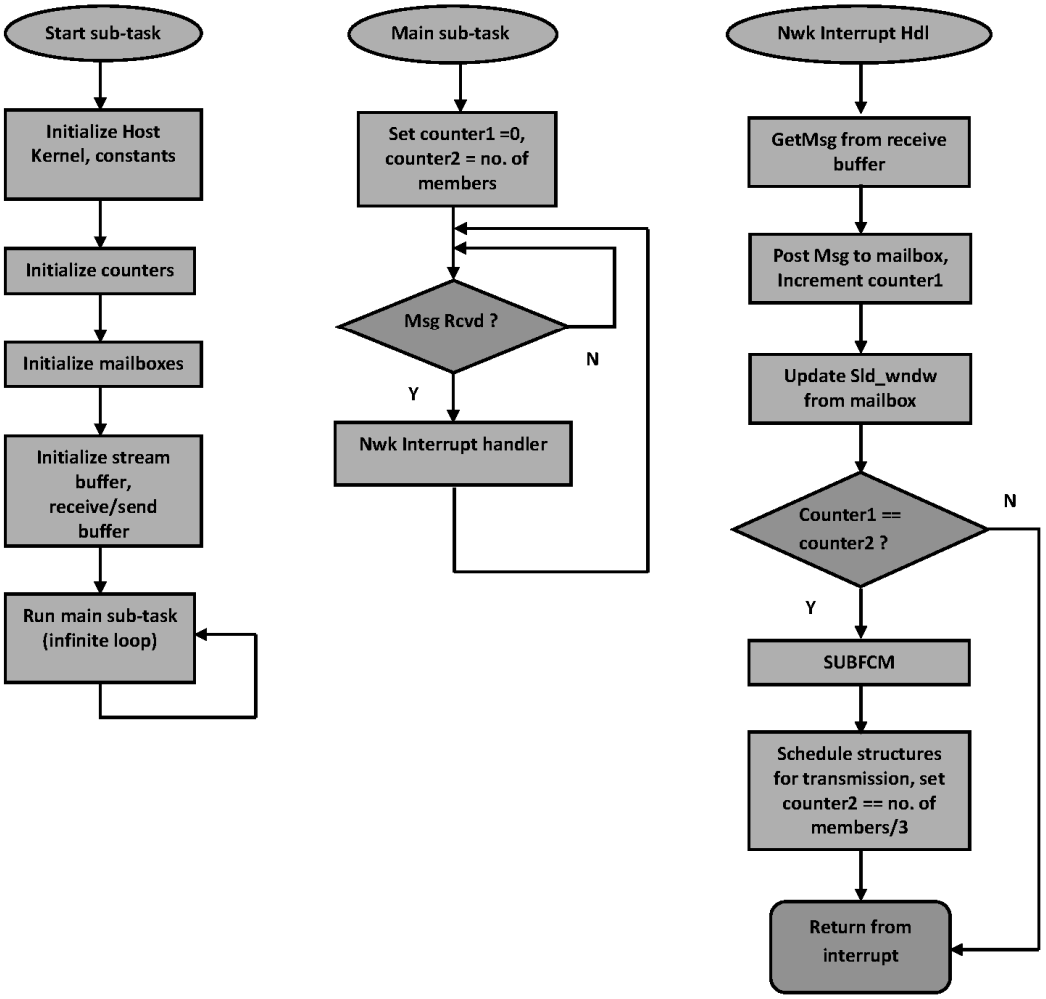


Figure 6.9: The cluster head sub-task model flowcharts.

```
%Network interrupt notifier
function [exectime, data] = NwkIntrpt(seg, data)

global temp

   switch seg,

      case 1,
      %Call the network interrupt handling task
      ttCreateJob('NwkIntrptHndlr2');
      exectime = -1;
```

```
  end
```

Listing 6.5: Network interrupt notifier model script for cluster head sub-task.

```
%Network interrupt handler
function [exectime, data] = NwkIntrptHndlr2(seg, data)
global temp temp2 counter1 indx index inx indxx v nummember
index + 1;

switch seg,

   case 1,
    temp = ttGetMsg;
    if(isfeild(temp, 'data'))
      if(isempty(temp.msg.data))
         else
         ttTryPost('chReceiveBox', temp);
      end
    end
    counter1 = counter1 + 1;
    exectime = 0.0018;

    case 2,
     temp2 = ttTryFetch('chReceiveBox');
     v = temp2.msg.data; %Receive an element sliding   window

     inx = 2
     if index <= nummember
      Sld_wndw(index) = v(index);
     else
```

```
      for indxx = inx:index -1
        Sld_wndw(indxx -1) = Sld_wndw(indxx);
      end
     Sld_wndw(indxx) = v(index)
     inx +1;
   end
  exectime = .00019;

  case 3,
  if counter1 == counter2
    ttCreateJob('SUBFCM_Task');
    ttCreateJob('chsend_Task');
    counter2 = nummember/3;
    else
  end
  exectime = -1;
end
```

Listing 6.6: Network interrupt handler model script for cluster head sub-task.

6.4.3. The Sink sub-task Model

The sink sub-task initializes the host kernel and radio transceiver. It subscribes to network interrupt handler to receive incoming data from cluster heads. It also initializes data transfer to the base station PC. The network interrupt handler caries out firstly, instant data transfer from receiver buffer to the sink mailbox, then from sink mailbox to the base station PC. The resource on the PC can be used to perform complex analysis of the collected information and present the results to the end user in real-time. The results could be stored to a database for historical analysis whenever required.

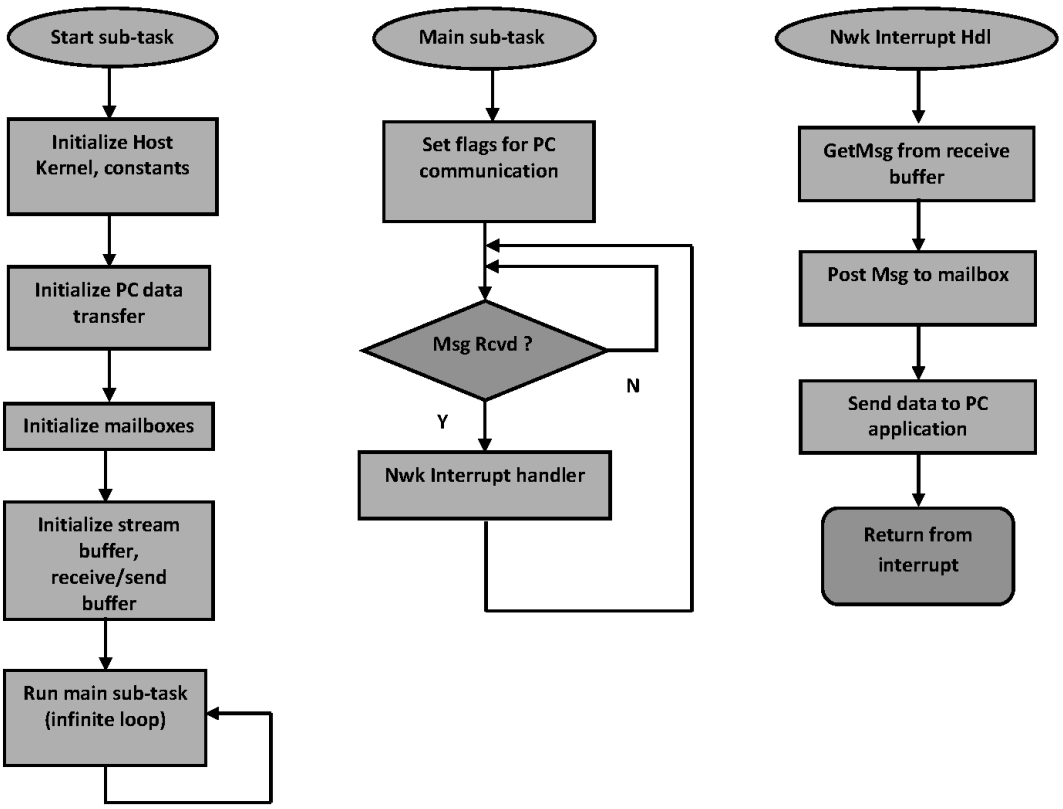


Figure 6.10: The sink sub-task model flowcharts.

```
%Network interrupt handler
function [exectime, data] = NwkIntrptHndlr3(seg, data)
global temp temp2 flag1 flag2


switch seg,

   case 1,
    temp = ttGetMsg;
    if(isfeild(temp, 'data'))
      if(isempty(temp.msg.data))
         else
         ttTryPost('sinkReceiveBox', temp);
```

```
    end
  end
  exectime = 0.0018;

  case 2,
  temp2 = ttTryFetch('sinkReceiveBox');
  exectime = .0001;

  case 3,
  set flag1;
  set flag2;
  exectime = -1;
end
```

Listing 6.7: Network interrupt handler model script for sink sub-task.

6.5. SYSTEM MODEL SIMULATION

In this section, the models implementing different components of the distributed incremental data stream mining WSN described in previous sections are assembled into a complete system model to evaluate its performance through simulation.

The general system model can be assembled as shown in Figure 6.11 where the stream mining code models are embedded into the node models and communication is accomplished through the wireless network model.

A series of simulation setups with varying stream complexities and network parameters are designed to analyze the system performance in terms of cluster quality and validity. Network service qualities for the mining application such as energy consumption, data delivery delay, and packet delivery ratio are also evaluated. An evaluation of the simulation results based on the benchmark systems described below is presented in the results and analysis chapter (Chapter 7).

A centralized flat multi-hop stream clustering WSN architecture (Figure 6.12) and a centralized cluster-based stream clustering WSN architecture (Figure 6.13) with similar scale are built and used as a benchmark to compare the performance of the distributed incremental clustering architecture.

The centralized flat multi-hop stream clustering architecture setup involves nodes taking data from onboard sensors and sending them to a sink node multi-hop. In this setup every node participates in relaying other sensors' data to the sink node besides sending its own data. Clustering is performed at the sink every time a data element from all nodes in the network has arrived and this repeats for subsequent data elements of the stream.

The centralized cluster-based stream clustering setup involves cluster heads collecting data elements from their members and forwarding them to the sink every period. The cluster heads in this setup collect and forward their members' data elements, one element at a time from every member, to the sink and the sink carries out the clustering.

The distributed incremental clustering setup involves cluster heads carrying out local clustering before forwarding their local structures to the sink where the global clustering will be carried out. The difference between this setup with the centralized clustering cluster-based setup is in their internal working models. However, their topology set up is exactly the same.

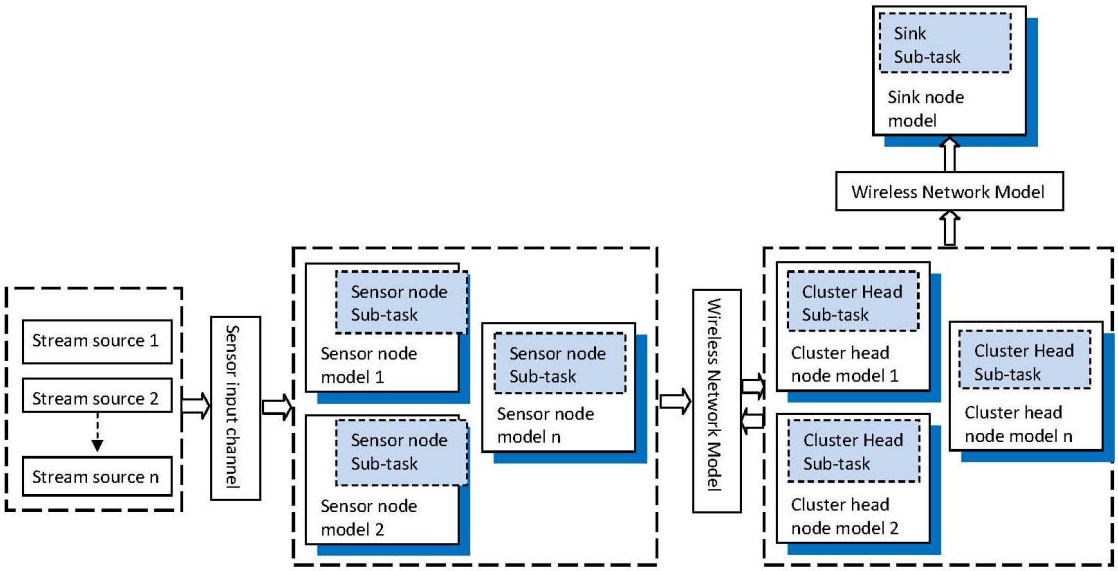


Figure 6.11: The general distributed incremental data stream mining system model.

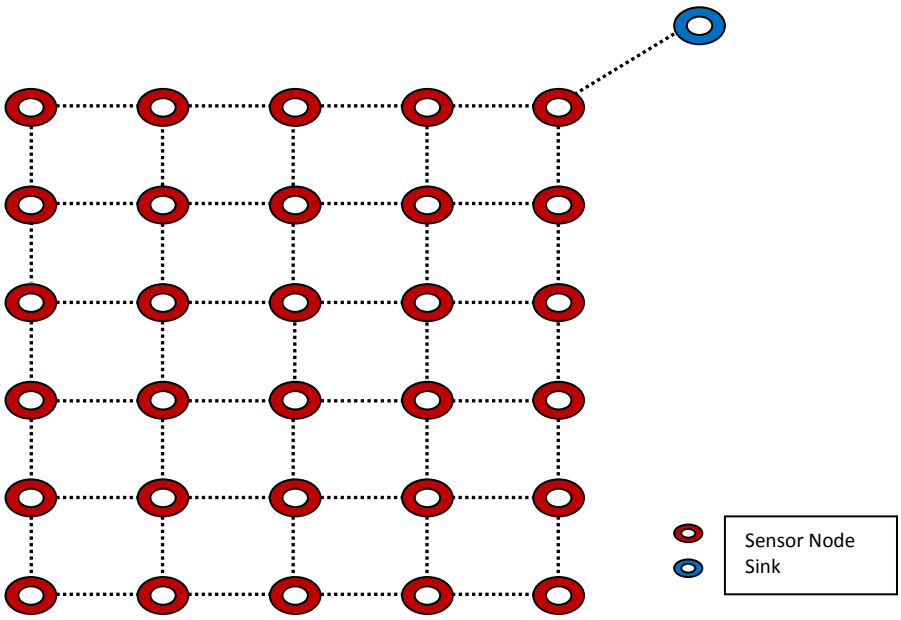


Figure 6.12: Centralized flat multi-hop stream clustering architecture network.

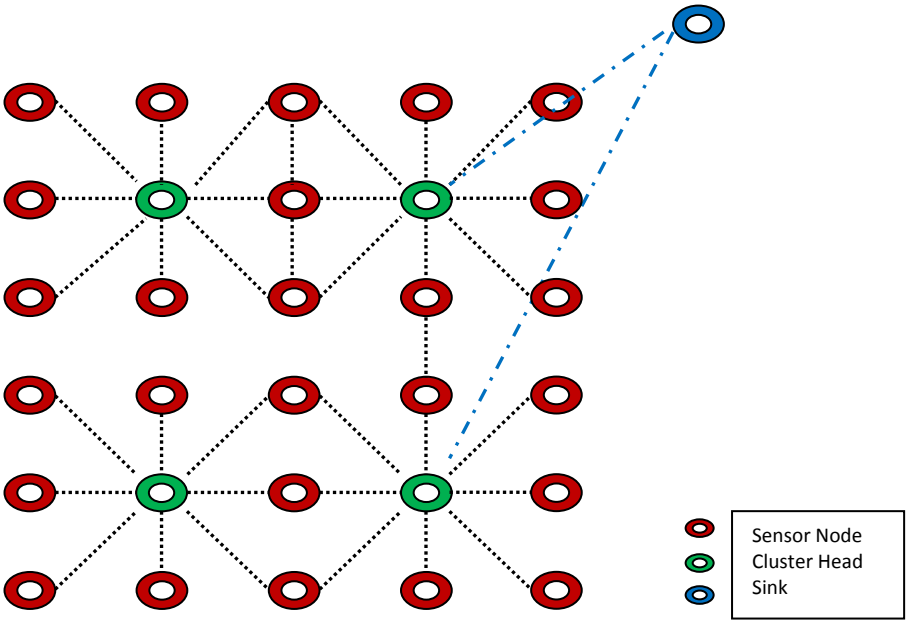


Figure 6.13: Centralized cluster-based stream clustering network.

6.6. DISCUSSION

A WSN model for distributed data stream mining applications is presented. The performance of the system model is evaluated through the TrueTime simulator in the MATLAB environment utilizing the Simulink discrete event simulation engine. Different components of the system are modeled separately and integrated as a whole system model.

The Streaming data is modeled as a combination of real data sets stored in a specific file and a process accessing the data sets one element at a time. The wireless sensor nodes are modeled as sensors, radio transceivers, and energy sources along with their application hosting microcontroller based on TrueTime base models and MATLAB programs. The TurTime ZigBee wireless network protocol model is utilized to model the wireless radio link among the sensor nodes. The distributed data stream mining application is modeled as MATLAB task scripts distributed throughout the network nodes.

Benchmark clustering algorithms and network architectures are built and used in evaluation of the performance of the distributed incremental data stream mining WSN model through simulation.

Chapter 7

7. RESULTS AND ANALYSIS

This chapter will elaborate on the model performance evaluation results gathered from the distributed incremental data stream mining WSN system model simulation. Based on the results and analysis, the model capabilities and limitations will be discussed.

7.1. SIMULATION RESULTS

The evaluation of the distributed incremental data stream mining WSN is performed through simulation using the TrueTime simulator 2.0 Beta 6. The results presented in this chapter regarding performance evaluation of the application and the network services are based on averages of 10 to 15 simulation runs with 95% confidence limits taking realistic parameters obtained from experimental tests.

7.2. SIMULATION SETUP

The simulations were implemented in a Simulink environment using the basic models in the TrueTime block library. The standard Simulink library blocks are utilized for all other blocks necessary for the modeling. All node functionalities including the mining algorithm within the nodes were programmed in MATLAB scripts. Cluster based

network topologies shown in Chapter 6 Figures 6.12, 6.13, and 6.14 are used. All simulations involve only stationary nodes. Even though node mobility is supported in TrueTime, node mobility is not considered in this research.

For the purpose of evaluating the distributed incremental data stream mining WSN system we consider two parts: evaluation of data stream mining quality and evaluation of network service quality.

7.3. SIMULATION ENVIRONMENT AND DATASETS

To evaluate the performance of distributed incremental data stream mining WSN, we use real datasets on a PC with 3.20GHz i5 core CPU and 4GB memory running Windows XP. The TrueTime simulation programmes are implemented using MATLAB M-file scripts.

The real dataset that we used contains 10 minutely weather observations of 200 days at Sydney, Australia, recorded from June 2011 to January 2012 [162]. Each day is regarded as a data stream and each stream has 144 points (24 x 60/10). Each data stream instance (stream object) consists of temperature, relative humidity, wind speed, and rainfall. The data streams are known to represent three levels of forest fire danger ratings (Low, Moderate and High) on the McArthur Fire Danger Index (FDI) scale [163].

7.4. SIMULATION AND ANALYSIS

Initially, we evaluate the cluster quality of the distributed incremental data stream mining WSN model using benchmark standard clustering algorithms; the K-Means and the FCM. Using the same data stream sets, we vary the stream dimension (number of data features), and stream periods to investigate the nature and the complexity of the streams that the distributed incremental data stream mining WSN model can handle. The performance of the model compared to the benchmark models should highlight the validity of the model under the simulated conditions and guidelines for optimal system design.

The first set of simulations considers the system performance on different stream dimensionality (i.e. stream elements with different number of feature space) and stream rates. Stream sets with single to four dimensions are used to investigate the mining performance with respect to the benchmark models. Sources generating streams as slow as every minute to as fast as every second are used to investigate the effect on cluster quality and validity.

The second set of simulations investigates the effect of network architectural variances on the mining performance. The variables considered in this simulation are cluster density, local model drift threshold (i.e. the maximum amount of local model drift before the system starts updating the global model), and non-uniform clusters.

The third and final sets of simulations consider the impact of the distributed incremental data stream mining WSN model by evaluating the network services behavior. This evaluation considers the average energy consumption, the average data delivery delay, and the packet delivery ratio.

7.5. SIMULATIONS 1

The setup for this simulation consists of a network of 200 sensor nodes, 25 cluster heads, and a sink node. The network is organized under 25 uniform clusters of 8 nodes each. Each node is configured to transmit an element of its stream at a period of 5 seconds. The other network parameters are as in Table 6.1 of chapter 6. The data stream source for this simulation consists of the weather data stream described in section 7.3. The first sets of simulations take only temperature streams and make up single dimensional streams. The 2D, 3D and 4D streams are formed in the same manner taking two, three and four features from the original data stream consisting of temperature, relative humidity, wind speed, and rainfall features, respectively.

7.5.1. One-Dimensional (1D) Stream Analysis

Figure 7.1(a) shows a snapshot of the first elements of the eight member nodes to have arrived at the first cluster head (CH1). This makes up the first sliding window of

the first cluster head stream on which the SUBFCM algorithm runs and extracts local cluster models. Figure 7.1 (b-f) show similar snapshots of cluster heads two to six (CH2 to CH6).

Therefore, there is a stream of 200 elements distributed within the network (eight stream elements at each of the 25 cluster heads) at each simulation step. The first stream elements from all clusters are combined and shown in Figure 7.2 below. The combined stream set at each simulation step is passed on to the benchmark clustering algorithms to extract the ideal reference clusters to compare to the clusters from the distributed model.

Global cluster models are computed at the sink by re-clustering the local cluster models obtained from all cluster heads at every simulation step. Figure 7.3 shows cluster centers obtained from the distributed model and the reference cluster centers during 144 simulation steps for the single dimensional data stream.

We captured the stream sets at each simulation step (total of 144 steps) for running K-Means and FCM systems offline. However, the SUBFCM is run distributed and online. As described before, the datasets are known to consist of elements belonging to three classes of Forest Fire danger levels on the McArthur scale [163]. However, in this particular simulation, we only take a single variable (Temperature) from each data stream element to form a single dimensional stream. Therefore, the number of clusters in each step is pre-set to three for K-Means and FCM clustering, whereas for the SUBFCM clustering, the cluster radius variable is calibrated via offline testing on the same datasets and fixed to $r_a = 0.10$ that partitions 97.3% of the stream sets into three clusters.

The Analysis of clusters obtained from the distributed SUBFCM system taking into account 144 simulation steps shows that the temperature cluster centers obtained deviate by 0.42 °C and 0.21 °C on average in reference to the central K-Means and FCM systems, respectively. The maximum cluster centers displacement observed are 0.59 °C and 0.46 °C compared to K-Means and FCM systems, respectively. Figure 7.4 shows the model cluster deviations with reference to K-Means and FCM for

the total 144 simulation steps. The maximum deviation is only 2.8% of the maximum temperature in the stream.

The results indicate that the distributed incremental data stream clustering WSN model can extract clusters comparable in quality to those obtained from batch clustering of data streams gathered at a central location using standard K-Means and FCM algorithms. A deviation of 2.8% of the maximum value is tolerable by several applications given the added advantages of distributed clustering.

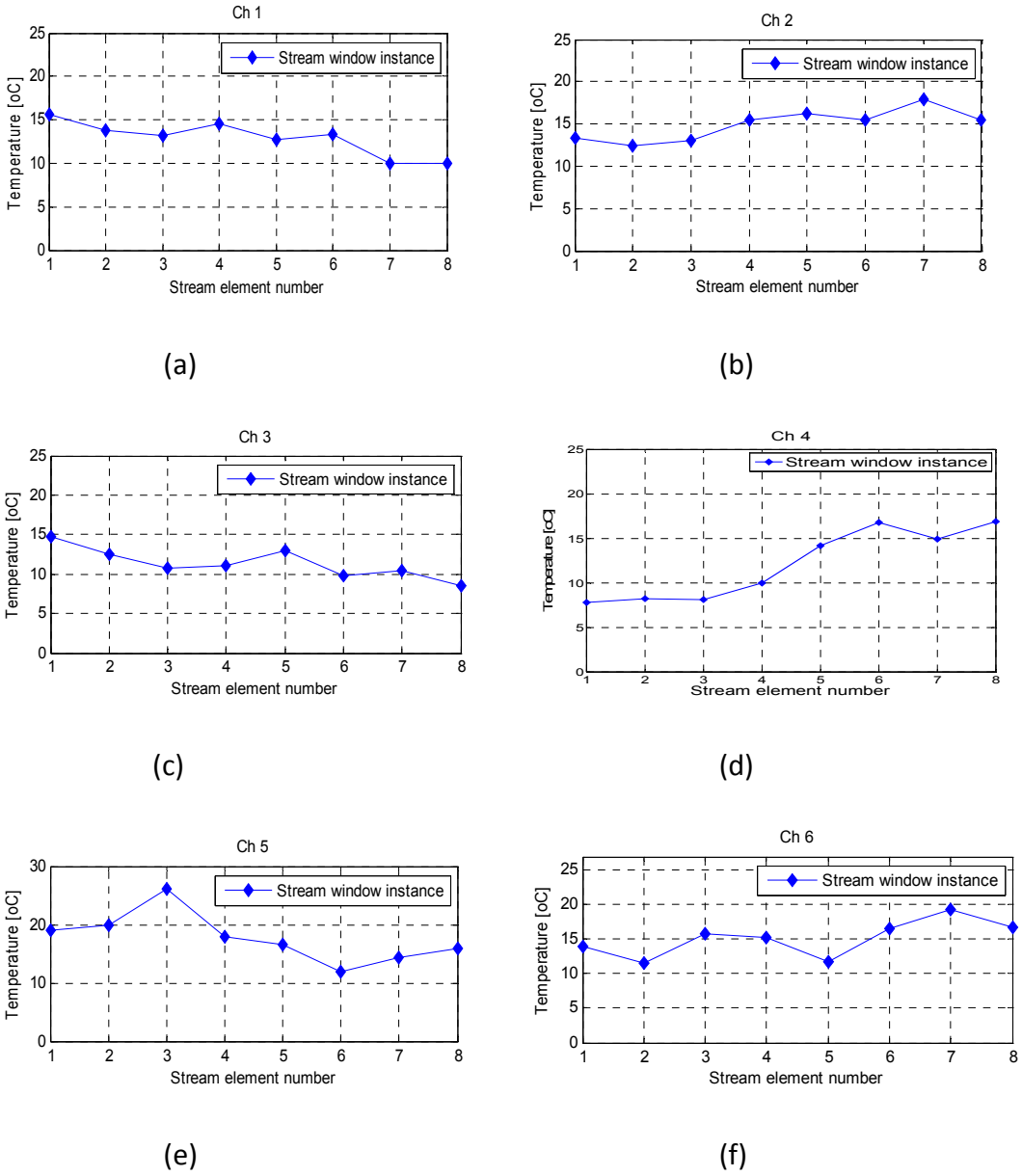


Figure 7.1: Cluster heads first sliding window snapshots.

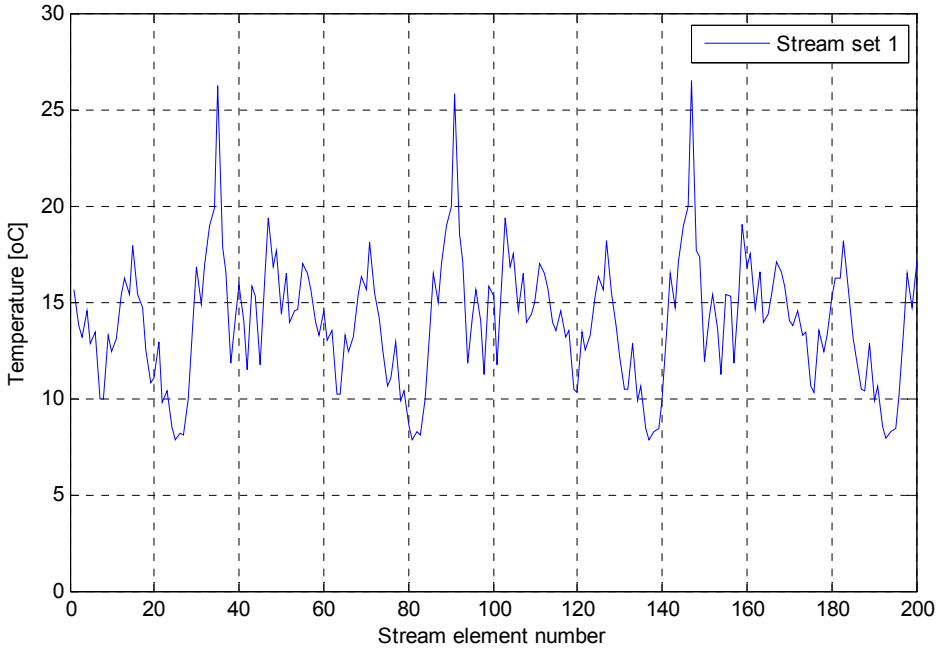


Figure 7.2: The sliding windows combined to form the first stream set.

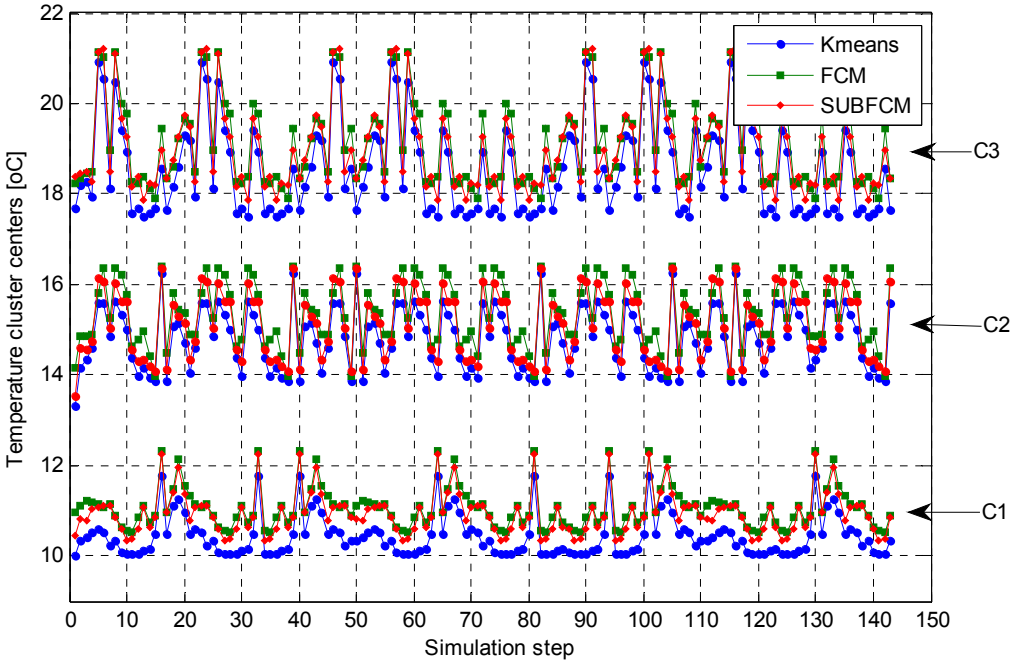


Figure 7.3: Cluster centers extracted during 144 simulation steps.

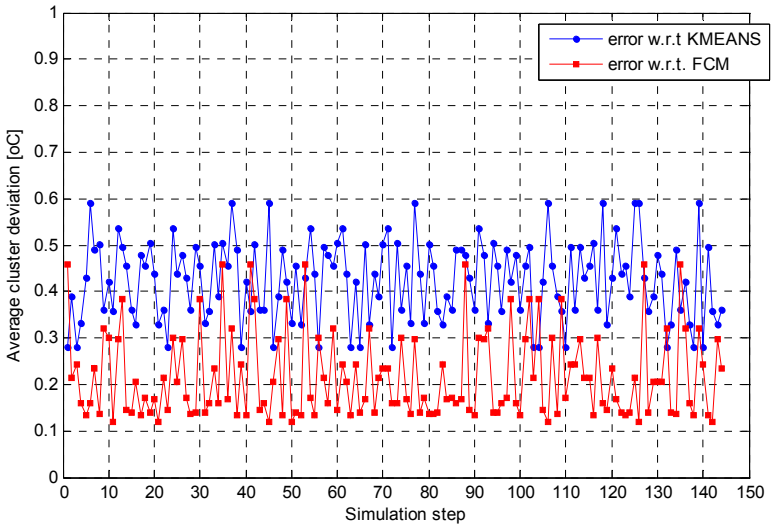


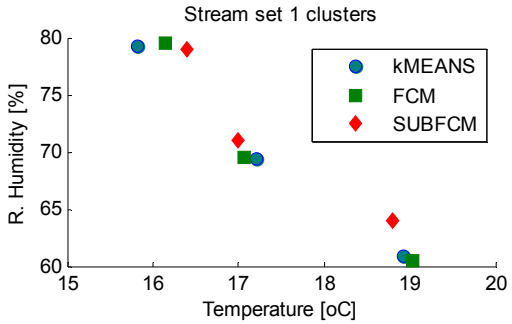
Figure 7.4: Average cluster deviations.

7.5.2. Two-Dimensional (2D) Stream Analysis

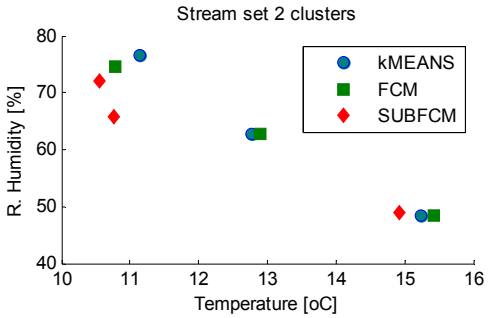
This simulation involves similar settings and the same data stream source as the previous section; however, two features (temperature and relative humidity) from each stream element are considered to make up 2D data streams. Figure 7.5 (a-f) shows a snapshot of the first sliding windows of cluster head one to cluster head six. The first stream set, taken at the first simulation step, along with the cluster centers obtained using the distributed SUBFCM system and reference systems K-Means and FCM are shown in Figure 7.6 below. The cluster radius for our distributed system for the 2D data stream case is set to $r_a = 0.15$ after calibrating offline using the same datasets. In 98% of the 144 steps, the distributed system produced three distinct clusters similar to the K-Means and FCM clusters. Figure 7.7 shows the average cluster centers deviation of distributed SUBFCM compared to the K-Means system. The average cluster centers deviation with respect to the FCM system is shown in Figure 7.8.

Analysis of the results obtained shows that the distributed system cluster centers deviate by 3.86% and 1.46% of the cluster radius on average with respect to the K-Means and FCM cluster centers respectively. The maximum cluster deviation

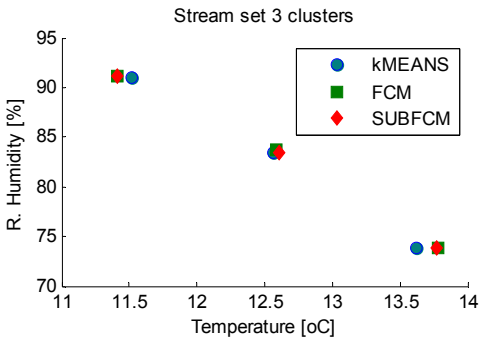
observed in this simulation is 13.22% with respect to K-Means and 5.16% with respect to FCM.



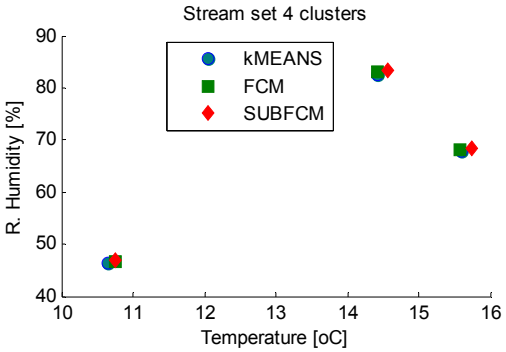
(a)



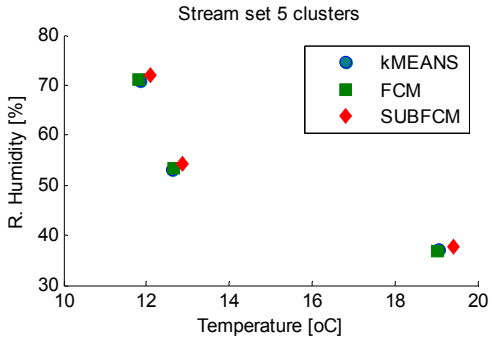
(b)



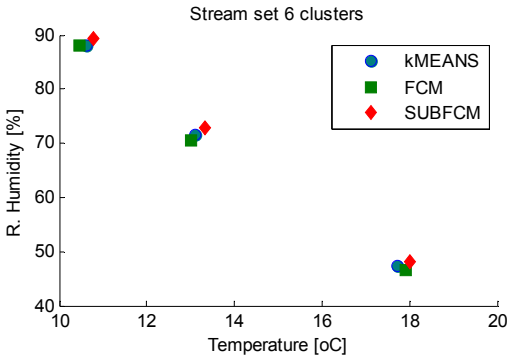
(c)



(d)



(e)



(f)

Figure 7.5: Cluster heads first sliding windows snapshot.

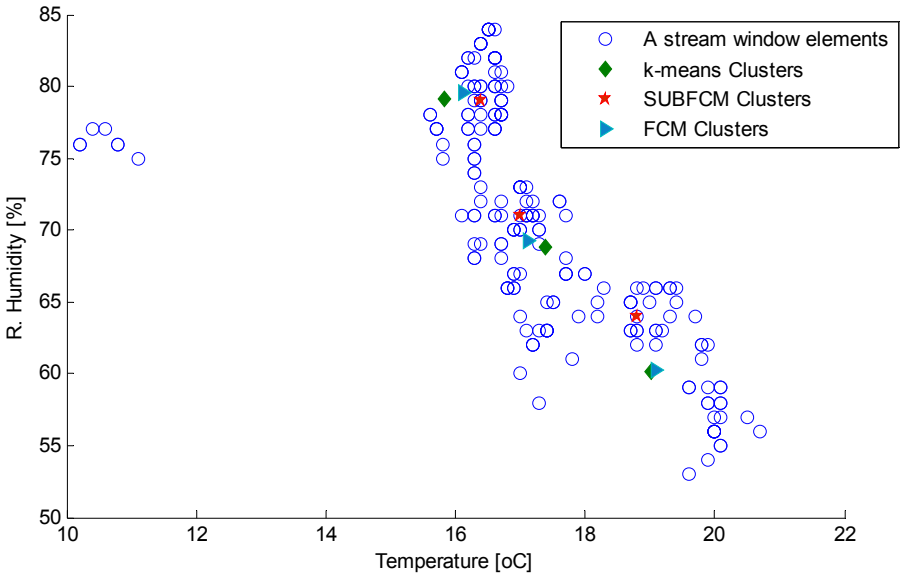


Figure 7.6: Stream sets and clusters from first simulation step.

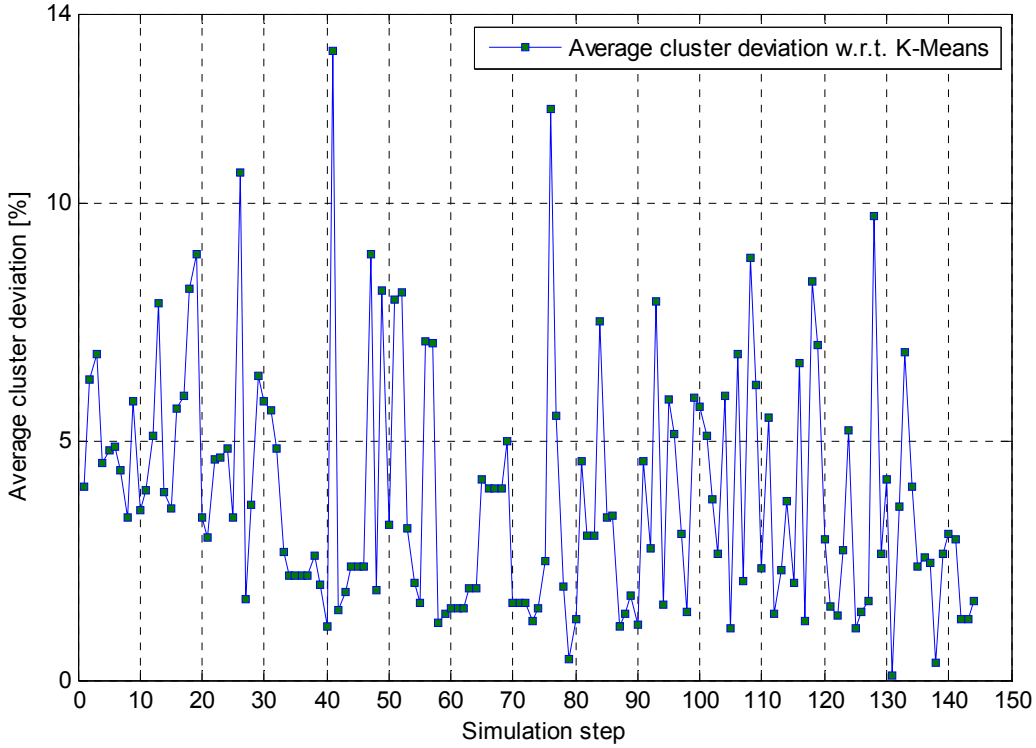


Figure 7.7: Average cluster deviations with respect to K-Means for 2D streams.

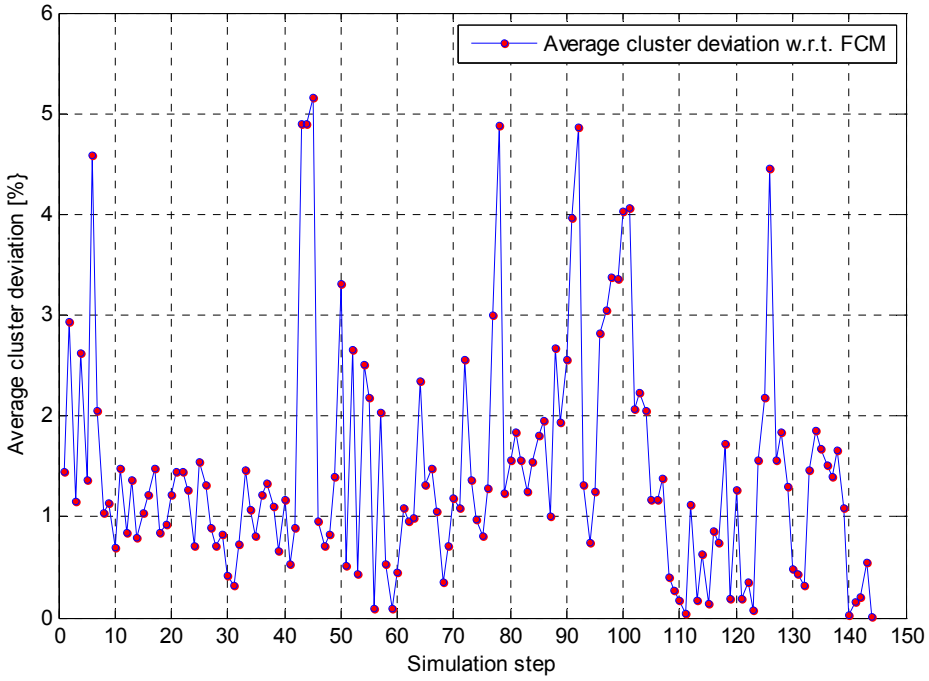
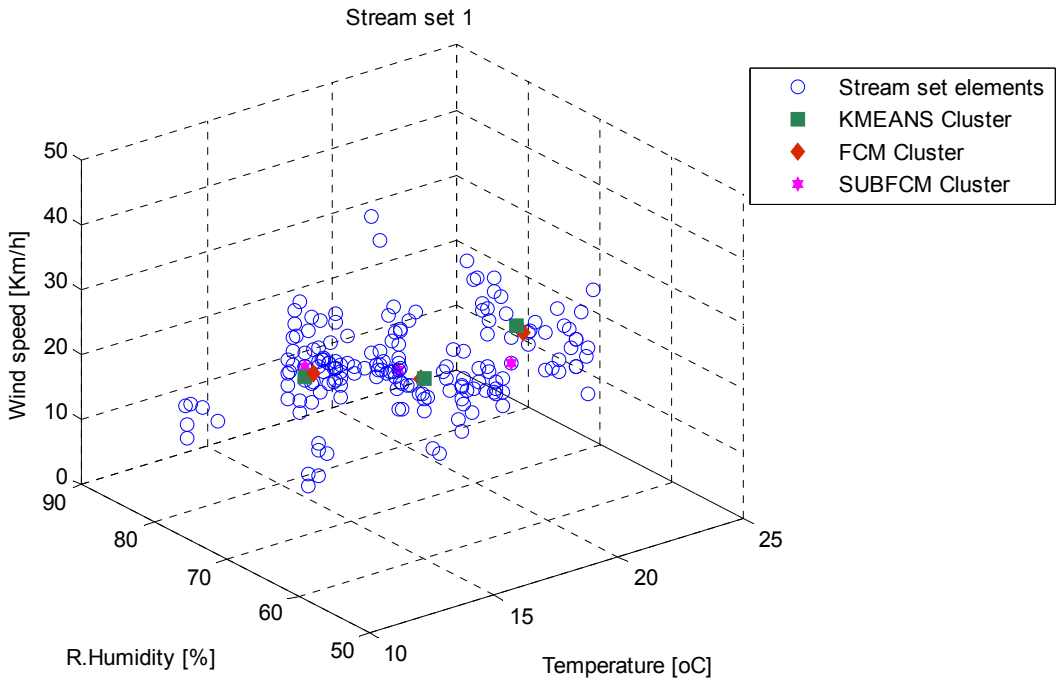


Figure 7.8: Average cluster deviations with respect to FCM for 2D streams.

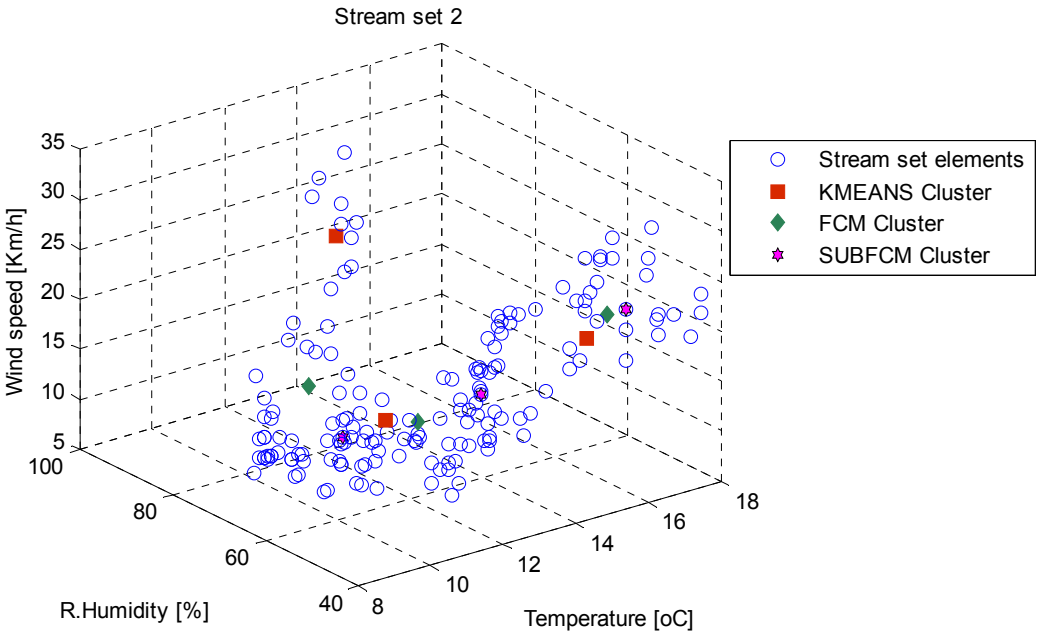
7.5.3. Three-Dimensional (3D) Stream Analysis

Three-dimensional stream analysis simulation considers three variables (temperature, relative humidity, and wind speed) from the same data sources used in previous simulations for 1D and 2D cases. The cluster radius for the distributed clustering is calibrated to $r_a = 0.065$. Other settings are as in previous simulations. The Snapshot of the first sliding windows of cluster head one to cluster head three are shown in Figure 7.9 (a-c). The second stream set in Figure 7.9 (b) shows the maximum cluster centers deviation.

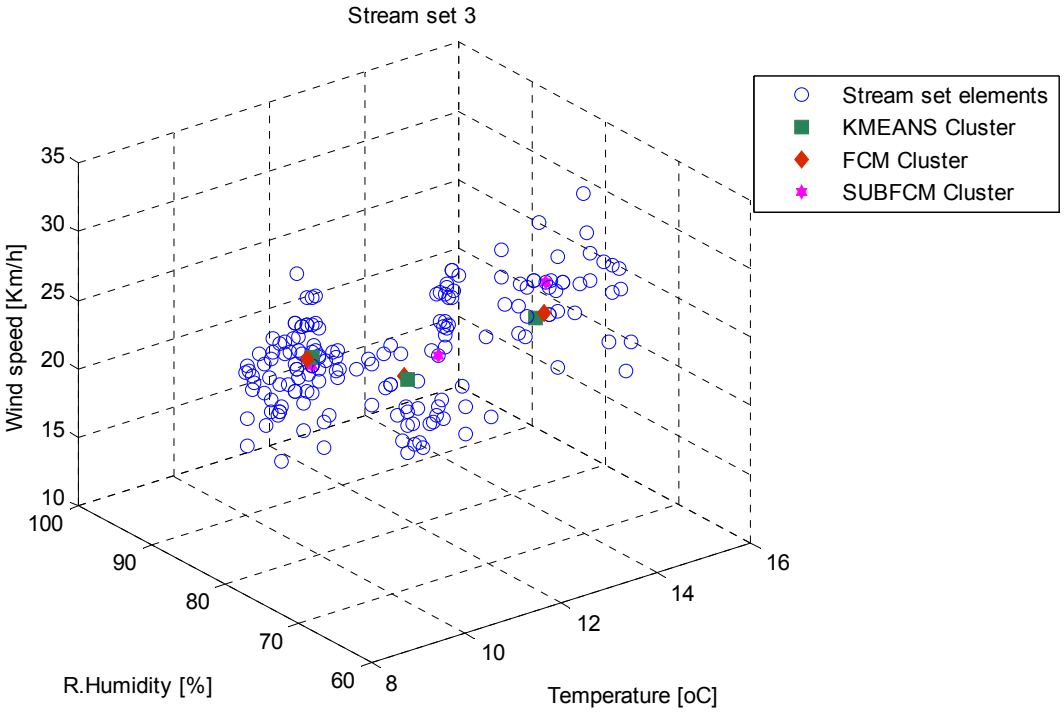
Simulation results analysis shows that the average cluster deviations of the distributed SUBFCM system is 11.63% and 6.05% compared to the K-Means and FCM systems respectively. Maximum observed cluster deviation is 15.52% compared to K-Means system. Figure 7.10 and Figure 7.11 show average cluster deviations for the 144 simulation runs with respect to K-Means and FCM respectively.



(a)



(b)



(c)

Figure 7.9: Cluster heads first sliding windows snapshot.

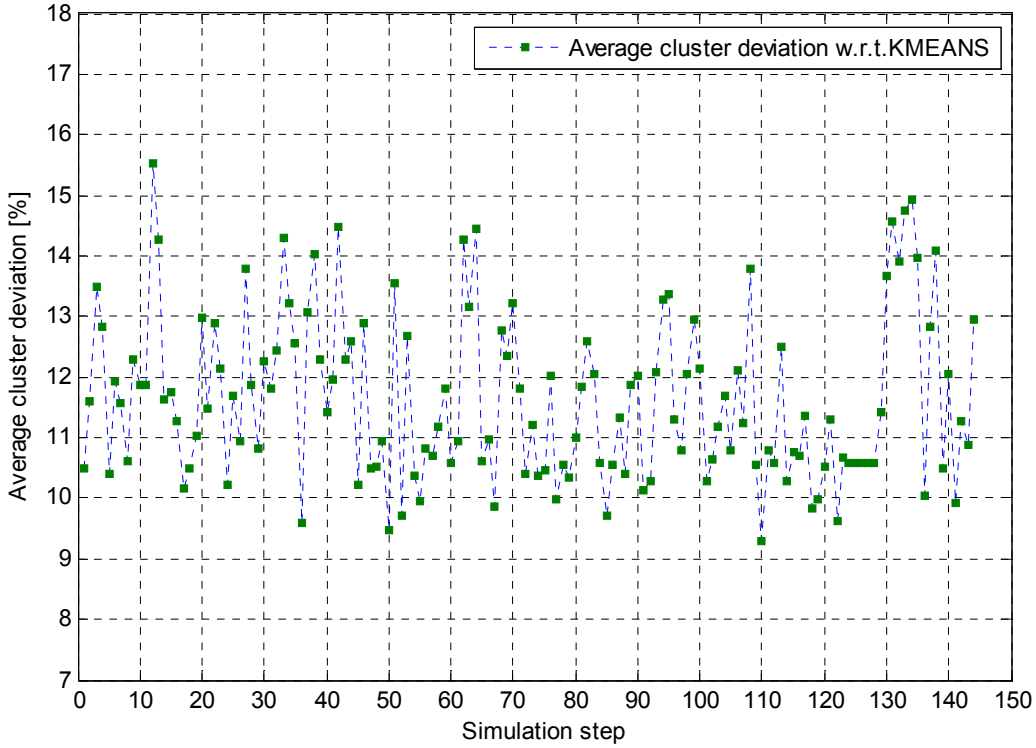


Figure 7.10: Average cluster deviations with respect to K-Means for 3D streams.

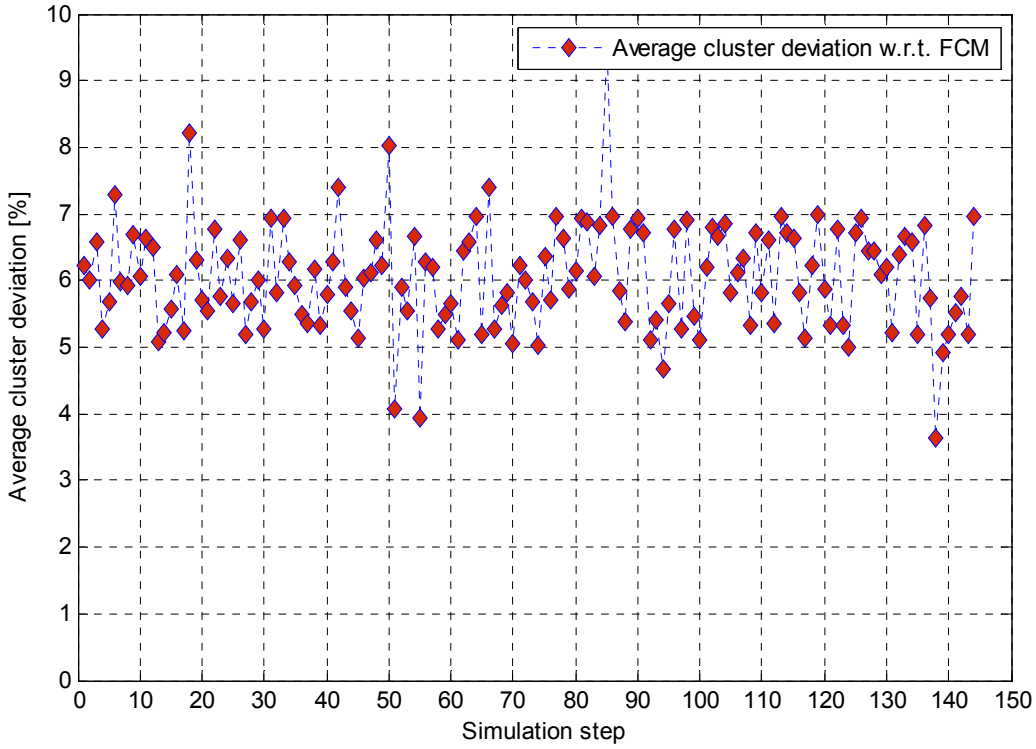


Figure 7.11: Average cluster deviations with respect to FCM for 3D streams.

7.5.4. Four-Dimensional (4D) Stream Analysis

Four-dimensional stream analysis simulation considers all the four variables (temperature, relative humidity, wind speed, rainfall) in the data source. The cluster radius for the distributed clustering is calibrated to $r_a = 0.09$. Other settings are as in previous simulations.

Simulation results analysis shows that the average cluster deviation of the distributed SUBFCM system is 7.74% and 6.29% compared to the K-Means and FCM systems respectively. Maximum observed cluster deviation is 14.6% compared to K-Means system. Figure 7.12 and Figure 7.13 show average cluster deviations for the 144 simulation runs with respect to K-Means and FCM respectively.

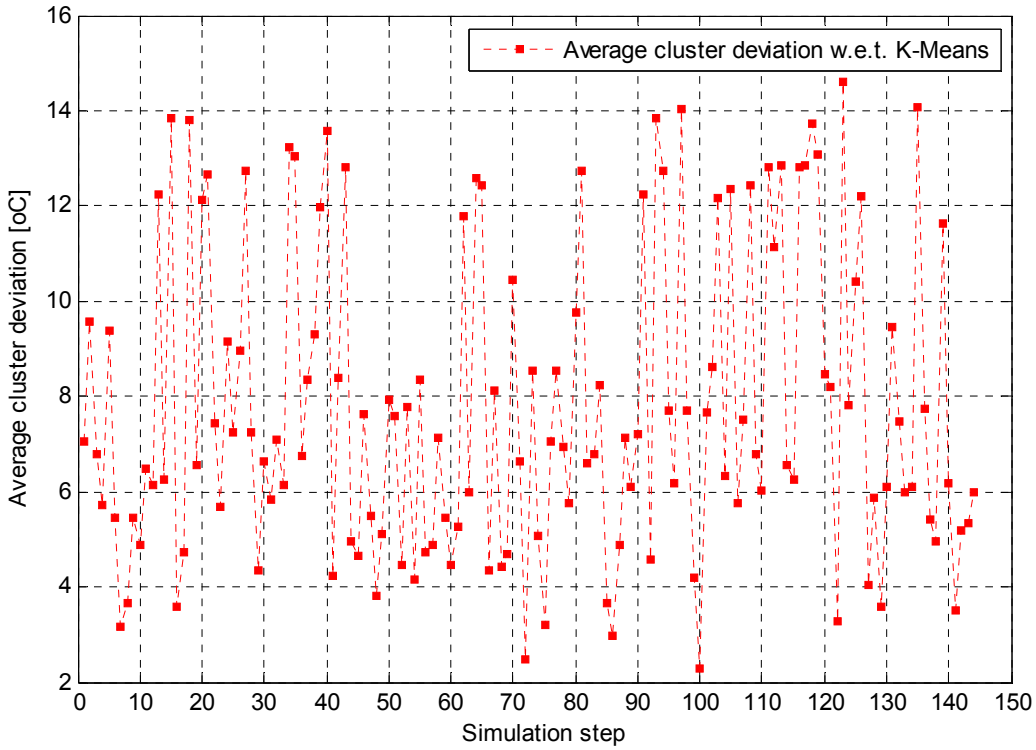


Figure 7.12: Average cluster deviations with respect to K-Means for 4D streams.

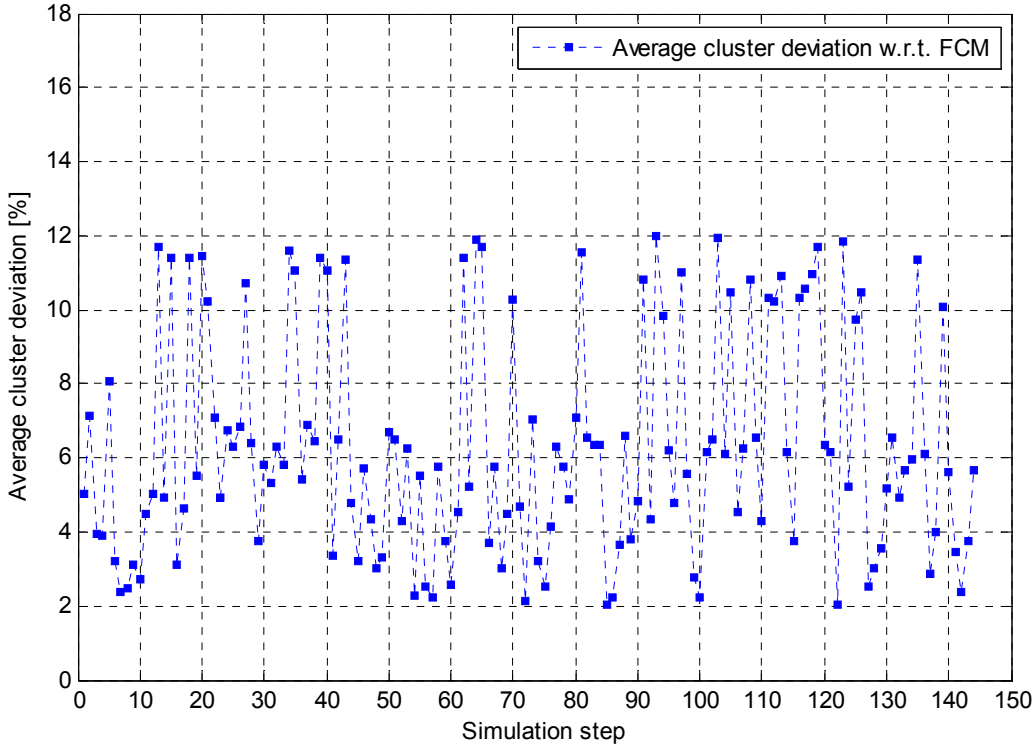


Figure 7.13: Average cluster deviations with respect to FCM for 4D streams.

The above analysis shows that the distributed incremental data stream mining WSN model generally performs well with data streams of different dimensions. The percentage deviation observed throughout is around 11% which is a tolerable margin for several applications. The average cluster deviations increase smoothly with increased stream dimensions. However, maximum and average cluster deviations decrease for 4D stream sets with reference to the K-Means. This may be due to the existence of three distinct non-overlapping classes when all the four features of the stream elements are considered. The data streams used are pre-known to contain three fire danger level classes when all the four features are considered. Further, the K-Means algorithm is known to outperform the FCM algorithm when the datasets contain non-overlapping classes.

7.5.5. Stream Rate Analysis

This simulation involves releasing constant size stream elements from their sources at varying periods. Their release periods are varied to control the stream rates. A number of simulation runs are averaged for every case varying the stream periods. The simulation is then repeated for the different stream dimensions discussed in previous sections. We simulated stream periods of 20 seconds, 15 seconds, 10 seconds, 5 seconds, and 1 second. The effect of stream periods on cluster deviation is shown in Figure 7.14. The streams with the higher dimensions show higher deviations as a function of stream period. 4D stream sets show the highest standard deviation of 3.23 around the mean deviation of 10.13%, while 1D stream sets exhibit the least standard deviation of 0.85 with a mean cluster deviation of 2.42%.

The results indicate that when the data streams consist of higher than 2D elements, the average cluster deviations increase with an increase in the stream periods. Based on 144 simulations runs, the distributed model's cluster deviations for 4D streams are 10.13% +/- 3.23. For the 1D stream, the deviations are bound to 2.42% +/-0.85.

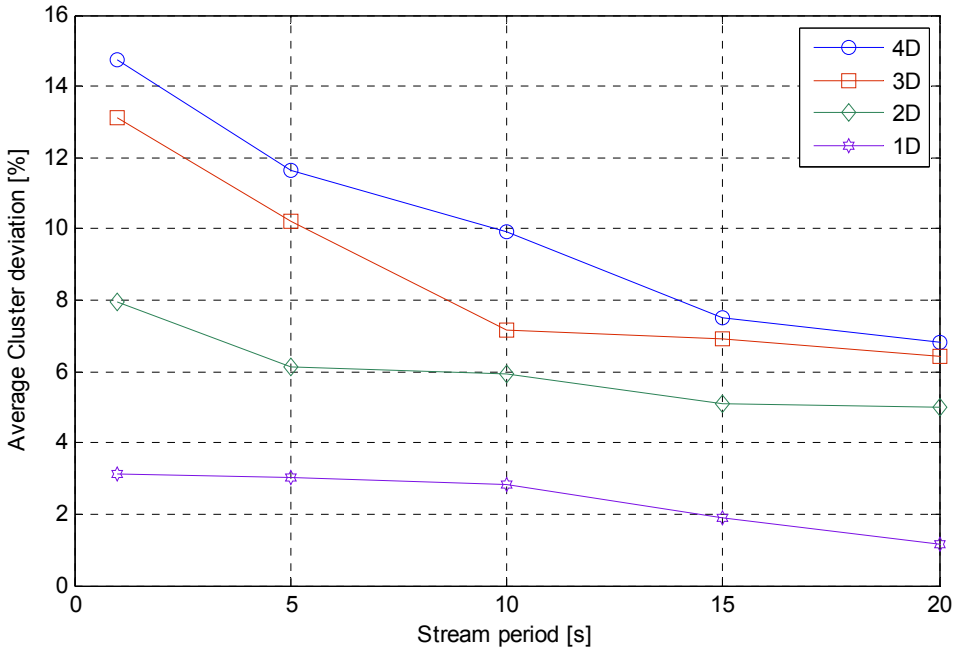


Figure 7.14: Average cluster deviation variation with stream period.

7.6. SIMULATIONS 2

Here we analyze the cluster deviations with varying number of nodes per cluster (cluster density), local model drift thresholds, and stream periods. We consider uniform and non-uniform cluster densities.

The simulations for uniform cluster densities consist of six cases of different cluster densities and seven different local model drift thresholds. A non-uniform clusters case is also simulated. In all cases, the total number of nodes is 200. The number of clusters is varied to accommodate the 200 sensors.

For each of the six uniform node density setups, seven simulations are performed one per local model drift threshold. For the non-uniform cluster density case too, seven simulations are performed: one per local model drift threshold.

Table 7.1: uniform cluster density setup.

Case	No clusters	No. of nodes per cluster	Total no. of nodes per network
1	25	8	200
2	20	10	200
3	10	20	200
4	5	40	200
5	4	50	200
6	2	100	200

Table 7.2: non-uniform cluster density setup.

Case	No. of clusters	No of node per cluster	Total no. of nodes
1	2	50	100
	1	40	40
	2	20	40
	2	10	20
Total no. of nodes per network			200

As shown in Table 1, the simulation cases considered are 25 clusters of eight nodes each, 20 clusters of 10 nodes each, 10 clusters of 20 nodes each, five clusters of 40 nodes each, four clusters of 50 nodes each, and two clusters of 100 nodes each. Local model drift thresholds of 10%, 20%, 30%, 40%, 50%, 60%, and 70% are simulated for each of the above cluster densities in each case keeping the stream periods constant. The stream periods considered are 1sec, 5sec, 10sec, 15sec, and 20sec. Table 2 shows a single case of non-uniform cluster density setup consisting of two clusters of 50 nodes each, a cluster of 40 nodes, two clusters of 20 nodes each, and two clusters of 10 nodes each. The data for this simulation considers all the four features (4D) of the data streams.

7.6.1. Uniform Cluster Density Analysis

When the cluster density is uniform, each cluster contains the same number of nodes. The amount of computation load on every cluster head is therefore similar. In the uniform cluster density setup, the cluster heads compute the local models in more or less similar time periods and hence achieve better global coordination of the model. Therefore the uniform cluster density architecture should perform better than the non-uniform architecture case.

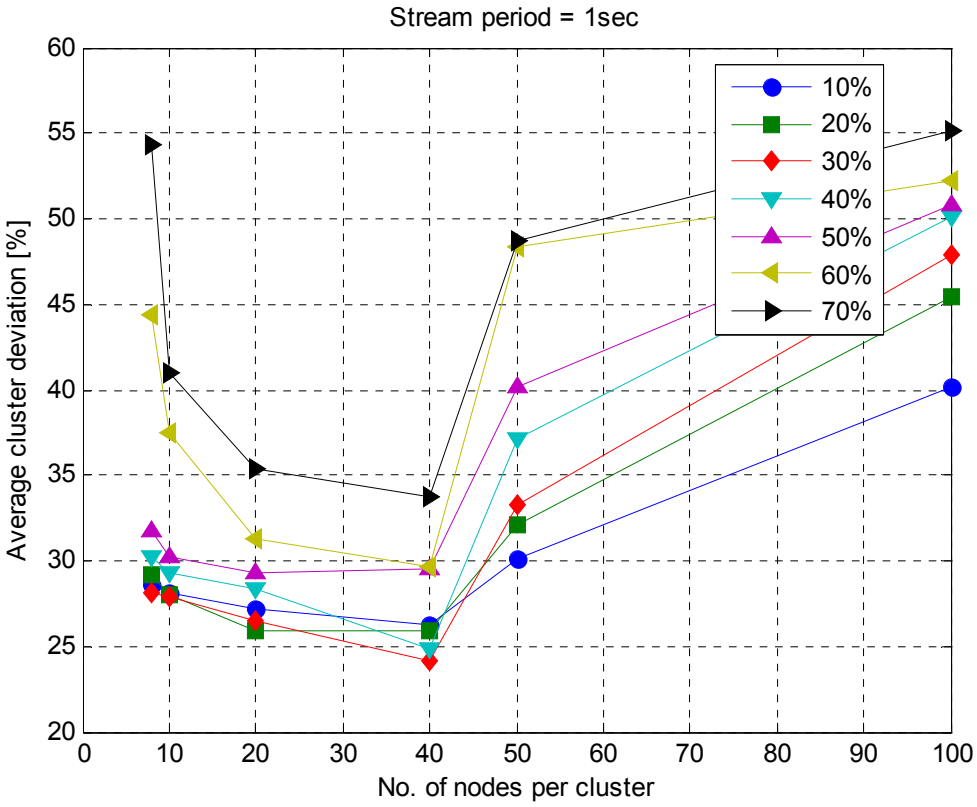


Figure 7.15: Average cluster deviation with varying cluster densities at 1sec stream period.

Figure 7.15 shows simulation results of average cluster deviations as a function of varying cluster densities when the stream period is set to 1 second. For all the local model drift thresholds, the average cluster deviations decrease as the cluster density increases until 40 nodes per cluster is reached and then starts to increase again. The minimum average cluster deviation of 24.15% is observed for the local model drift threshold of 30% and cluster density of 40 nodes per cluster.

Figure 7.16 shows simulation results of average cluster deviations as a function of varying cluster densities when the stream period is 5 seconds. For all local model drift thresholds the average cluster deviations decrease as the cluster density increases in similar fashion as in Figure 7.15. However, in this case, the average deviations are lower and increase proportionally to the local model drift thresholds. The minimum average cluster deviation of 4.55% occurs again at cluster density of 40 nodes per

cluster. The minimum average cluster deviation however corresponds to the local model drift threshold of 10% rather than 30% as in previous simulation.

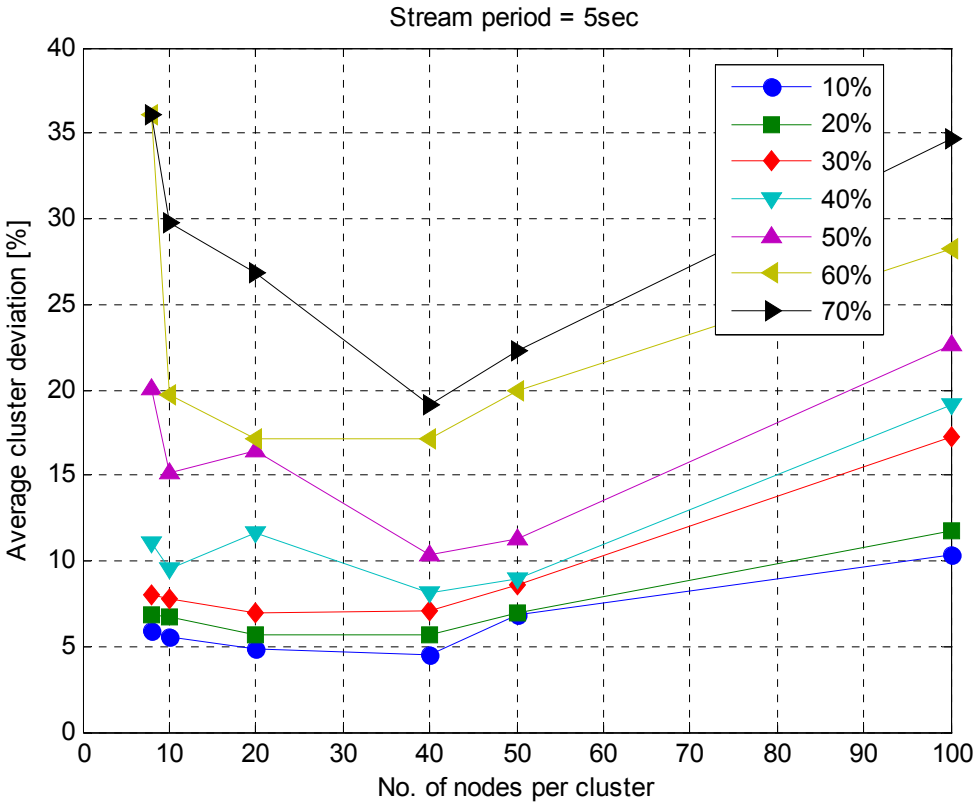


Figure 7.16: Average cluster deviation with varying cluster densities at 5sec stream period.

Increasing the stream period to 10 seconds, 15 seconds, and further to 20 seconds, the average cluster deviations show similar patterns as in Figure 7.16. The average cluster deviations decrease with increase in cluster densities until a minimum is reached at 40 nodes per cluster and starts to increase for further increases in cluster densities. The simulation results for stream release rates of 10 seconds, 15 seconds and 20 seconds are shown in Figures 7.17, 7.18, and 7.19 respectively.

In Figure 7.20, the average cluster deviations for the stream periods of 1sec, 5sec, 10sec, 15sec, and 20sec are averaged at each cluster density. The average cluster deviations remain below 11% at 40 nodes per cluster for all stream periods except in the case of stream period of 1 sec which is 27.73%.

The above observations reveal that the optimum cluster density for best clustering results under the given network architecture and distributed incremental stream mining model is 40 nodes per cluster. The ability of the model to handle data streams arriving in periods of longer than 1 second is also observed. The stream period of 1 second or lower is however too fast for the model as manifested in relatively higher average cluster deviations.

High cluster deviations at a very low number of nodes per cluster in all simulations show that by dividing a given large quantity of datasets into smaller sets, mining these smaller sets at distributed locations simultaneously and incrementally extracting the hidden global structures can yield results comparable to that of mining the whole dataset at a central location. However, as the number of divisions increases, the number of distributed mining locations increase with very small sub-sets of data and the mining results start to degrade in comparison to the central mining results.

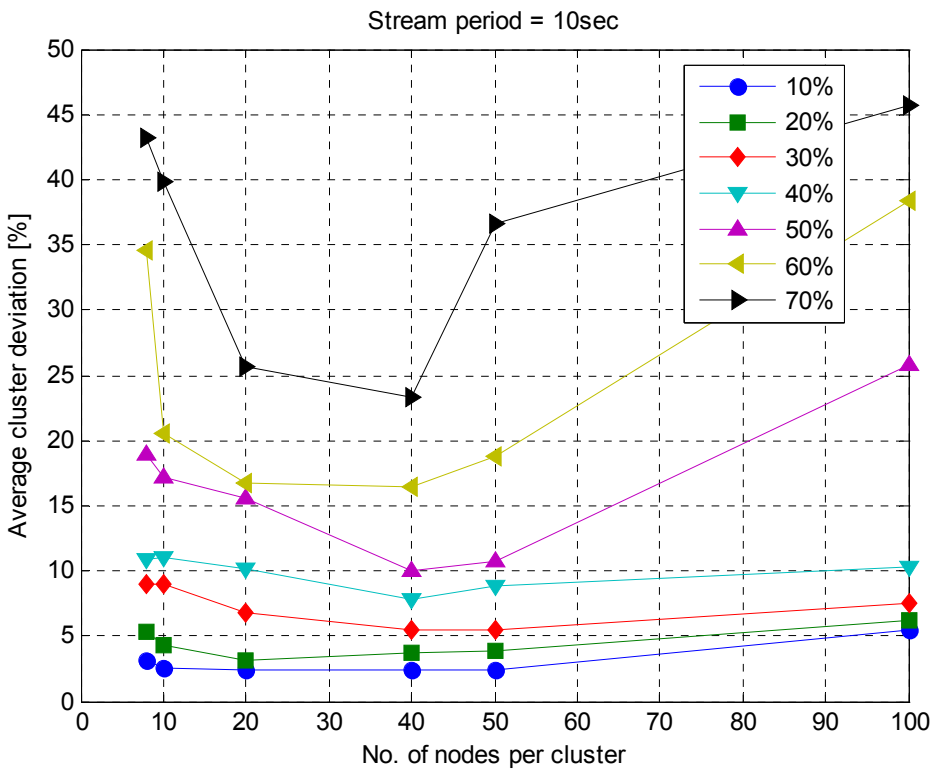


Figure 7.17: Average cluster deviation with varying cluster densities at 10sec stream period.

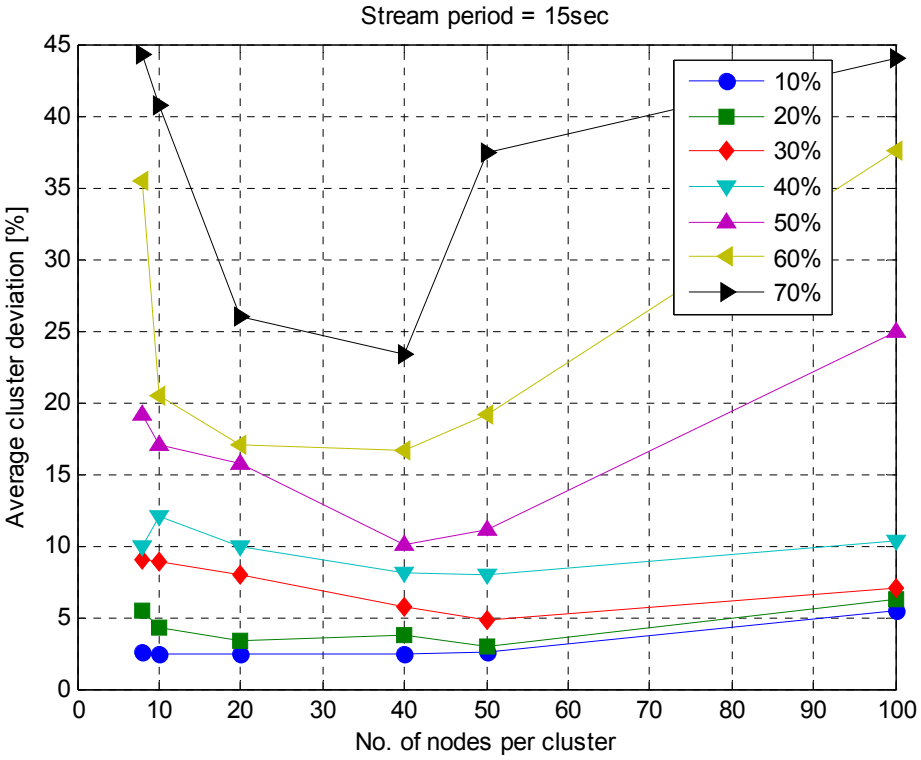


Figure 7.18: Average cluster deviation with varying cluster densities at 15sec stream period.

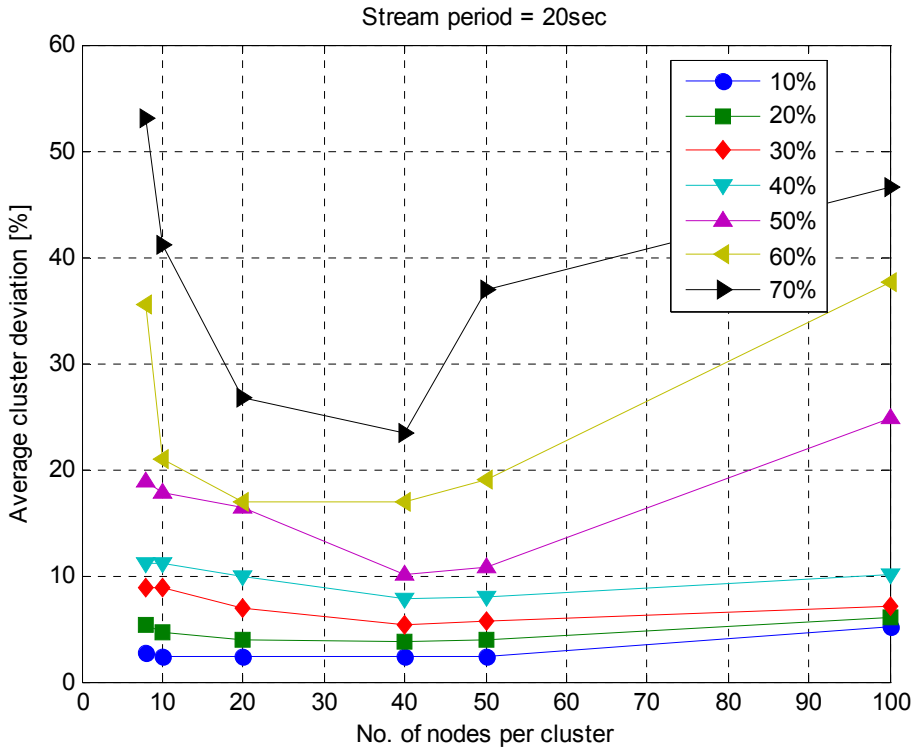


Figure 7.19: Average cluster deviation with varying cluster densities at 20sec stream period.

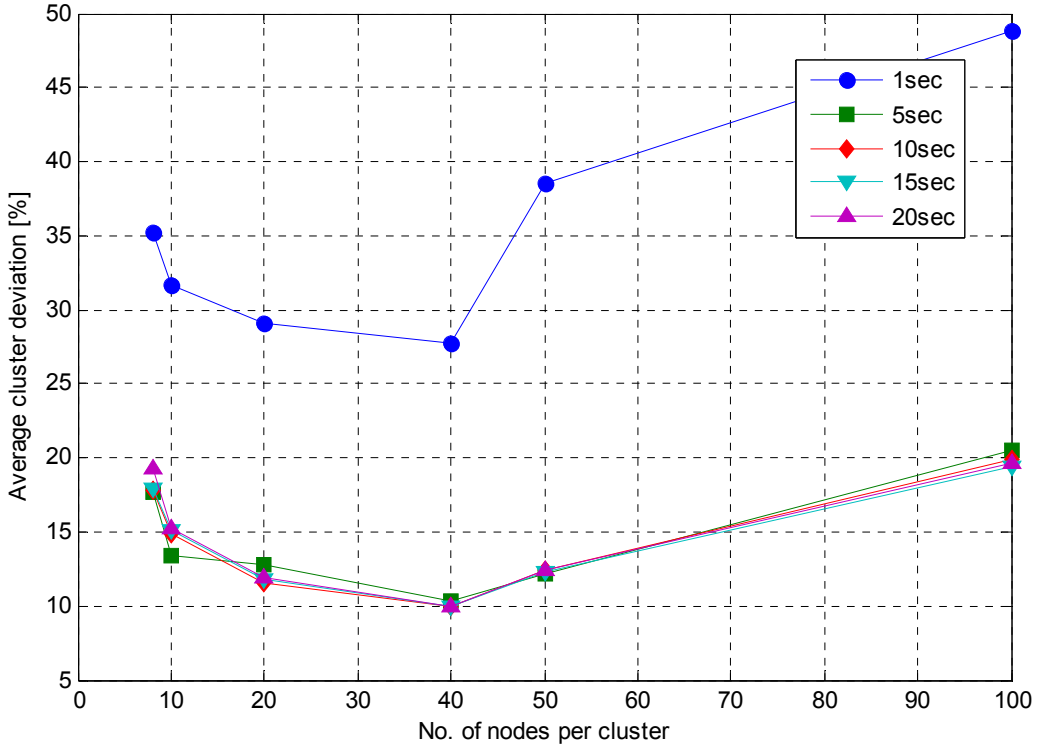


Figure 7.20: Average of cluster deviations for different stream periods.

7.6.2. Non-uniform Cluster Density Analysis

For the case of non-uniform cluster density, the average cluster deviation as a function of local model drift threshold for the different stream periods is plotted in Figure 7.21 below. The cluster deviations are generally higher compared to the uniform node densities per cluster. The stream periods exhibit different minimum and maximum cluster deviations for different local model drift thresholds. The minimum cluster deviation is 31.36% at 40% local model drift threshold when the stream period is 15 sec. The maximum cluster deviation at 40% local model drift threshold is 39.95% when the stream period is 1sec. The average cluster deviation for the non-uniform cluster density of 35.3% at 40% local model drift threshold is noticeably higher than the uniform cluster density counterpart at similar local model drift threshold where the average remained below 10%.

The irregular cluster deviation pattern is due to the assignment of the same stream period for clusters of different densities.

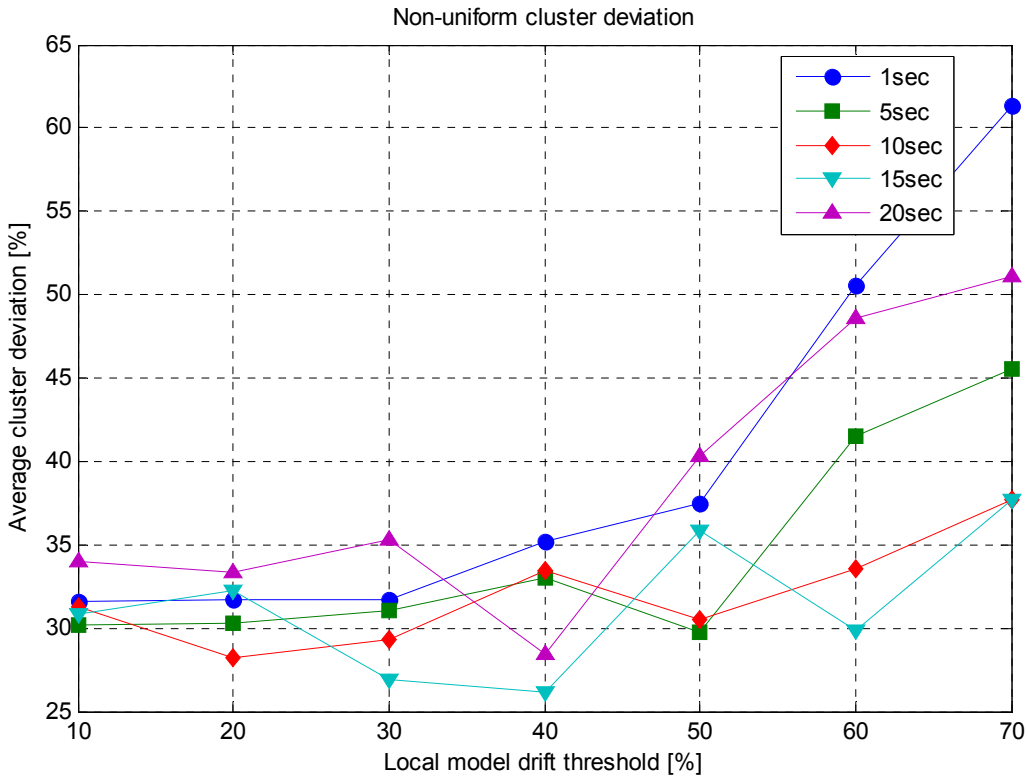


Figure 7.21: Average cluster deviation for non-uniform cluster density.

7.7. SIMULATION 3

This simulation setup aims to evaluate the cost and quality of service that the network can provide for the mining application. Average energy consumption, average data delivery delay, and packet delivery ratio impacts are analyzed.

7.7.1. Average Energy Consumption

The energy consumption of a node includes the energy consumed during the sleep state and active states. The active states are the transmit state, receive state, and algorithm processing state. These different states consume different amounts of energy and last for different lengths of time. The average energy consumption of a node over a given time period is the mean of the energy consumptions of the above states. The base energy consumption of the nodes during all the states except the

algorithm processing state is given in the node hardware datasheet. Based on this we model the nodes' average energy consumption incorporating the algorithm processing state. By logging the time duration of every state during simulation, we calculate the average energy consumption of the sensor nodes and the cluster heads.

The average energy consumption of a typical node taking the average of all nodes in a cluster for a single step of simulation run is shown in Figure 7.22. Similarly, the average energy consumption of a typical cluster head taking the average of all cluster heads in the network for a single simulation step is shown in Figure 7.23. The average energy consumption for the entire simulation run can be found by multiplying this with the number of simulation steps.

In Figure 7.22, the average energy consumption for the sensor nodes and cluster heads is presented. This simulation averages data logged from a network of 25 clusters of eight sensor nodes each. The energy consumption of both the sensor nodes and the cluster heads diminish with increasing stream period. The average energy consumption in both types of nodes decreases rapidly as the stream period increases from one second to five second periods. After the five second stream period the average energy consumption decreases smoothly. This shows the impact of frequent data transmission on the average energy consumption. Increasing the stream period beyond 10 seconds does not significantly decrease the average energy consumption. This indicates that the increased energy consumption at shorter stream periods is partly due to increased packet collisions and retransmissions at faster stream arrival rates. The specific distributed incremental stream mining WSN model energy consumption performance is optimal for stream periods of 5 seconds or more.

The impact of cluster density per cluster on average power consumption of both sensor nodes and cluster heads are shown on Figure 7.23 and Figure 7.24 respectively. In Figure 7.23, the average energy consumption impact due to different node densities per cluster are plotted for a range of stream periods. It can be observed that the impact of stream periods on the average energy consumption of sensor nodes is significantly higher than the impact of node densities per cluster. Figure 7.24 shows the plot of average energy consumption of cluster heads as a function of node density per cluster for the same range of stream periods as in Figure 7.23. The impact of node

densities per cluster on the average energy consumption of the cluster heads is as significant as the impact of stream periods. The impact of cluster density on the cluster heads' average energy consumption at shorter stream periods are more significant. The increased average energy consumption by the cluster heads at higher node densities can be explained by the increased number of packet receptions from their member nodes. Packet collisions and new datasets catching up with unprocessed previous datasets account for increased average energy consumption of cluster heads at fast stream rates.

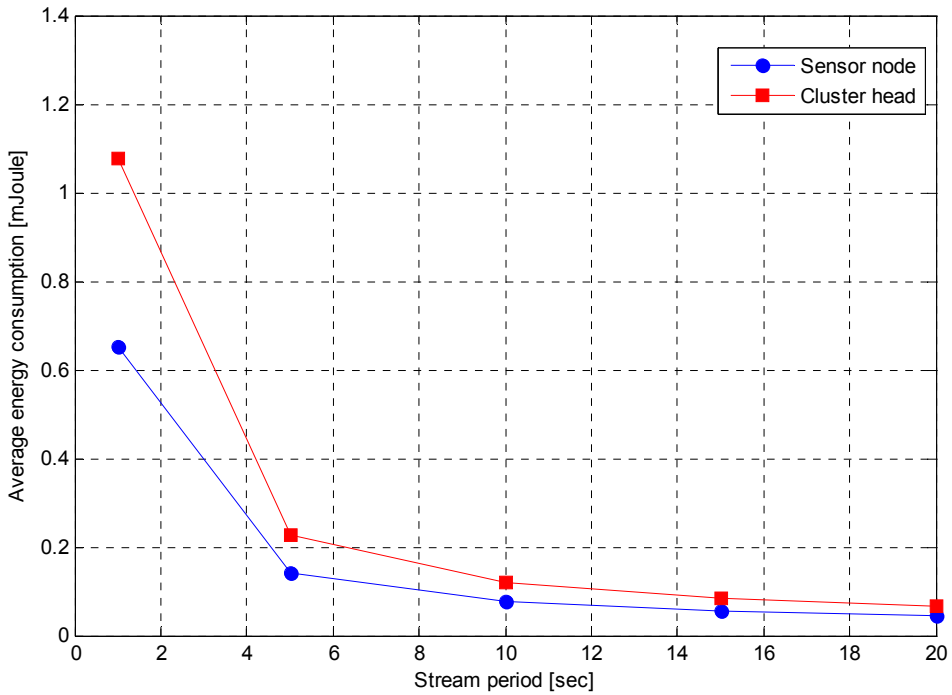


Figure 7.22: Sensor nodes and cluster heads average energy consumption.

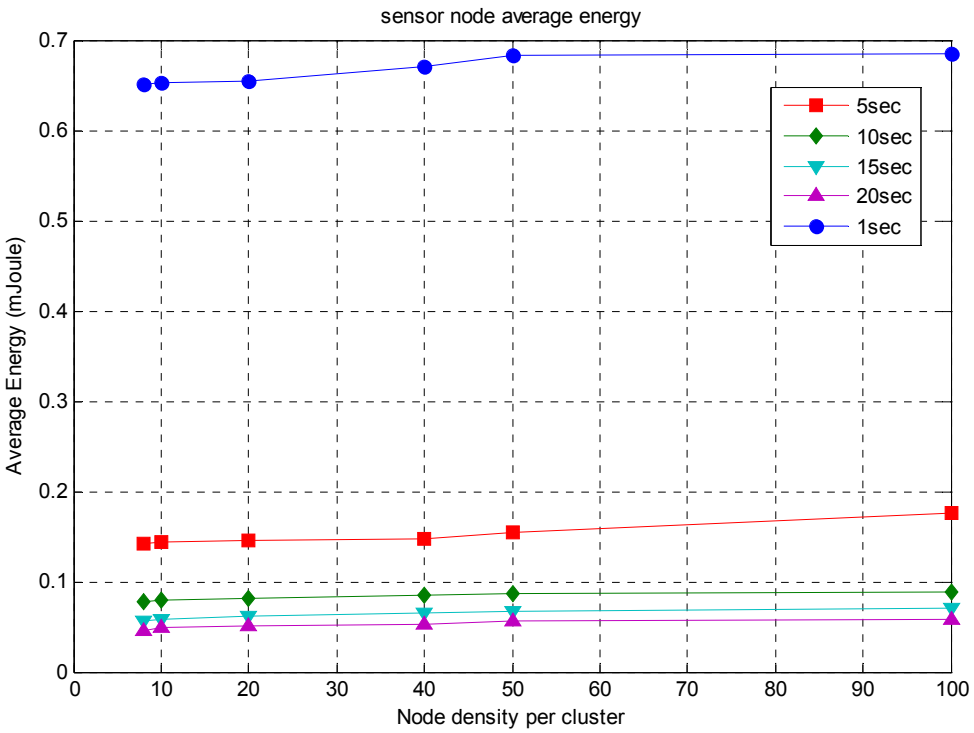


Figure 7.23: Sensor nodes average energy consumption.

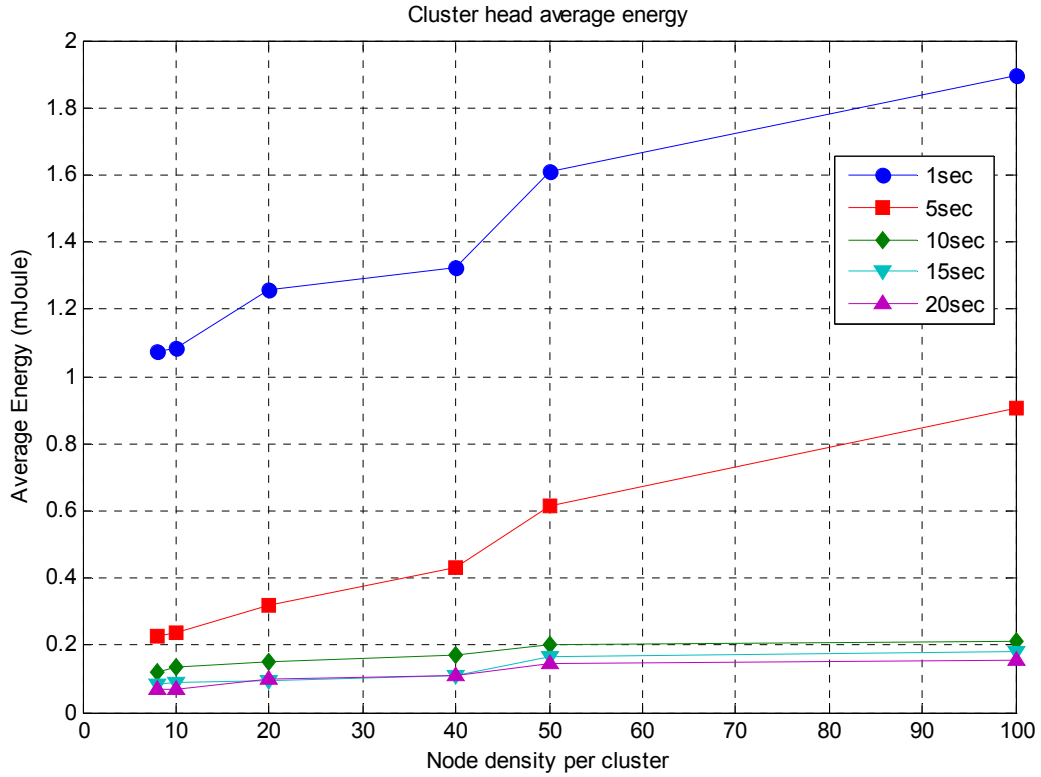


Figure 7.24: Cluster heads average energy consumption.

The average energy consumption of the sensor nodes and the cluster heads largely vary with the stream periods and node densities per cluster. The results in Figure 7.22 show that streams with shorter periods, i.e. streams arriving fast, result in higher average energy consumption than those with longer periods. This is because for longer stream periods the nodes spend more time in sleep mode. The sensor nodes and cluster heads consume above 90% less energy when mining data streams with a period of 20 second compared to mining the same data streams with a period of one second. This could also be due to higher packet collision and lower data delivery ratio at such fast speeds.

7.7.2. Average Data Delivery Delay

The packet delivery delay is defined as the time elapsing between the instant at which a packet is generated at a source, and the instant at which a copy of the packet is first received by the destination [164]. In our model, we define the data delivery delay as the time elapsed between the instant a data packet is generated at a source, and the instant at which the local cluster models generated at the cluster heads, corresponding to the data packet, arrives at the sink.

The average data delivery delay of a typical sensor node taking the average of data delivery delays of every sensor node in the network for that instance is shown in Figure 7.25. From the plot of average data delivery delays as a function of cluster density in Figure 7.25, we can observe that the average data delivery delay increases as the number of nodes per cluster increases. For a stream period of one second, the average data delivery delay exceeds the stream period when more than 40 nodes exist per cluster. This situation indicates saturation of the system due to streams arriving at a rate higher than the rate at which the system can process them. However, for stream periods of five seconds or more, the average data delivery delays are well below 500 ms with the exception of 5 seconds stream period at more than 50 nodes per cluster density.

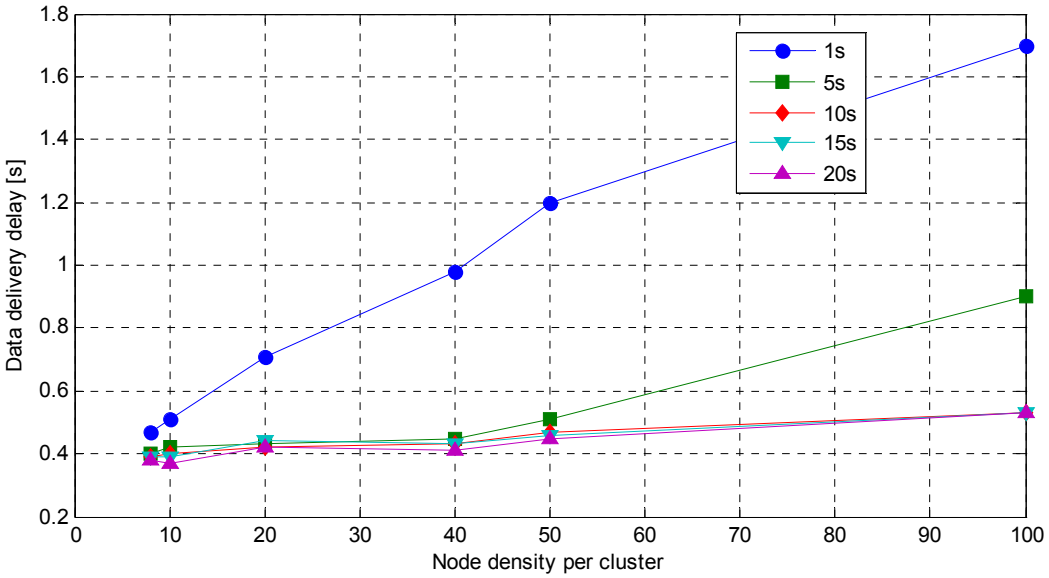


Figure 7.25: Packet delivery delay variation with cluster density.

7.7.3. Packet Delivery Ratio

The packet delivery ratio is the ratio of packets received at the sink to the packets generated by all other nodes [165]. In our model, using cluster architecture and in-network processing, we define two packet delivery ratios: the sensor node-to-cluster head packet delivery ratio and the cluster head-to-sink packet delivery ratio. The sensor node-to-cluster head packet delivery ratio is the ratio of packets received by cluster heads to the packets sent by cluster members, whereas the cluster head-to-sink packet delivery ratio is the ratio of packets received by the sink to the packets sent by the cluster head.

The packets delivery ratios are observed to vary with stream period and cluster density variations. Packet delivery ratios of between 96.7 - 99% are observed when every cluster contains less than 20 nodes, no matter what the stream periods are, as exhibited in Figure 7.26. However, packet delivery ratios drop rapidly to as low as 93.7 to 96.8% when the node density is increased to 50 nodes per cluster, especially at faster stream periods. The stream periods have significant impact on the rate of packet delivery ratio drop as node density increases. This indicates that for optimal performance, applications utilizing the distributed model should limit the cluster density to below 20 if a packet delivery ratio of above 96% is desired. Conversely, a

packet delivery ratio drop should be expected if 50 or a higher cluster density is desired.

The packet delivery ratios of cluster head-to-sink are shown in Figure 7.27. Packet delivery ratios drops of 99.5 – 96.5 are observed for node densities of eight to 50 nodes per cluster for all stream periods, except for one second stream period which further drops to 96%. The lowest packet delivery ratio observed is about 96% for node density of 100 nodes per cluster. For the case of cluster head-to-sink packet delivery ratio, the stream periods do not show significant impact on the rate of packet delivery ratio drop as in sensor node-to-cluster head packet delivery ratio drop.

Applications that can tolerate stream packet delivery ratios as low as 94% at sensor node-to-cluster head and 96% at cluster head-to-sink can deploy as much as 100 nodes per cluster given that the increased energy consumption and data delivery delays as a consequence are acceptable.

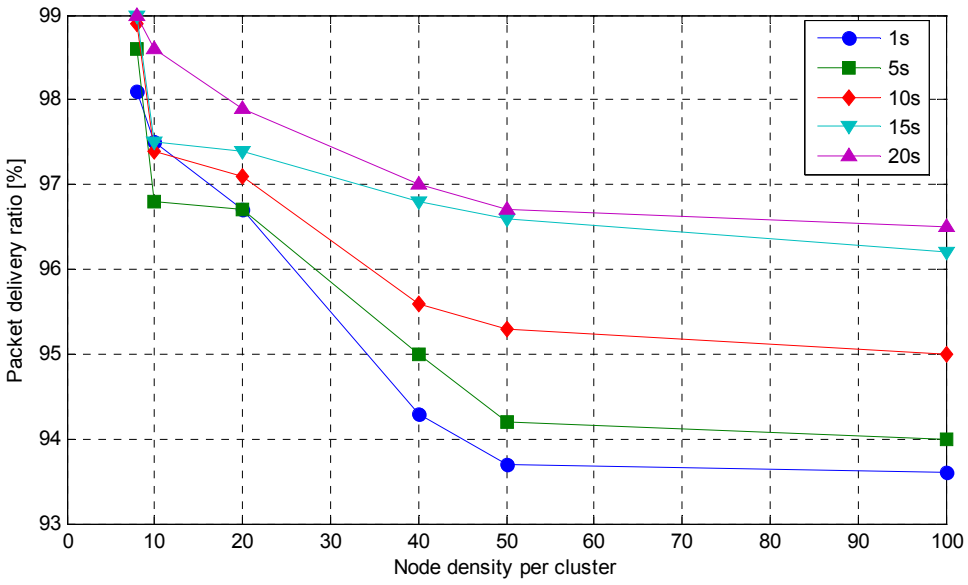


Figure 7.26: Packet delivery ratio of sensor node-to-cluster head.

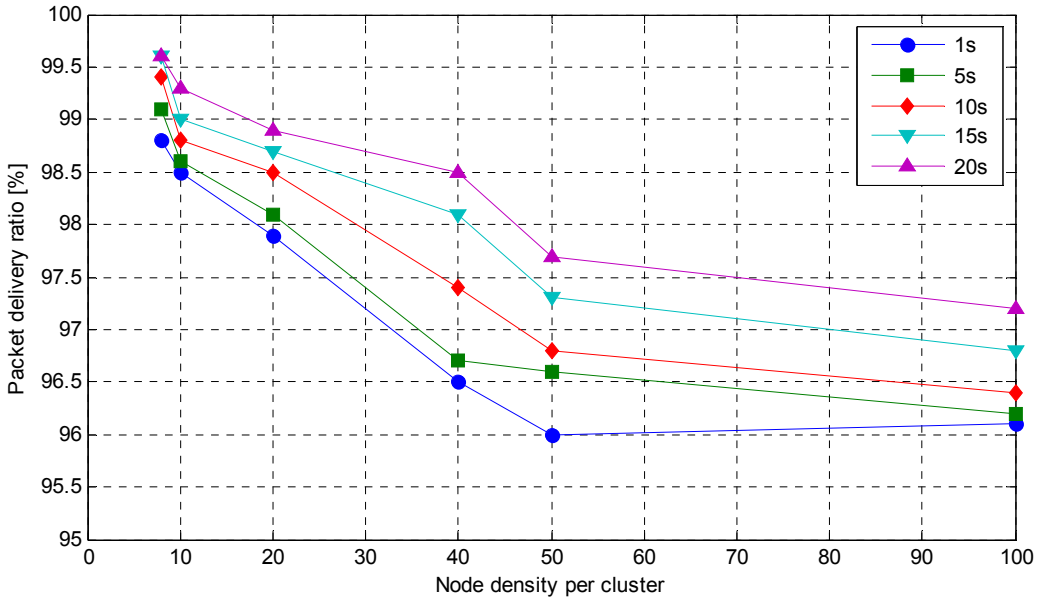


Figure 7.27: packet delivery ratio of cluster head-to-sink.

7.8. CONCLUSIONS

In this chapter, the distributed incremental data stream mining WSN model is evaluated through simulations. The robustness of the model to different data stream dimensions and data stream rates is demonstrated through the first set of simulations. Benchmarking on standard mining algorithms, the K-Means and the FCM algorithms, we have demonstrated that the model can perform high quality distributed data stream mining tasks comparable to centralized data stream mining. The second set of simulations have shed light on the network architectural design guidelines required to satisfy desirable applications demands without compromising the distributed data stream mining task integrity. The third and final set of simulations have also discussed the energy cost and network quality of service impacts for optimal system performance.

Chapter 8

8. CASE STUDY: MICRO-SCALE FOREST FIRE WEATHER INDEX AND SENSOR NETWORK

8.1. INTRODUCTION

The Micro-scale Forest fire Index (FWI) system is an attempt to implement a scaled-down version of the Canadian FWI system for fire danger monitoring of a relatively smaller geographic area. It considers an area as small as a few square meters or as large as many square kilometers. It is specifically important for local forest zones where the nature of the vegetation and topography largely differs from the surrounding forest area. It is based on deployment of large number of low-cost, low-power, and small-sized weather sensor nodes linked by a low-power wireless communication network.

A Micro-scale FWI system locates impending bushfires to their exact locations well before their occurrence, and remotely alerts the authorities with detailed fire management information. It enables high temporal and spatial resolution of information on bushfire activity. This is considered ideal for an early-warning systems of bushfire-prone regions. A large number of low-cost, intelligent and wireless sensors are deployed within the area of interest with the sensors located at short distances apart. These sensors intimately interact with the physical environment of the bush/forest floor and gather the necessary information which is shared among the neighboring sensors wirelessly. The information is fed to the hazard prediction algorithm embedded into each sensor unit to generate an alarm for any fire hazard

and for data management. The system could be organized to alert the fire management authority timely via the available backbone communication (GSM, Internet or satellite) link.

The Micro-scale FWI system, developed during this research work is used as a case study to demonstrate the merits of the distributed incremental data stream mining WSN system.

8.2. FWI SYSTEM

Fire Weather Index (FWI) is an estimation of the risk of wildfire based on the Canadian empirical model developed by Van Wagner [166]. It is one of the most comprehensively used forest fire danger rating systems in North America based on several decades of forestry research [167, 168]. FWI is used to estimate fuel moisture content and generate a series of relative fire behavior indices based on weather observations. The fuel moisture and fire behavior indices are used by operational personnel to aid in the estimation of expected daily fire occurrence, potential fire behavior and difficulty of suppression across a fire management district, region or province [169].

The FWI system takes current weather parameters, elevation data and produces the indexes of the FWI system daily at noon local time. The FWI indexes are indicators of daily potential and behavior of bushfires. The FWI system relies on sparsely distributed meteorological stations as its current weather parameters' data sources. Data from several meteorological stations is transferred to a central processing and repository center via satellite communication. At the central processing and repository center, weather parameter grids for the entire national area will be produced. Geographic Information Systems (GIS) software is used to interpolate the weather data between stations taking into account elevation data to produce gridded weather maps. The FWI System components are then calculated on a cell-by-cell basis according to a set of equations to produce the FWI maps [170].

The FWI system models the complex relationships between the forest weather variables (fire weather observations), the forest floor moisture profiles known as Fuel

Moisture Codes, and the Fire Behavior Indices. Six standard components of the FWI System provide numerical ratings of relative wild land fire potential. The first three components are fuel moisture codes that follow daily changes in the moisture contents of three classes of forest fuel with different drying rates. For each, there are two phases - one for wetting by rain and one for drying - arranged so that the higher values represent lower moisture contents and hence greater flammability. The final three components are fire behavior indices, representing rate of spread, amount of available fuel, and fire intensity; their values increase as fire weather severity worsens. The system is dependent on weather parameters only and does not consider differences in risk, fuel, or topography. It provides a uniform method of rating fire danger across wild land. The six components of the FWI system are described below.

The first three indices are the Fine Fuel Moisture Code (FFMC), the Duff Moisture Code (DMC), and the Drought Code (DC). The FFMC relates the fire weather observations to ease of ignition of the litters and other cured fine fuels at top layer of the forest floor, hence a good indicator of likeliness of forest fire ignition at the observed locations. The DMC relates fire weather observations to the rate of fire fuel consumption at the loosely compacted organic layers of moderate depth duff layers. The DMC is indicative of the amount of fuel that would have been consumed in this layer, had the fire materialized. The DC relates the fire weather observations to the seasonal drought effects on the deep compact organic layers. The DC is an indication of the amount of smoldering in deep duff layers and large logs.

The last three indices are the Initial Spread Index (ISI), The Build Up Index (BUI), and the Fire Weather Index (FWI). The ISI is a numeric rating of the expected rate of fire spread. The BUI is a numeric rating of the total amount of fuel available for combustion. The FWI is the numeric rating of the fire intensity, and it’s a general index of fire danger of a given forest area.

The model equations of the fuel moisture codes and fire behavior indices are described below based on the general structure of the FWI system as shown in Figure 8.1**.** The fire danger severity rating on FWI scale is shown in Table 8.1.

8.2.1. Fine fuel moisture code (FFMC)

FFMC is a numerical rating of the moisture content of litter and other cured fine fuels. It indicates the relative ease of ignition and flammability of fine fuel. Calculation of FFMC requires the current moisture condition m of the fuel, which is determined by the combined effect of rainfall and absorption/desorption of atmospheric moisture. The rainfall effect is described as the rain modified moisture content m_r. The absorption/desorption of atmospheric moisture is described as diffusion of the gradient of the initial moisture from its wetting equilibrium (E_w)/drying equilibrium (E_d) moisture content. The wetting and drying diffusion coefficients (K_w and K_d, respectively) are functions of relative humidity (RH) in %, temperature (T) in ^{0}C, and wind speed (v) in km h^{-1}. The wetting equilibrium and drying equilibrium moisture contents of the fuel are given by relative humidity and temperature in reference to noon temperature of 21.1 ^{0}C.

$$FFMC = 59.5\frac{250 - m}{147.2 + m} \tag{8.1}$$

8.2.2. Duff Moisture Code (DMC)

DMC is a numerical rating of the average moisture content of loosely compacted organic layers of moderate depth. This code gives an indication of fuel consumption in moderate duff layers and medium-size woody material.

The DMC is a combined effect of rainfall modified duff moisture code DMC_r and evaporation from the duff layer DMC_d which are functions of temperature (T), relative humidity (RH) and effective day length (L_{eff}).

$$DMC = DMC_r + DMC_d \tag{8.2}$$

where

$$DMC_r = 244.72 - 43.43\ln(m_r - 20)$$

$$DMC_d = 1.894(T + 1.1)(100 - RH)L_{eff}10^{-4}$$

m_r is the rain modified moisture code.

8.2.3. Drought Code (DC)

DC is a numerical rating of the average moisture content of deep, compact, organic layers. This code is a useful indicator of seasonal drought effects on forest fuels, and amount of smoldering in deep duff layers and large logs.

The DC is determined by estimating the change in a moisture equivalent scale caused by a source term (i.e. the effective rainfall) and loss term (evapotranspiration and drainage). During the rainfall phase, the rainfall modified drought code DC_r is a function of rain modified moisture equivalent scale Q_r. During the drying phase, the moisture loss from the duff layer is approximated by DC_d, which is a function of temperature (T) and seasonal day length adjustment, L_f.

$$DC = DC_r + DC_d \tag{8.3}$$

where $DC_r = 400\ln(800/Q_r)$ and $DC_d = 0.5(0.36(T + 2.8) + L_f)$

8.2.4. Initial Spread Index(ISI)

ISI is a numerical rating of the expected rate of fire spread. It combines the effects of wind and FFMC on rate of spread without the influence of variable quantities of fuel.

The ISI is related to FFMC and wind speed, v, limited to a maximum of 100 km h^{-1}. It has the wind speed component, FW and the FFMC component, FF, related through the current moisture condition m.

$$ISI = 0.208(FW)(FF) \tag{8.4}$$

where $FW = e^{0.5039v}$ and $FF = 91.9e^{-0.1386m}\left(1+\frac{m^{5.31}}{4.93\cdot 10^{7}}\right)$

8.2.5. Buildup Index (BUI)

BUI is a numerical rating of the total amount of fuel available for combustion that combines DMC and DC.

The BUI is calculated by combining DMC and DC. A form of the harmonic mean of the DMC and the DC is used to calculate the BUI [168], to ensure that changes about smaller values of either the DMC or the DC will receive a greater weight.

$$BUI = \begin{cases} \dfrac{0.8DMC\cdot DC}{DMC+0.4DC} & DMC \le 0.4DC \\ DMC-\left(1-\dfrac{0.8DC}{DMC+0.4DC}\right)\left[0.92+(0.0114DMC)^{1.7}\right] & DMC > 0.4DC \end{cases} \tag{8.5}$$

8.2.6. Fire Weather Index (FWI)

FWI is a numerical rating of fire intensity that combines ISI and BUI. It is suitable as a general index of fire danger throughout the forest areas. Table 8.1 below shows the range of fire danger severity rating of each component.

The FWI is a function of both the BUI and the ISI and is given as;

$$FWI = \begin{cases} B & B < 1 \\ e^{2.72(0.434 \ln B)^{0.647}} & B \geq 1 \end{cases} \tag{8.6}$$

where, $B = 0.1(FD)(ISI)$ and $FD = \begin{cases} 0.626BUI^{0.809} + 2 & BUI \leq 80 \\ \dfrac{1000}{25 + 108.64e^{-0.023BUI}} & BUI > 80 \end{cases}$

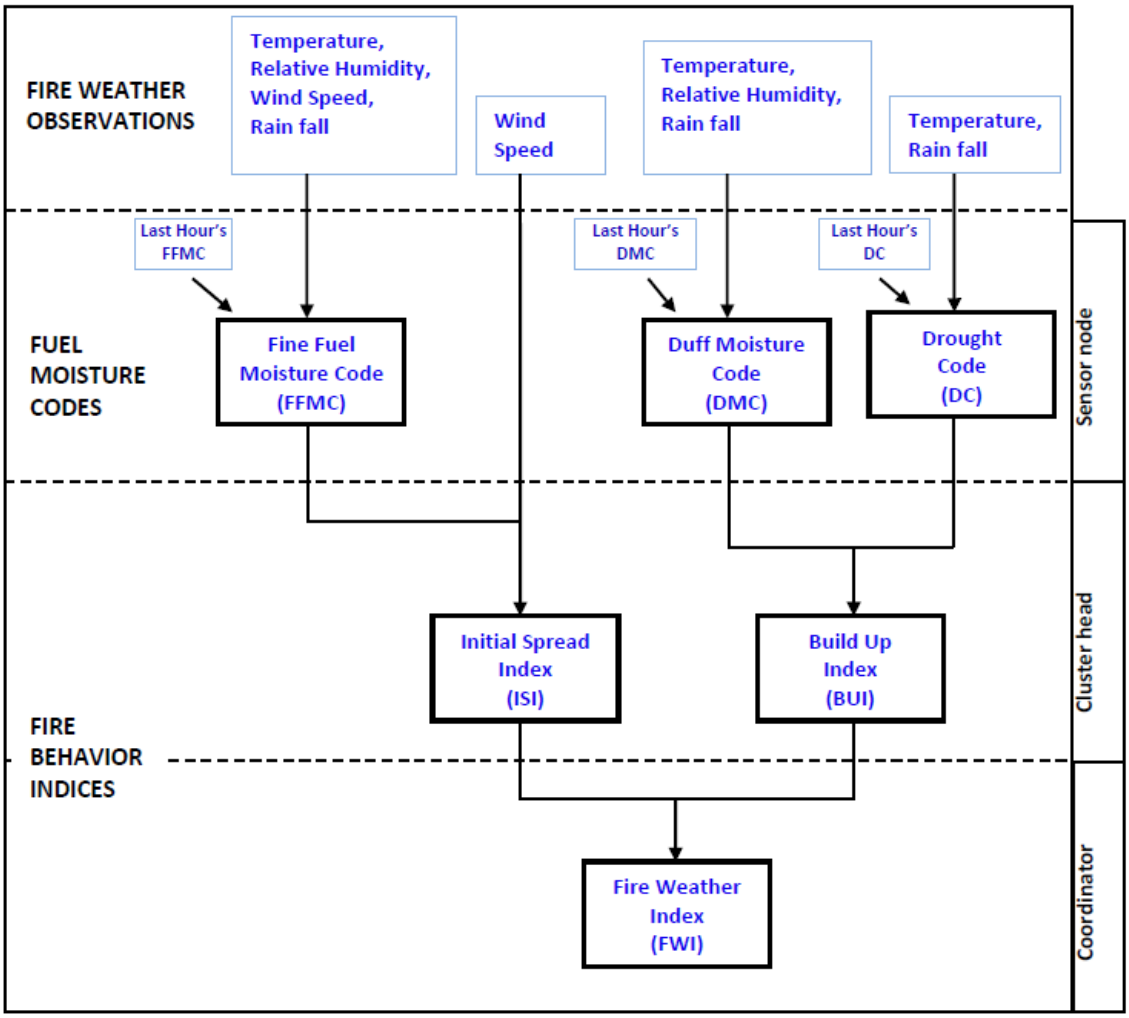


Figure 8.1: The general structure of the FWI system.

Table 8.1: Fire danger severity rating on the FWI scale.

Index	Low	Moderate	High	Extreme
FFMC	0.0 - 80.8	80.9 - 86.8	86.9 - 89.9	90.0+
DMC	0.0 - 15.8	15.9 - 30.8	30.9 - 50.9	51.0+
DC	0.0 - 139.9	140.0 - 239.9	240.0 - 340.9	341.0+
ISI	0.0 - 2.1	2.2 - 4.9	5.0 - 9.9	10.0+
BUI	0.0 - 19.9	20.0 - 35.9	36.0 - 60.9	61.0+
FWI	0.0 - 2.9	3.0 - 9.9	10.0 - 22.9	23.0+

8.3. LIMITATIONS OF THE STANDARD FWI SYSTEM

The main limitations of the standard FWI system are its fire danger rating time and space resolution. The standard FWI system rates relative mid-afternoon fire danger from noontime weather data [171]. Hence, the fire danger rating is on a daily or 24 hours cycles. The effect of fluctuations in weather parameters on the fire danger rating within the 24 hours period is thus ignored by the system. The FWI system does not account for the differences in forest cover types and relies on interpolated point-source weather records [171]. Further, the FWI weather data sources, which are meteorology stations, are sparsely distributed and hence weather records of a large area between the stations is estimated based on interpolation. This shows that the spatial resolution of the fire danger rating is very low. In unfortunate events of station failures, due to either lightning strike or technical fault, the spatial resolution degrades further as interpolation values are directly proportional to the number of reference points.

another limitation of the FWI system is that the system does not allow real-time querying of specific forest domain at specific times for hazard rating. This may be necessary when there are specific domains of the forest that require special attention such as urban-rural-interface, national parks, and nuclear facility. A new station start-up and integration also involves high cost, expensive labor and long time.

The above mentioned limitations of the standard FWI system can potentially be overcome by the Micro-scale FWI system.

In the micro-scale FWI system, the forest zone is divided into grids of small square cells (e.g. 20m x 20m) and a low-cost weather sensor node is placed in each cell. Since the size of the square cells can be as small as possible, high spatial resolution weather parameter measurements can be made by taking a large number of measurement points in the region. The fire danger rating values' accuracy is robust to a single or a few sensor failures due to the dense measurement points. The size of the forest zone that can be monitored using a Micro-scale FWI system is determined by

the WSN which usually employs numerous low cost nodes communicating over multiple hops to cover a large geographical area.

The micro-scale FWI system also provides high temporal resolution fire danger rating as the system can make low-power frequent measurements and transmit (e.g. hourly) to base station. The system can also operate in event detection mode, in which the individual sensor nodes instantaneously send information to the base station upon detection of a predetermined danger rating threshold.

The micro-scale FWI system allows intermittent interaction to the normal (e.g. hourly) operation for querying a specific region's situation. Such queries can be generated and injected into the network at the base station. The Micro-scale FWI system deployment is fast as the weather sensor nodes have self-configuration and healing capabilities to automatically link and form a robust network.

8.4. THE MICRO-SCALE FWI SYSTEM

This work implements a scaled down version of the standard Canadian FWI System-*the Micro-scale FWI system*. Its main purpose is to provide high temporal and spatial monitoring of forest zones spanning as small as few square meters or as large as few square kilometers referred to as micro-scale area. The Micro-scale FWI system produces the FWI rating map of a micro-scale area based on wireless weather sensor nodes deployed throughout the micro-scale area in large numbers. The weather sensors and FWI System components are carefully analyzed and calculations are transposed to suit a high temporal and spatial resolution micro-scale area fire danger rating system.

The Micro-scale FWI system consists of a large number of low-cost, low-power, and small-sized weather sensor nodes linked via a low-power wireless communication network. At the root of the wireless network is a sink node, which is physically connected to a base station computer. The base station computer acts as a gateway to an external networks (e.g. Internet, GSM) for remote management of the system.

The Micro-scale FWI system can be described based on the main three components of the system; the weather sensor nodes, the FWI indices processing algorithm embedded within each weather sensor node, and the wireless communication network linking these nodes.

8.4.1. The Weather Sensor Nodes

The weather sensor nodes are the basic building blocks of the micro-scale FWI system. They are the sources of the weather data for the system and play a similar role to that of meteorology stations in the standard FWI system. They are capable of individually probing their surroundings and acquire weather parameters such as temperature, relative humidity, wind speed, and rainfall. They are also capable of minor data processing and short range wireless communication with other nearby nodes.

A weather sensor node is a wireless node also known as a mote [172] with weather parameter sensing capability. It is a small low-power electronic device that combines programmable general-purpose computing with multiple weather parameter sensing and wireless communication capabilities. The basic components of a wireless sensor node are microcontroller, transceiver, sensors, and power source.

A Robust weather sensor node can be constructed based on state-of-the-art wireless node platforms such as MEMSIC's low-power platforms (IRIS, MICAz/MICA2, TelosB, Cricket) and Texas Instrument's system-on-chip (CC2430/31, CC2480, CC2530). All of these platforms provide a low-power Micro Controller Unit (MCU) and an IEEE 802.15.4 compliant radio transceiver on a small-size device onto which sensor boards can be plugged for specific applications.

The Power source for the nodes are often batteries. The sheer number of sensor nodes required for most applications makes battery replacement expensive [173]. Therefore, using dynamic power management schemes and using batteries rechargeable through solar cells are often recommended. Sensors are used by the node as its means of interacting with the physical world. The continual analog signal produced by the sensors is digitized by the analog-to-digital converter (ADC) unit of the microcontroller and is passed on to the application for further processing. As wireless

sensor nodes are typically very small electronic devices, they can only be equipped with a limited power source. Hence, sensors have to have extremely low energy requirements in probing the environment. A typical sensor node architeture is shown in Figure 8.2.

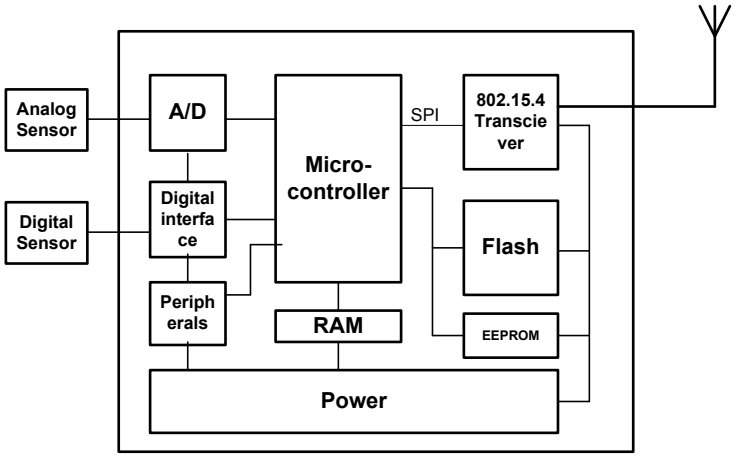


Figure 8.2: Wireless sensor node architecture.

In the Micro-scale FWI system application there are three classes of sensor nodes; the weather sensor nodes, the cluster heads, and the sink.

The weather sensor nodes are the most elementary devices of the three classes. They host atmospheric temperature and relative humidity sensors. Their primary purpose is to probe their environment for the two physical parameters, minor data processing and regularly transmit the information to the cluster heads while keeping their energy consumption as low as possible.

The cluster heads are better equipped devices with more processing power and energy source than the weather sensor nodes. They host wind speed and rainfall sensors. The cluster heads take regular measurements of wind speed and rainfall and make these data available for their member weather sensor nodes. They regularly receive information from their member weather sensor nodes and undergo information processing before transmitting to the sink. They also act as relay nodes for other cluster heads' data to the sink.

The sink is a root node of the network responsible for creating and maintaining the network. It is a general harvesting point of the information produced by the

network. The raw data and information collected by the individual weather sensor nodes is fused, in stages, and forwarded to the sink node that provides the interface to the outside world. The sink is physically connected to a base station computer and hence has constant power supply.

8.4.2. Siting and Exposure of the Sensors

Selecting an appropriate site for weather station is critical for obtaining accurate meteorological data that represents the general area of interest. There are guidelines on weather station sitings and sensor placements defined by regulatory bodies such as World Meteorological Organization (WMO), other standard climatologists such as the American Association of State Climatologists (AASC), and research agencies such as the Environmental Protection Agency (EPA).

According to the EPA, wind speed and direction sensors should be located at a distance of at least 10 times the height of nearby objects [174]. The standard measurement height for wind speed and direction also should be 10m [174, 175]. See Figure 8.3 for reference.

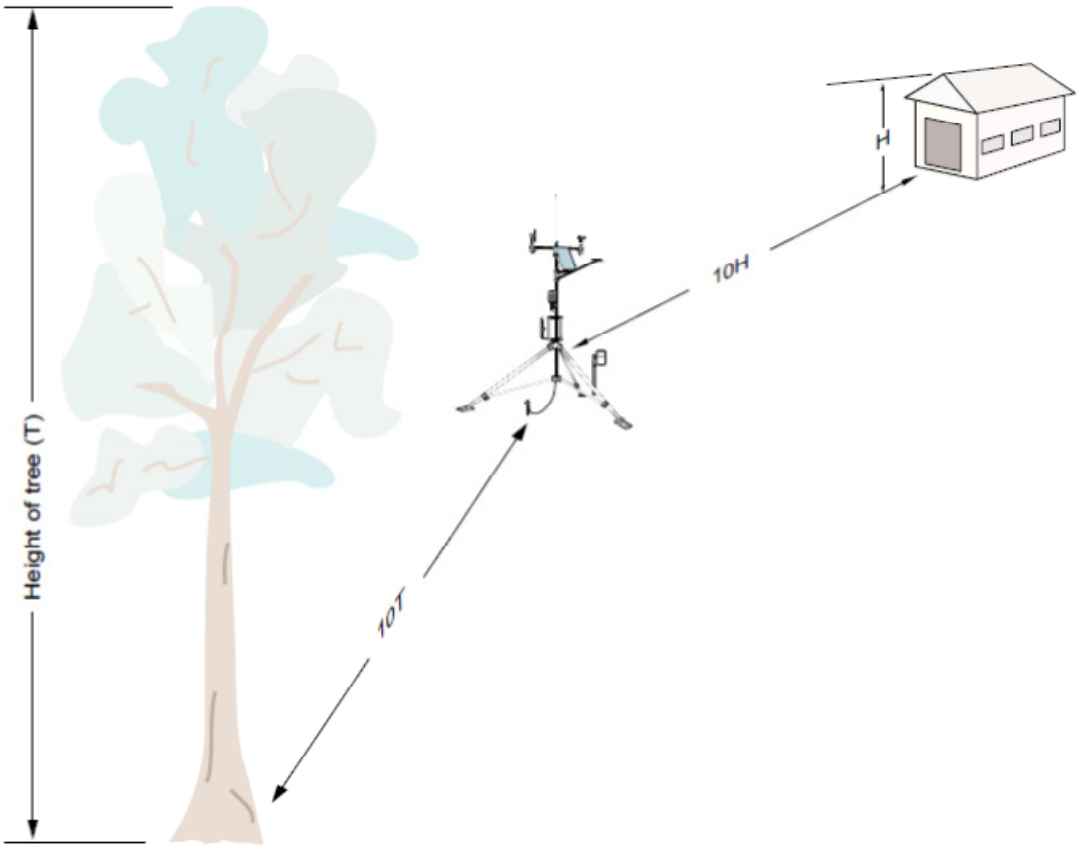


Figure 8.3: Wind speed and direction sensors siting and exposure.

Temperature and relative humidity sensors should be housed in a ventilated radiation shield and be sited no closer than four times the nearest obstruction's height [174]. The recommended standard measurement height is 1.25 to 2.0 meters [175].

The AASC and EPA suggest tipping buckets for precipitation measurement be no closer than four times the height of an obstruction [174, 176]. Typically, tipping buckets are sited on level ground covered with short grass or gravel. WMO and EPA recommend standard precipitation measurement height of 30cm minimum [174, 175].

The guidelines described above on siting and exposure of the weather stations is based on standard meteorological stations whose readings are required to be general representatives of a large area. Thus the sensors should be sited such that their readings closely correlate with the readings of the large area in general. However, in distributed dense point weather parameter measurement systems such as the Micro-scale FWI system, the objective is to measure the actual weather parameters at

a large number of locations in close proximity. Therefore, these guidelines can only be used as general guidelines in Micro-scale FWI system's sensor siting and do not directly apply. The optimal location and orientation of the nodes of Micro-scale FWI system are according to the manufacturer specifications of the sensors used and consider connectivity, coverage, and reading accuracy.

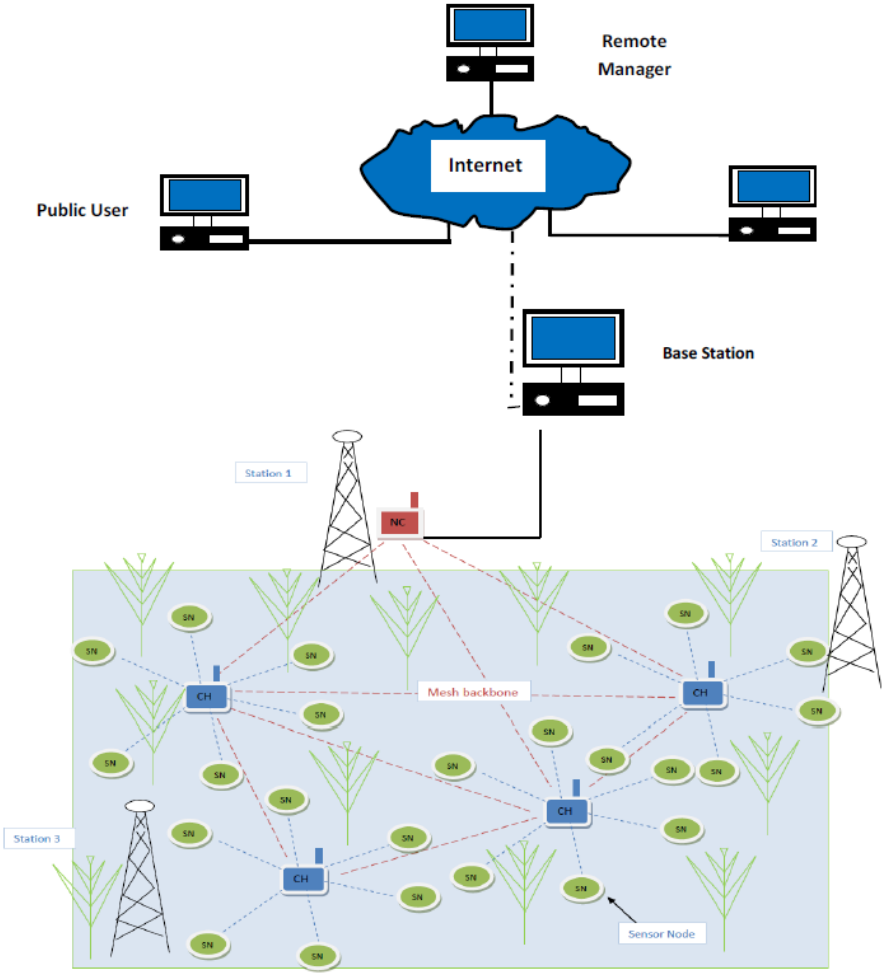


Figure 8.4: The Micro-scale FWI system WSN architecture.

8.5. THE FIRE WEATHER NETWORK

A Micro-scale FWI system acquires current weather parameters data simultaneously from a large number of sources (weather sensor nodes) densely distributed on a given forest zone. The weather sensor nodes, cluster heads and sink are linked via wireless communication to form a structured fire weather network. The fire weather network structure consists of weather sensor nodes as data sources, cluster heads as points of data aggregation and false alarm filtering, and a sink as a gateway to the base station and external network. The data acquired by a large number of weather sensor nodes at different locations in the area monitored will be transmitted to a base station to be processed and provide fire potential, prediction, and behavior information to fire managers. In an integrated system, where nodes are placed around a metrology station, the metrology station can replace the cluster head. The general system architecture is depicted in Figure 8.4 above.

The fire weather network is based on IEEE 802.15.4 LR-WPAN - ZigBee. The IEEE 802.15.4 LR-WPAN is designed to be used in applications requiring simple wireless communication links over short-ranges with limited power and low throughput. ZigBee defines a high-level communication protocol using the IEEE 802.12.4 based small, low-power digital radios. The fire weather network, based on ZigBee wireless network, operates in the unlicensed ISM band of 2.4 GHz.

The fire weather network is configured to operate in a two-tiered architecture. In the first tier, the weather sensor nodes form a star topology with their cluster head as the central coordinator. In the second tier, the cluster head nodes and the sink form a mesh topology utilizing the peer-to-peer communication feature of the ZigBee protocol. The fire weather network architecture is shown in Figure 8.5 below.

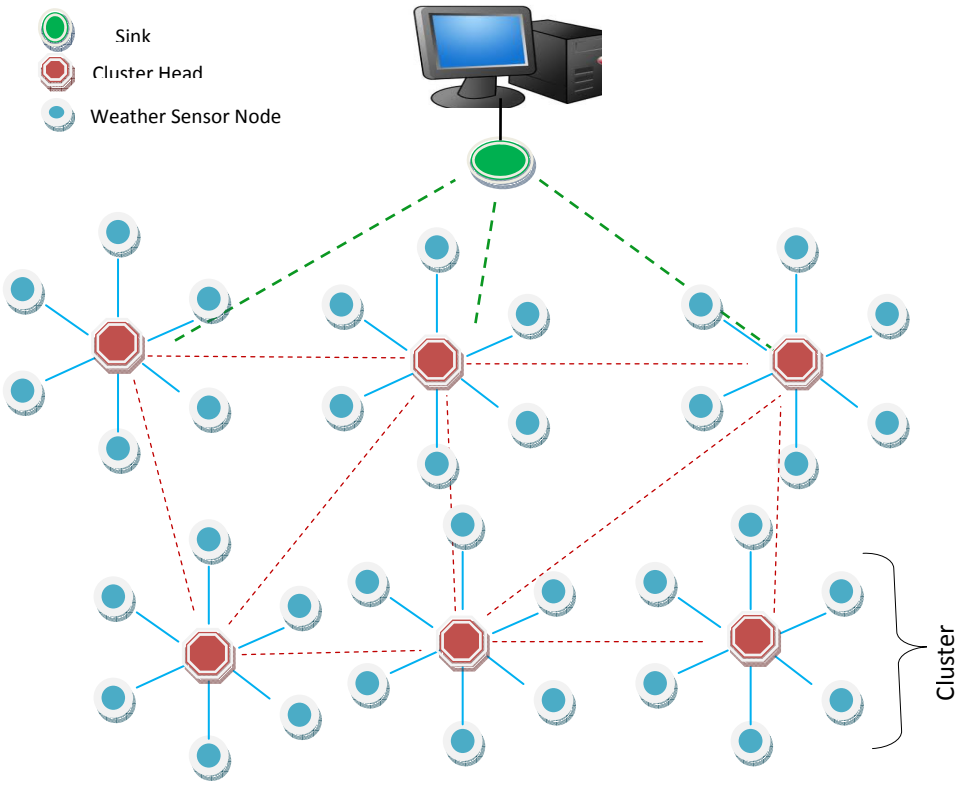


Figure 8.5: The fire weather network architecture.

8.6. FWI INDICES PROCESSING ALGORITHM

The Micro-scale FWI system utilizes the distributed incremental in-network computing of the FWI indices for two main purposes. First, the distributed in-network computing will reduce the amount of raw data transmissions, which will in effect conserve the limited energy resource of the nodes and consequently extend the network lifetime. Second, computing the FWI indices close to the data sources and transmitting a compact summary to the base station will reduce the information latency.

The different nodes of the fire weather network (sensor nodes, cluster heads, and sink) are assigned different tasks of computing parts of the FWI indices. The fire weather network architecture (Figure 8.5) facilitates the coordination of the individual nodes' tasks so that the complex computation of the FWI indices can be achieved through minor local computations.

The Micro-scale FWI system is designed to handle streaming weather data in near real-time and produce fire rating information dynamically as weather parameters change. The weather parameters can change unpredictably and drastically causing the fire danger rating concept drift. The fire weather network's response to such drastic changes may cause the computational and communication complexity to grow beyond what the fire weather network is designed to handle. To mitigate this, the system further embeds the distributed subtractive fuzzy clustering algorithm-SUBFCM [152] onboard each cluster head to contain computational and communication complexity growth as a result of FWI concept drift.

8.7. THE NODES TASK SUBDIVISION

The weather sensor nodes periodically acquire ambient temperature and relative humidity data locally as well as receive wind speed and rainfall data over the air from their cluster head. They instantly compute local fuel moisture codes (FFMC, DMC, DC) using the weather data acquired. If their local fuel moisture codes exceed a set fuel moisture code threshold then they will send the set of new values to their cluster head and request an update. However, if their local fuel moisture codes are within the set threshold, then they enter the low power mode until the next period and repeat this cycle.

The cluster heads periodically acquire wind speed and rainfall data and send to their member weather sensor nodes. They receive fuel moisture code data from member nodes and cluster them using the SUBFCM clustering algorithm whereby each cluster contains associated member nodes. On the first run, the cluster heads send this information to their member nodes so that the member nodes can use them as a reference to decide either to send their subsequently computed fuel moisture codes to their cluster head or do nothing and return to the low power mode. Following the first run, the cluster heads receive fuel moisture codes from their member nodes and categorize them into compact clusters. The cluster heads further compute fire behavior indices (ISI, BUI) using the compact clusters of fuel moisture codes and send them to the sink. In case there are a set number of update requests from member

nodes, the cluster heads send fuel moisture code cluster information to their member nodes.

The sink computes the FWI index based on the fire behavior indices received from the cluster head nodes. It merges clusters of information received from the cluster heads and weather sensor nodes into global information regarding the monitoring area. The sink also maintains the global locations of all the network nodes and physical node clusters.

The Base station, hard wired to the sink, carries out computation and energy intensive analysis of the global information produced by the sink. The base station is the first point of user interaction with the sensor network. It provides graphical information output and well as data statistics results.

One of the graphical outputs by the base station is dynamic virtual clusters. The virtual clusters are clusters of fire hazard rating or intensity mapped onto the exact physical node locations as shown in Figure 8.6. The virtual clusters provide a spatial map of the hazard distribution as an overlay to the physical nodes clusters. The virtual cluster is, in effect, a fire hazard situation distribution and the associated sensors. The virtual cluster viewed periodically can provide fire hazard situation dynamics such as speed and direction of hazard movements. The virtual cluster information can further be utilized to reconfigure the physical cluster for better and more efficient WSN resource utilization.

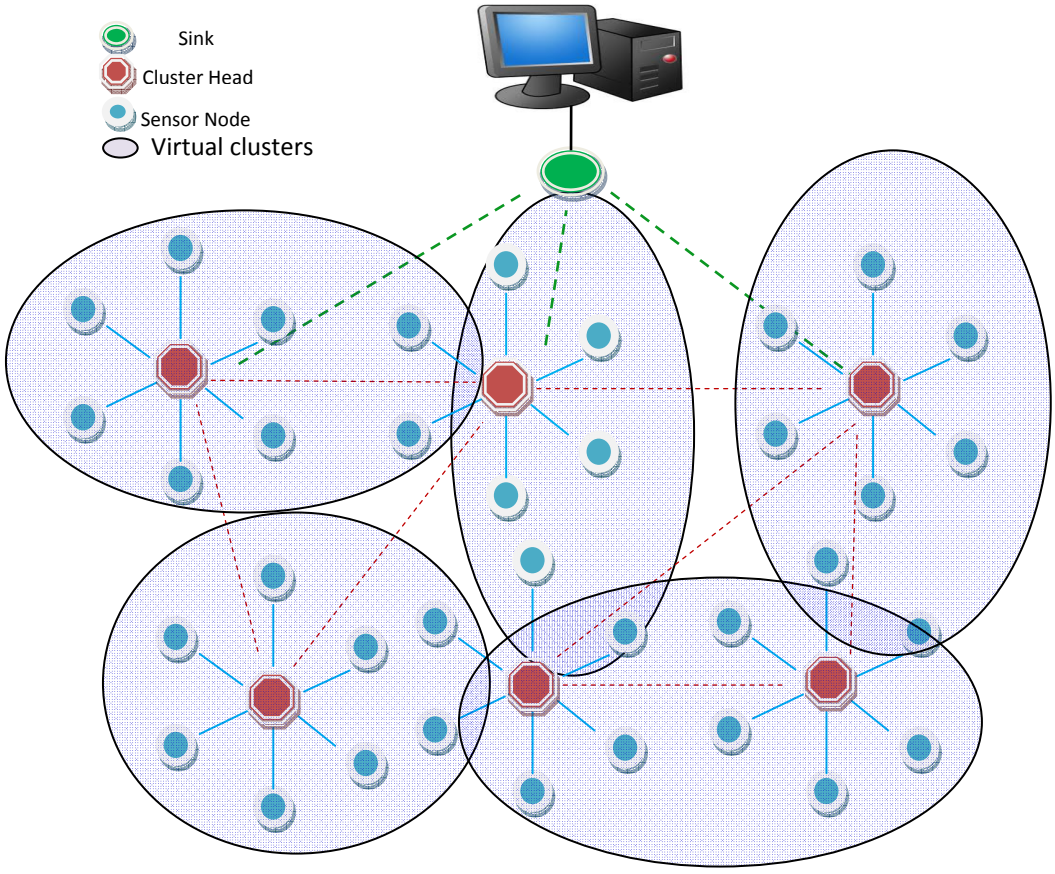


Figure 8.6: Virtual clusters of fire hazard rating or intensity mapped onto the node locations.

8.8. THE Micro-scale FWI SYSTEM DATA MODEL

Data sourced by the sensor nodes is modeled as a data stream. A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of tuples. It is not possible to control the order in which tuples arrive, nor feasible to locally store the stream in its entirety [177]. A tuple is similar to a row in a database table. Each tuple in a stream has fields which contain payload, ID, data type, size, etc. The weather data stream of interest for this work consists of air temperature (T), relative humidity (Rh), wind speed (V), and rainfall or precipitation (Rf) fields. A stream of weather tuples is shown in Figure 8.7 below.

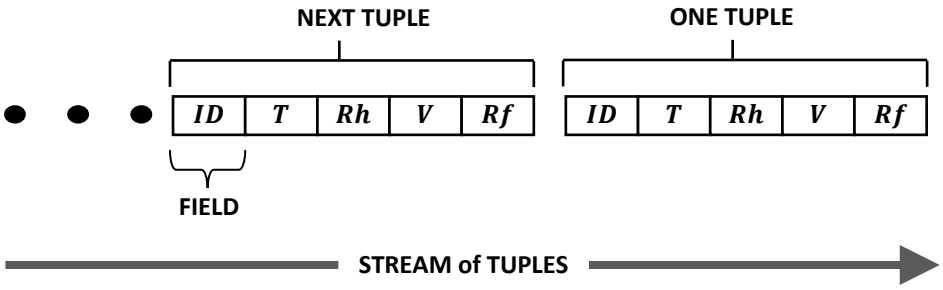


Figure 8.7: The stream of weather tuples.

The time difference between generation of a tuple and the next tuple in a stream determines the speed of the stream. The lower the time difference the higher is the speed of the stream. In this context, a weather data stream W is an unbounded sequence of elements $\langle w, t \rangle$, where w is weather data, and t is a monotonically increasing timestamp indicating the arrival time of the elements. w is a vector of weather values, $w = (T, Rh, V, Rf)$.

8.9. THE Micro-scale FWI SYSTEM SIMULATION MODEL

The Micro-scale FWI system model is developed using the MATLAB environment. The three different types of weather data processing model objects are created using MATLAB representing weather sensor node, cluster head, and sink sub-tasks. The modular algorithms running within these objects describing the FWI indices computation and SUBFCM clustering are also programmed using the MATLAB script language.

TrueTime 2.0 beta 6 is used to model and simulate the fire weather network. TrueTime is a MATLAB/SIMULINK based network modeling and simulation tool that provides a customizable Kernel model, analogue input model, wireless network protocol model, location, and battery models. For the purpose of this study, the Kernel model is customized to represent the Texas Instruments (TI) MSP430F2274 Microcontroller used to host the weather data stream mining application. The wireless network model is also customized to represent Chipcon's CC2530 IEEE 802.15.4 radio transceiver along with the ZigBee wireless protocol stack. The analogue input model is used to represent temperature, relative humidity, precipitation, and wind speed sensor inputs. The sensor nodes battery model is made to represent two standard AAA size alkaline batteries, which normally power the physical TI's MSP430 and Chipcon's CC2530 radio transceiver. The Kernel model further provides a simple interface for algorithms and models developed in MATLAB or Simulink to run on. The location of each node can also be set through the x and y inputs or can be interfaced to GPS modules for dynamic localization. However, for this purpose we determined the locations as no mobile nodes are considered. The simulation parameter settings are shown in Table 8.2 below. The TrueTime models of Sensor node, Cluster head, and Sink are shown in Figures 8.8, 8.9 and 8.10 below.

Table 8.2: The simulation model parameter settings.

Model parameter	Value	Model parameter	Value
Network type	ZigBee	Receiver signal strength (dBm)	-85
Network number	1	Path-loss exponent	3.5
Number of nodes	6	ACK timeout (sec)	.0004
Data rate (bits/s)	250 000	Retry limit	5
Minimum frame size (bits)	16	Error coding threshold	0.03
Transmit power (dBm)	0		

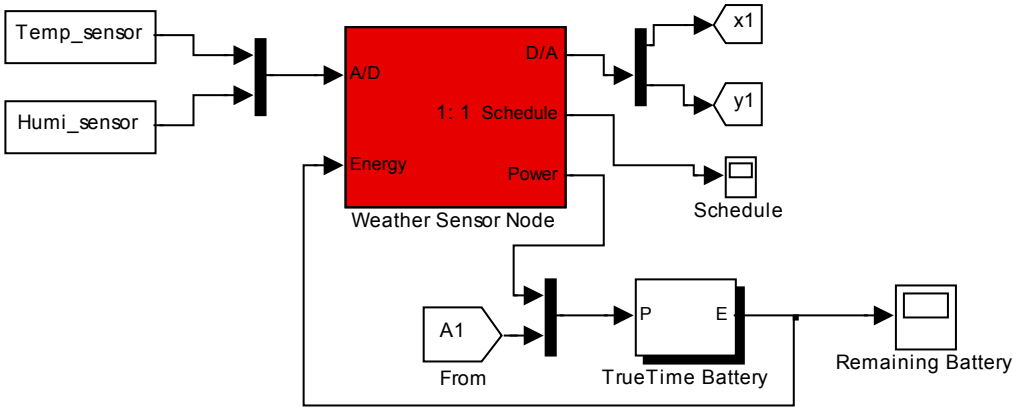


Figure 8.8: The Micro-scale FWI system weather sensor node sub-model.

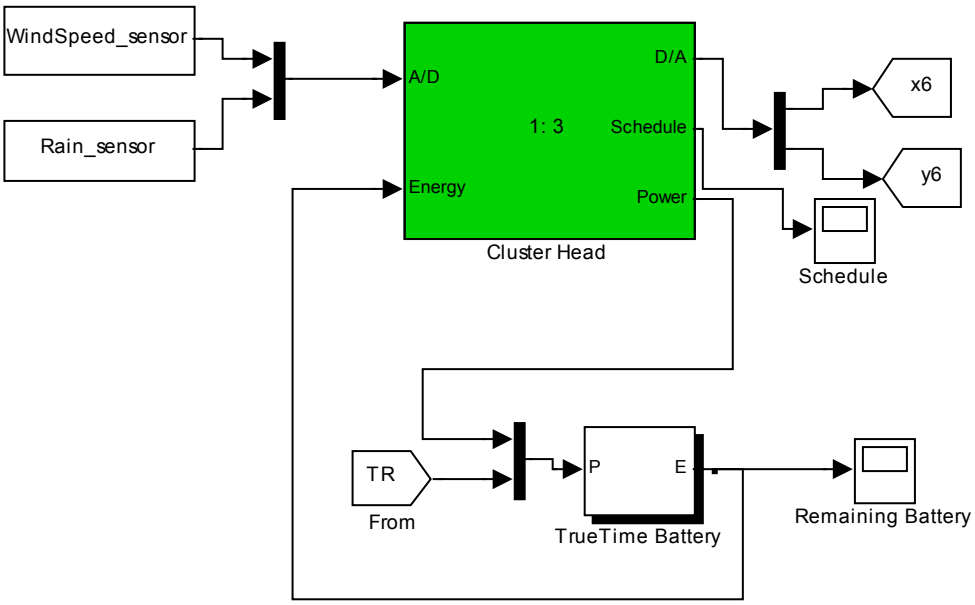


Figure 8.9: The Micro-scale FWI system cluster head sub-model.

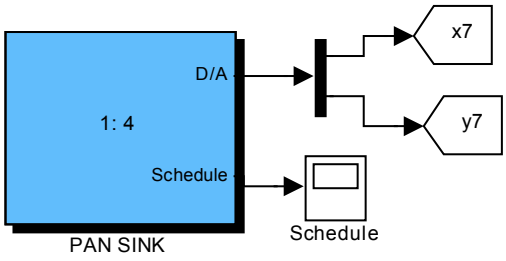


Figure 8.10: The Micro-scale FWI system sink sub-model.

8.10. SIMULATIONS AND RESULTS

8.10.1. Micro-scale FWI System Model Validation

System Model validation is an essential part of the model development process if the model is to be adopted and tried in a real-world setting. Model validation involves ensuring that the model meets its intended requirements in terms of the methods employed and the results obtained. The Micro-scale FWI system model is validated based on real world FWI system setup data. Real weather network data set containing all critical cases that the real system could exhibit, is used in the validation of the Model.

Real weather data sets recorded at several meteorology stations (Darfield, Ashburton, Burnham, Aero, and Snowdon) in South Island, New Zealand, obtained from the National Institute of Water and Atmospheric Research (NIWA) are used as benchmarks to validate the system performance. The geographic locations of the five stations are shown in Figure 8.11. These data sets consist of 3653 records of weather parameters (Temperature, relative humidity, Rainfall and Wind speed) along with their corresponding Canadian FWI indices (FFMC, DMC, DC, ISI, BUI, and FWI) collected hourly for the years 1994 to 2004. The data contains a burst mode weather stream as each element is acquired periodically at one hour time intervals. However, for simulations and for the purpose of model validation it is treated as a continuously incoming weather data stream rather than the burst mode, the data is fed to the sensor nodes in shorter periods.

The Micro-scale FWI system model representing the Canterbury weather network, shown in Figure 8.12, is configured and fed the real data sets from these stations. The model computed the fire danger ratings in-network in a distributed incremental fashion and logged results at the base station. The Model generated daily fire danger ratings that are then analyzed in comparison to those of the actual Canterbury weather network.

Initially some part of the real wild fire case data is fed to the model to verify model sanity. The simulation scenario comprises distributed sensor nodes collecting weather data sets and computing FFMC, DMC, and DC indices to be sent to their

cluster heads. The cluster heads compute ISI and BUI indices for each data set received and forward the aggregated results to the network sink. The simulation runs for 1960 seconds, whereby sensor nodes periodically transmit data packets every 20 seconds.

The Micro-scale model simulation results are in close agreement with the known actual FWI indices. The indices from both systems are plotted for about 100 real weather dataset records in Figures 8.13 to 8.18, where the fire predictions are processed in-network and transmitted to the sink every 20 seconds. Figure 8.13 shows a plot of the wildfire FFMC index produced by the model along with the corresponding known FFMC index. Figure 8.14 shows a sample plot of the wildfire DMC index produced by the model along with the corresponding known DMC index. Figure 8.15 shows a sample plot of the wildfire DC index produced by the model along with the corresponding known DC index. Figure 8.16 shows a sample plot of the wildfire ISI index produced by the model along with the corresponding known ISI index. Figure 8.17 shows a sample plot of the wildfire BUI index produced by the model along with the corresponding known BUI index. Figure 8.18 shows a sample plot of the wildfire FWI index produced by the model along with the corresponding known FWI index. From the wildfire hazard prediction point of view all the Micro-scale FWI model indices errors are insignificant. Figure 8.15 shows the highest variation between the two systems. This is because the DC index reflects the longest term fuel drying and the Micro-scale model is computing very frequent (hourly) fuel drying effect. Analyzing the general error thresholds produced by the Micro-scale FWI model, we can see that no fire situation is wrongly classified.

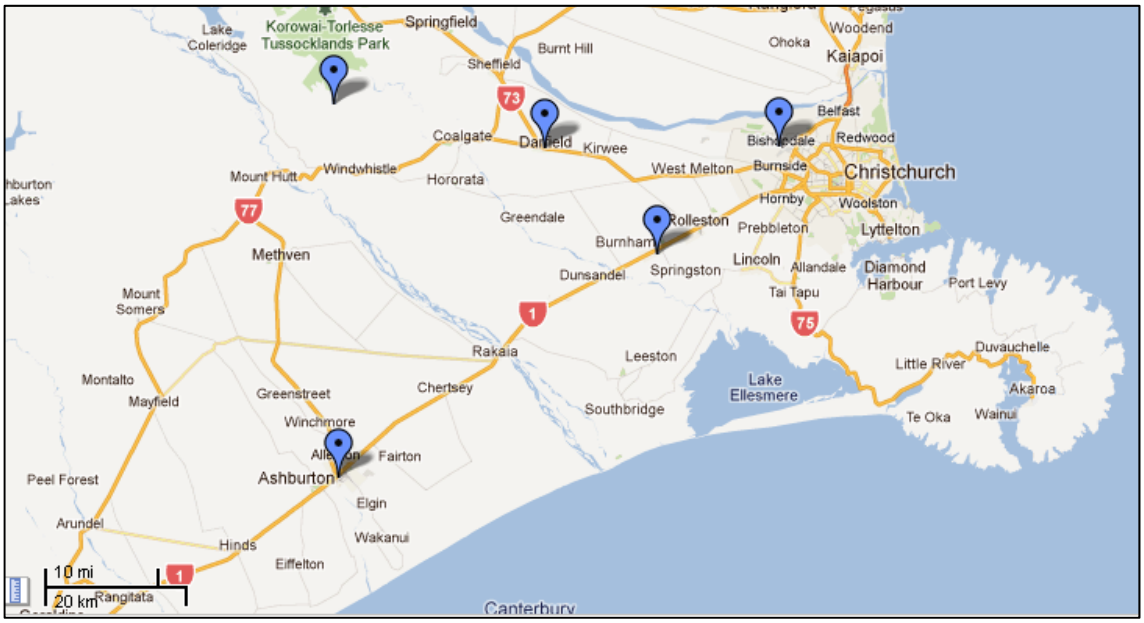


Figure 8.11: The Canterbury weather network map.

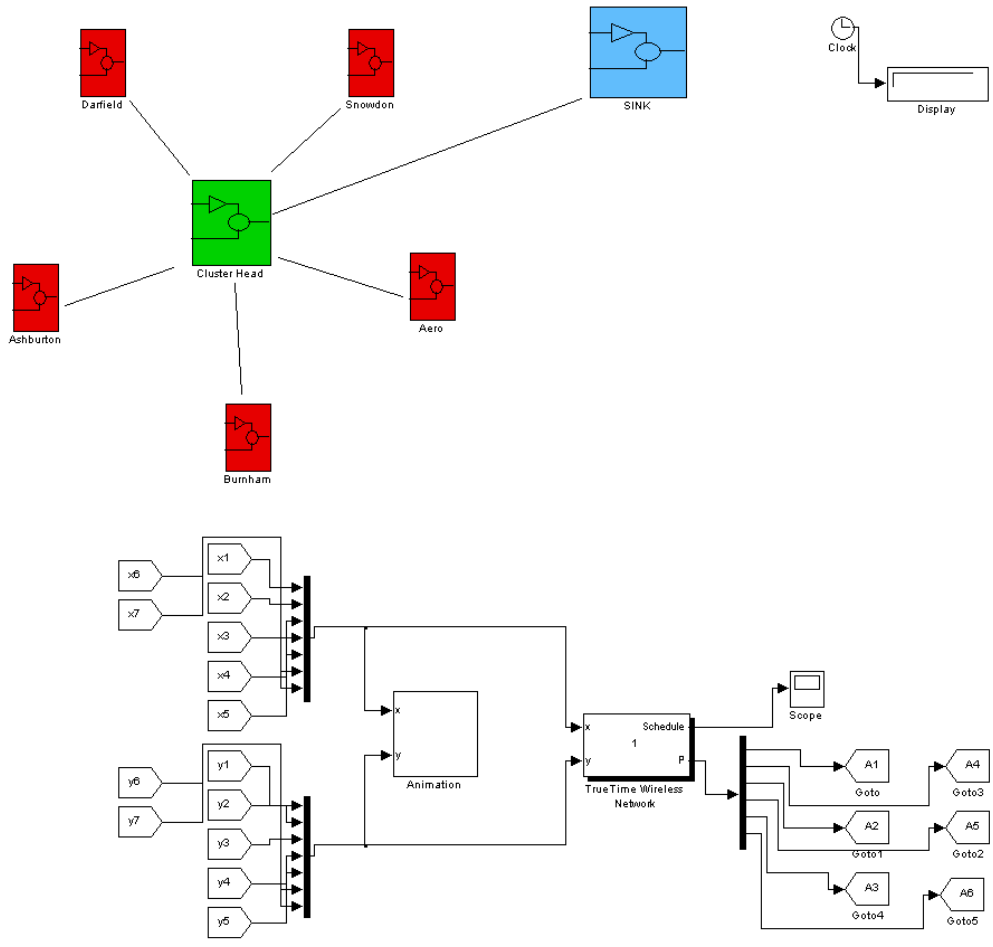


Figure 8.12: The Micro-scale FWI system model representing the Canterbury weather network.

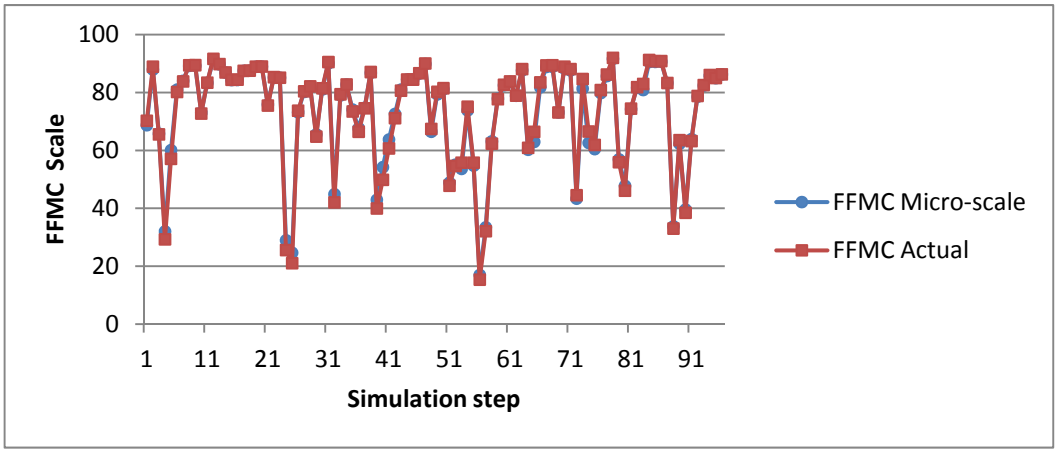


Figure 8.13: Micro-scale against Actual FFMC comparison.

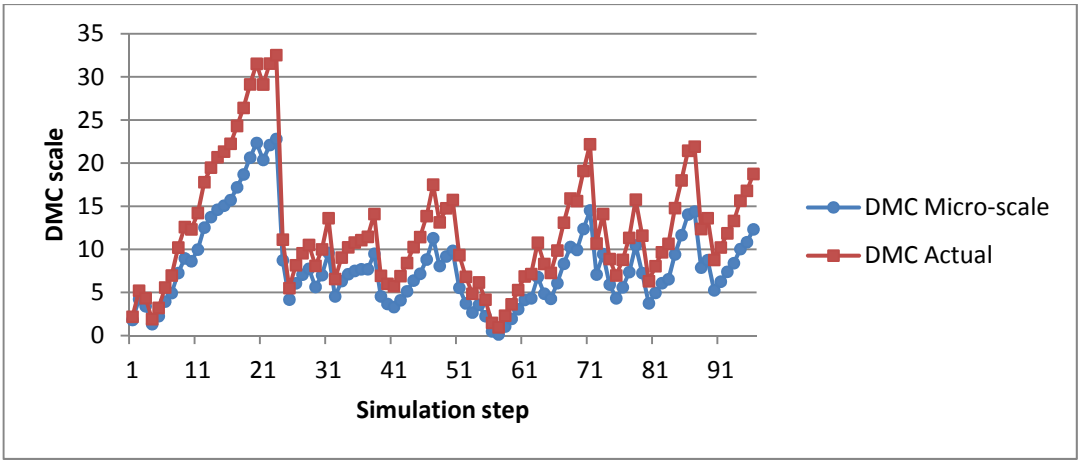


Figure 8.14: Micro-scale against Actual DMC comparison.

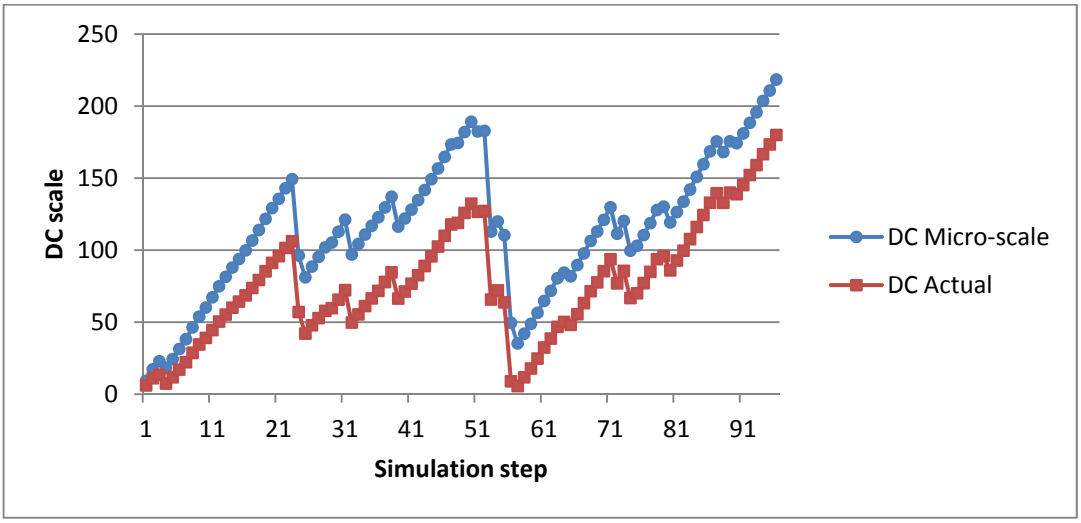


Figure 8.15: Micro-scale against Actual DC comparison.

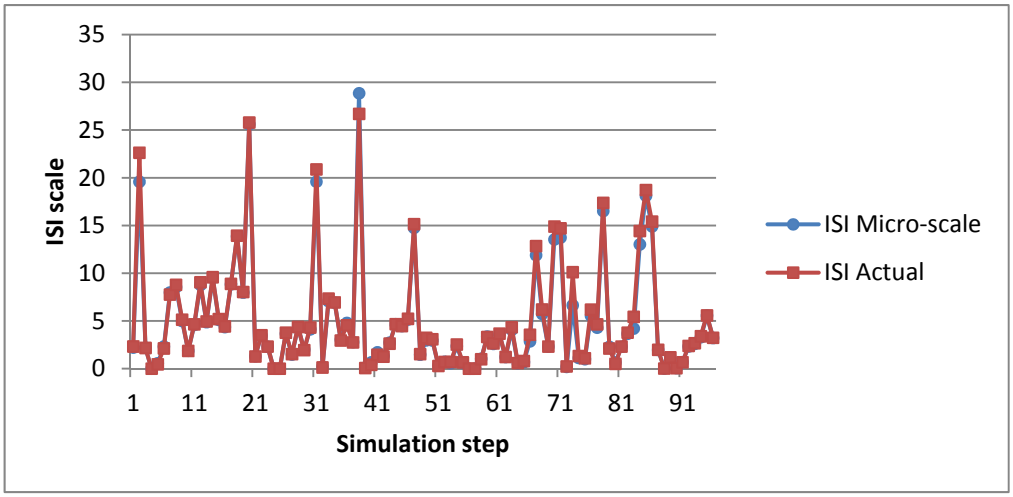


Figure 8.16: Micro-scale against Actual ISI comparison.

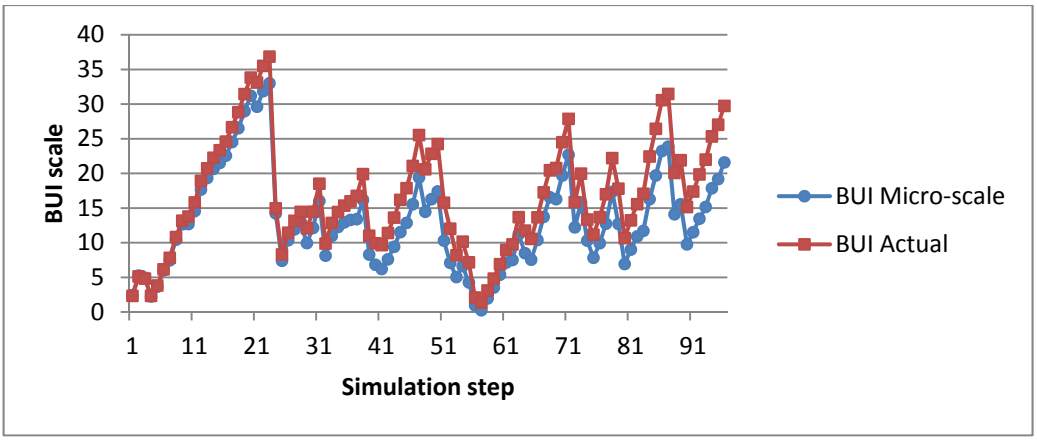


Figure 8.17: Micro-scale against Actual BUI comparison.

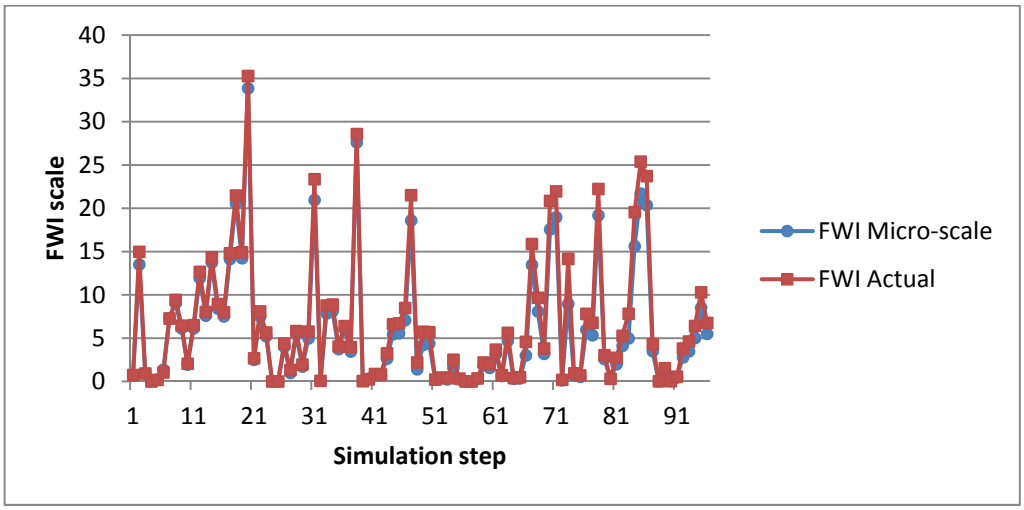


Figure 8.18: Micro-scale against Actual FWI comparison.

The main objective of further simulation is to try to find the model behavior such as end-to-end delays, packet loss characteristics, and energy efficiency.

8.10.2. End-to-End Delay

The average end-to-end delay of the Micro-scale model is shown in Figure 8.19 below. This is the sum of transmission, propagation, FWI indices processing and queuing delays. In this simulation, the sensor nodes transmit their partial prediction information every 60 seconds. The cluster heads process semi-prediction and queue packets for transmission as soon as the channel is available. 18 weather sensor nodes and 2 cluster head nodes competing for the wireless channel, the maximum delay observed is 10 seconds, with an average delay of 5.0747 seconds. This has been repeated for 54 weather sensor nodes and 6 cluster heads node and similar results have been observed as shown in figure 8.20.

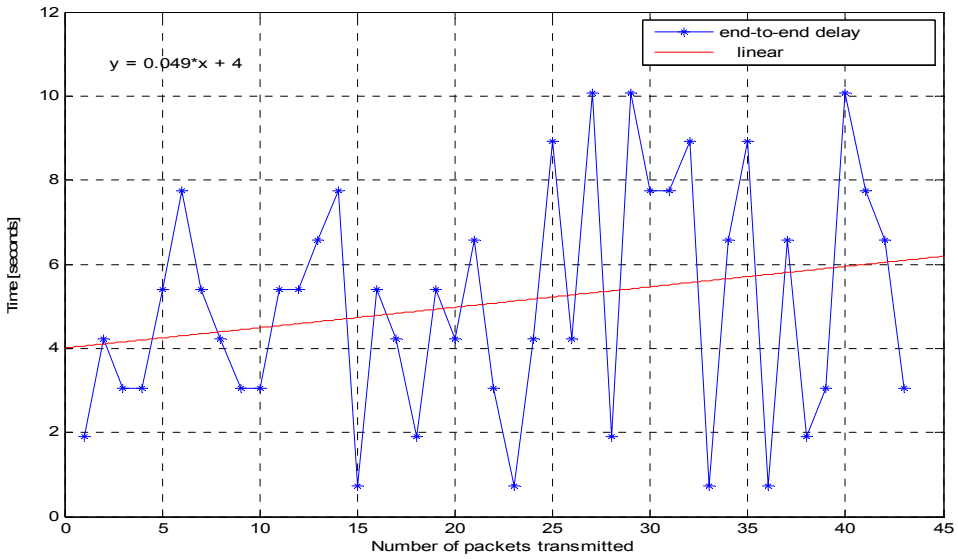


Figure 8.19: end-to-end delay of the model.

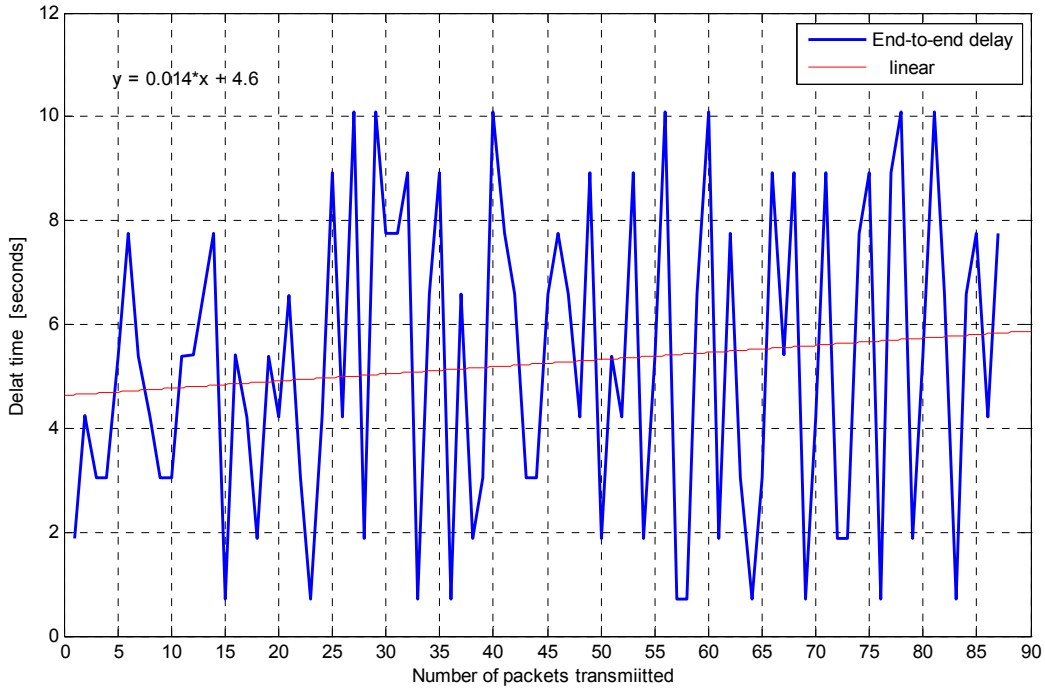


Figure 8.20: end-to-end delay of the model.

8.10.3. Packet Loss

The Micro-scale model is evaluated for the number of data packets lost under different sensors data transmit periods and whether the number of packets dropped has a significant impact on the hazard prediction results. The simulation scenario involves sensor nodes transmitting data packets with varying data transmit periods. The results obtained shown in Figure 8.21 show that the number of packets lost decreases exponentially as sensors transmit data packets less frequently for a network of 16, 54 and 90 weather sensor nodes. The prediction results obtained under this scenario indicate that data collision is not an issue at such network scales as long as the number of packet losses are not significantly high for a given cluster of sensor nodes. This also shows the fault tolerance of the Micro-scale model under the specific conditions.

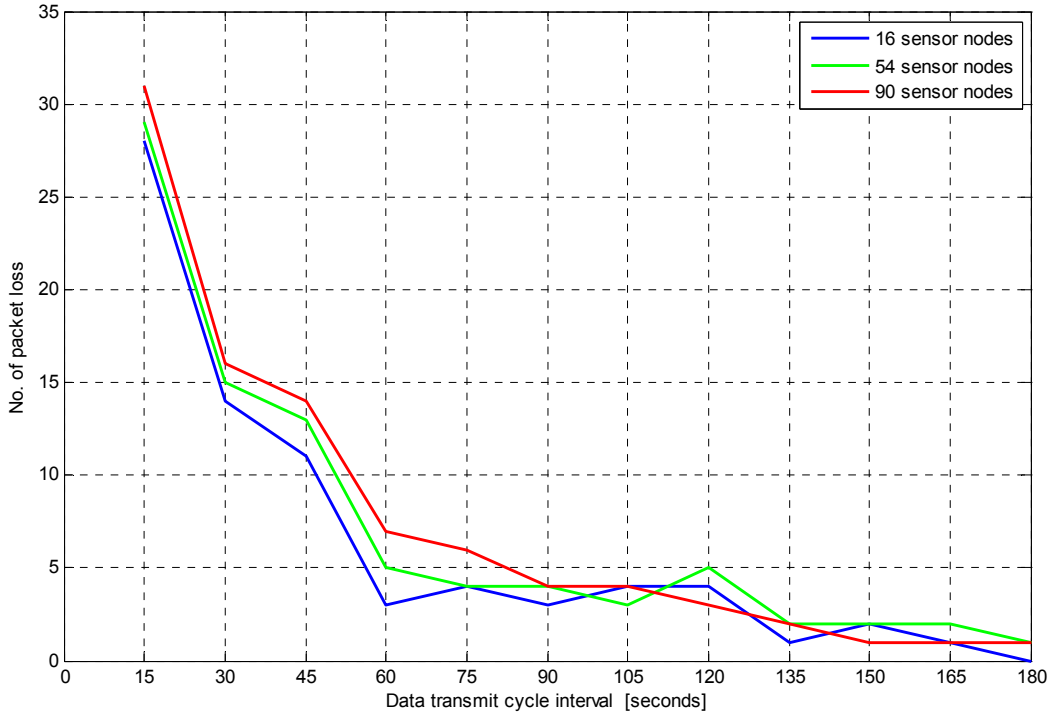


Figure 8.21: packet loss performance.

8.10.4. Energy Consumption

The model energy consumption is analyzed based on the TrueTime battery model. The battery model computes energy consumption due to kernel data processing, packet transmission/reception, and idle waiting consumption. The battery performance of the model is shown in Figure 8.22. The energy source is two AAA batteries of each 1200mAh (2*1200mAh). The figure shows a simulation of the Micro-scale application that continues to run until the remaining battery power is below 300mAh for both the weather sensor nodes and cluster head nodes.

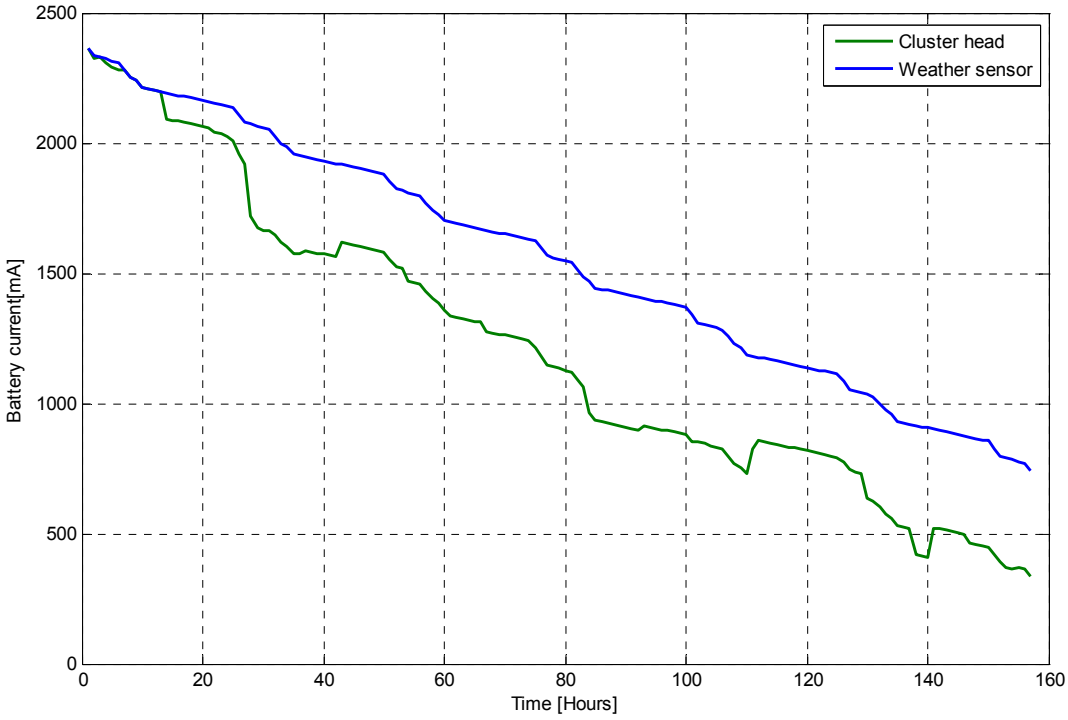


Figure 8.22: Remaining battery power of sensors and cluster heads.

8.11. CONCLUSIONS

This chapter presented a WSN based FWI system through TrueTime modeling and simulation software. The Micro-scale FWI model sanity has been verified through real wildfire datasets. Distributed in-network processing of FWI indices based on WSN has several advantages while producing similar results to a satellite communication based FWI system. The end-to-end delay, packet loss and energy consumption performance of the WSN model have been observed through simulations. The simulation results indicate that for a multi-tiered WSN architecture, the influence of end-to-end delay, energy consumption and packet loss on the FWI results are insignificant. This system provides a high spatial and temporal resolution wildfire hazard prediction system which is cost-effective, energy efficient, easily deployable for emergency situations and provides for user interaction.

Chapter 9

9. CONCLUSIONS AND FUTURE WORK

This chapter concludes the work that has been done to achieve the research objectives as specified in Chapter one and the solutions to the research problems and questions stated in the same chapter. Suggestions for possible advancement of the research, limitations, and future work are also discussed in this chapter.

9.1. CONCLUSIONS

The field of data stream mining in WSNs has seen considerable research interest recently. The development of resource efficient WSN design has been the central focus of most research in this field. The advancement of WSNs has made it possible to deploy low-cost networked sensors to address many distributed monitoring application challenges. As a result, WSNs are becoming increasingly appealing to data streaming applications. However, the resource constraints of WSNs and the resource demands by these applications have posed huge challenges for the research community. This thesis presents some of the challenging areas of the WSNs and distributed data stream mining.

The ultimate goal of this research is to develop an efficient distributed data stream mining framework for WSN systems. This is to address the resource constraint problem and demonstrate the real world adaptability of the framework using a case

study. The perceived framework integrates autonomous cluster based data stream techniques and wireless network architecture. The objectives are to address the problems of WSN energy constraints, network lifetime, and distributed mining of data streaming. The process requirement for a relevant case study from both spatial and temporal coverage resolution needs have also been considered.

This research has investigated the possibilities of employing WSNs for mining distributed data streams “on the fly” and extract useful information. The research has taken advantage of the distributed-architecture nature of the WSNs and the individual data processing capabilities of their nodes to implement the distributed data stream mining framework.

This thesis has formulated a lightweight autonomous data clustering algorithm called SUBFCM. The SUBFCM algorithm remains embedded within the individual nodes to analyze the locally generated streams ‘on the fly’ in cooperation with a group of nodes. The simulation results suggest that a SUBFCM algorithm can autonomously cluster streaming data and produce results comparable to standard batch clustering algorithms such as K-Means and Fuzzy C-Means algorithms. Simulation results have also highlighted the capability of the SUBFCM algorithm to incrementally produce very good approximate clustering results on the fly while proportionally utilizing the system resource. This capability of the algorithm has been the core driver that has enabled multiple local nodes stream mining tasks and hence has greatly minimized the quantity of data that has to be transmitted. In contrast to K-Means and FCM, the SUBFCM algorithm does not require prior knowledge of the number of clusters within the dataset and can function autonomously.

The thesis has studied the effects of data stream characteristics such as stream dimensions or feature spaces and stream periods or data flow rates. The study has also covered the effects of the network architecture such as node density per cluster (both uniform clusters and non-uniform clusters) on the overall performance of the SUBFCM. These studies have concluded that WSNs can provide good quality of service (QoS) feasible for online distributed incremental data stream mining applications.

The integrated distributed incremental data stream mining WSN framework has showed that its mining results are not significantly affected by the dimensionality of the streams and that it is stream dimension scalable. In the simulations, the average mining results deviation from the benchmark remained below 12% for as high as four dimensional streams.

The stream rate effect analysis has shown that the average cluster deviations increased smoothly with increasing stream rate. The streams with high dimension have higher cluster deviations at very fast stream rates. However, the average cluster deviations have never exceeded 10.13%.

The network architecture is an important factor in mining results quality and should be designed carefully to optimally utilize the basic concept of the distributed incremental data stream mining framework. For both uniform and non-uniform cluster densities, the effect on the quality of the mining results is significant only when mining very fast streams. The tolerated approximation error bounds, determined by the local model drift threshold parameter, plays a significant role in the cluster density effect.

The thesis has analyzed the quality of service, or a certain guaranteed level of performance that the WSN architecture can provide to applications utilizing the framework. The research has considered the average energy consumption, average data delivery delay, and packet delivery ratio of the proposed framework.

The average energy consumption of the framework largely varies with the stream rate and node density per cluster. Simulation results show that an average energy consumption of as low as 0.1 milliJoule per stream period can be achieved for 40 nodes per cluster at a stream period of 10 seconds or longer. However, the average energy consumption can rise to 1.3 milliJoule per stream period when the stream period is decreased to 1 second.

It is observed that the average data delivery delay generally increases as the number of nodes per cluster increases, which is obviously due to the increased packet colliusions and retransmissions. Also stream periods shorter than the average data delivery delay saturate the system with datasets and eventually cause loss of coordination among the nodes.

The stream periods have significant impact on the rate at which the packet delivery ratio drops as the node density increases. This indicates that for an optimal performance, applications utilizing the framework should determine the cluster density if certain packet delivery is desired. Hence, the thesis concludes that the overall mining quality is directly related to the combined effect of the stream characteristics, the network architecture, and the desired performance measures.

The thesis has also developed a novel high spatiotemporal resolution version of the standard Canadian fire weather index (FWI) system called the Micro-scale FWI based on the distributed incremental data stream mining framework. Simulations on real weather datasets indicate that the Micro-scale FWI implemented based on the framework can closely approximate the results obtained from the Standard FWI system while providing highly superior spatial and temporal information. This can offer direct local and global interaction with the few meter square space as against the tens of kilometers square of the present systems.

9.3. FUTURE WORK

While this study delivered promising results within the particular goals and framework, further study should explore other dimensions to extending performance, scope, and application of this research. The following is a list of suggestions for future work in the research:

1. **Synchronized WSN:** Future research developments should investigate the framework in a time-synchronized WSN architecture.

2. **Combination of Mining Strategies**: This study explored one of the most prominent stream mining techniques, namely, stream clustering. There are several techniques discussed in the literature which are adopted for WSN requirements. Combining more than one stream mining technique at different levels and exploring their potential in distributed mining frameworks are recommended.

3. **Mobile Nodes**: This study considers stationary WSN nodes to perform the distributed stream mining task. Especially with the advent of ubiquitous computing applications using mobile networks, this framework will find prominent applications in

mobile computing. We therefore recommend extending this framework using mobile nodes. This may extend the concept such that existing mobile nodes relevant to other applications may contribute to the process. Applications like wildlife monitoring and that of bushfire may be one of the most relevant associated ones. Here the nodes used for monitoring animal movement may also contribute to the Bush fire hazard condition monitoring.

4. Platform: This study has focused on a very low power embedded microcontroller based design, specifically the MSP430 family of microcontrollers from Texas Instruments. These low power microcontrollers are designed for very limited computational power requiring control applications. The main advantage of these microcontrollers is that they allow very low power sleep modes. Most WSN applications take advantage of these low power modes by switching off the microcontroller and associated peripherals when not performing any operations. However, due to their low computational capabilities, these controllers take a long time when engaged in intensive computations; thereby, extending the time they remain in full power mode and consequently consuming more power. To this end, we recommend exploring the use of Advanced RISK Machines (ARM) based microcontrollers such as that used in smart phone technologies. The ARM based microcontrollers boast more computational capability compared to the microcontrollers currently in use in WSNs and they further include low power sleep modes.

9.3.1. LIMITATIONS

This thesis has investigated the distributed incremental data stream mining WSN framework basing on hierarchically clustered sensor nodes. Even though the nodes compute parallel cooperating tasks, they are not time synchronized. The cooperative tasks that are computed at distributed nodes introduce slight delays between results in order to produce the final results.

References

[1] S. Misra, M. Reisslein, and X. Guoliang, "A survey of multimedia streaming in wireless sensor networks," *Communications Surveys & Tutorials, IEEE,* vol. 10, pp. 18-39, 2008.

[2] C. Song, "Mining and visualising wireless sensor network data," *Int. J. Sen. Netw.,* vol. 2, pp. 350-357, 2007.

[3] V. Cantoni, L. Lombardi, and P. Lombardi, "Challenges for Data Mining in Distributed Sensor Networks," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, 2006, pp. 1000-1007.

[4] A. Kulakov and D. Davcev, "Data mining in wireless sensor networks based on artificial neural-networks algorithms," in *Workshop on Data Mining in Sensor Networks in conjunction with 2005 SIAM International Conference on Data-Mining*, Newport Beach, CA, USA, 2005, pp. 10-16.

[5] S. M. McConnell and D. B. Skillicorn, "A distributed approach for prediction in sensor networks," in *Workshop on Data Mining in Sensor Networks in conjuction with 2005 SIAM International Conference on Data-Mining*, NewPort Beach, CA, USA, 2005.

[6] G. Bontempi and Y. L. Borgne, "An adaptive modular approach to the mining of sensor network data," in *Workshop on Data Mining in Sensor Networks in conjunction with 2005 SIAM International Conference on Data-Mining*, Newport Beach, CA, USA, 2005.

[7] A. A. Hatim, M. H. Alaaeldin, and M. R. A. Ghazy, "An efficient stream mining technique," *WSEAS Trans. Info. Sci. and App.,* vol. 5, pp. 1272-1281, 2008.

[8] M. Kholghi and M. Keyvanpour, "An analytical framework for data stream mining techniques based on challenges and requirements," *International Journal of Engineering Science and Technology (IJEST),* vol. 3, pp. 2507-2513, March 2011.

[9] H. Jiawei and F. Yongjian, "Discovery of Multiple-Level Association Rules from Large Databases," in *Proceedings of the 21th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 1995.

[10] S. Ramakrishnan and A. Rakesh, "Mining Generalized Association Rules," in *Proceedings of the 21th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 1995.

[11] O. Zhenzheng, W. Quanyuan, and W. Tao, "An Efficient Decision Tree Classification Method Based on Extended Hash Table for Data Streams Mining," in *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, 2008, pp. 313-317.

[12] S.Prasanna and S. Rao, "An Overview of Wireless Sensor Networks Applications and Security," *International Journal of Soft Computing and Engineering (IJSCE),* vol. 2, pp. 538-540, 2012.

[13] A. Bifet and R. Kirkby, "Data stream mining: a practical approach," The University of Waikato, 2009.

[14] S. Ullah, H. Higgins, B. Braem, B. Latre, C. Blondia, I. Moerman, S. Saleem, Z. Rahman, and K. S. Kwak, "A Comprehensive Survey of Wireless Body Area Networks: On PHY, MAC, and Network Layers Solution," *Journal of Medical Systems,* vol. 36, pp. 1065-1094, 2012.

[15] B. O'Flyrm, R. Martinez, J. Cleary, C. Slater, F. Regan, D. Diamond, and H. Murphy, "SmartCoast: A Wireless Sensor Network for Water Quality Monitoring," in *Local Computer Networks, 2007. LCN 2007. 32nd IEEE Conference on*, 2007, pp. 815-816.

[16] T. Kotsilieris and G. T. Karetsos, "A Mobile Agent Enabled Wireless Sensor Network for River Water Monitoring," in *Wireless and Mobile Communications, 2008. ICWMC '08. The Fourth International Conference on*, 2008, pp. 346-351.

[17] S. Subana, G. Akbar, and S. Philip, "Sensor data acquisition for climate change modeling," *WSEAS Trans. Cir. and Sys.,* vol. 7, pp. 942-952, 2008.

[18] K. Lakshman, A. Robert, B. Phil, C. Jasmeet, F. Mick, K. Nandakishore, N. Lama, and Y. Mark, "Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea," in *Proceedings of the 3rd international conference on Embedded networked sensor systems* San Diego, California, USA: ACM, 2005.

[19] B. Lu, T. G. Habetler, R. G. Harley, and J. A. Gutierrez, "Applying wireless sensor networks in industrial plant energy management systems. Part I. A closed-loop scheme," in *Sensors, 2005 IEEE*, 2005, p. 6 pp.

[20] L. Bin, T. G. Habetler, R. G. Harley, and J. A. Gutierrez, "Applying wireless sensor networks in industrial plant energy management systems. Part II. Design of sensor devices," in *Sensors, 2005 IEEE*, 2005, p. 6 pp.

[21] R. Jafari, A. Encarnacao, A. Zahoory, F. Dabiri, H. Noshadi, and M. Sarrafzadeh, "Wireless sensor networks for health monitoring," in *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, 2005, pp. 479-481.

[22] E. Lawrence, K. F. Navarro, D. Hoang, and Y. Y. Lim, "Data Collection, Correlation and Dissemination of Medical Sensor Information in a WSN," in *Networking and Services, 2009. ICNS '09. Fifth International Conference on*, 2009, pp. 402-408.

[23] W. Dai-Hua and L. Wei-Hsin, "Wireless transmission for health monitoring of large structures," *Instrumentation and Measurement, IEEE Transactions on,* vol. 55, pp. 972-981, 2006.

[24] S. Kundu, S. Roy, and A. Pal, "A power-aware wireless sensor network based bridge monitoring system," in *Networks, 2008. ICON 2008. 16th IEEE International Conference on*, 2008, pp. 1-7.

[25] K. Chintalapudi, J. Paek, O. Gnawali, T. S. Fu, K. Dantu, J. Caffrey, R. Govindan, E. Johnson, and S. Masri, "Structural damage detection and localization using NETSHM," in *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, 2006, pp. 475-482.

[26] T. Antoine-Santoni, J. F. Santucci, E. de Gentili, and B. Costa, "Using Wireless Sensor Network for Wildfire detection. A discrete event approach of environmental monitoring tool," in *Environment Identities and Mediterranean Area, 2006. ISEIMA '06. First international Symposium on*, 2006, pp. 115-120.

[27] V. Raghunathan, C. Schurgers, S. Park, M. Srivastava, and B. Shaw, "Energy-Aware Wireless Microsensor Networks," in *IEEE Signal Processing Magazine*, 2002, pp. 40-50.

[28] K. Holger and W. Andreas, *Protocols and Architectures for Wireless Sensor Networks*: John Wiley & Sons, 2005.

[29] A. M. Jafari and W. Lang, "Optimal Sample Rate for Wireless Sensor Actuator Network," *IAENG International Journal of Computer Science,* vol. 36, pp. 387-393 2009.

[30] V. Raghunathan, S. Ganeriwal, and M. Srivastava, "Emerging techniques for long lived wireless sensor networks," *Communications Magazine, IEEE,* vol. 44, pp. 108-114, 2006.

[31] S. Panichpapiboon, G. Ferrari, and O. K. Tonguz, "Optimal Transmit Power in Wireless Sensor Networks," *Mobile Computing, IEEE Transactions on,* vol. 5, pp. 1432-1447, 2006.

[32] S. Narayanaswamy, V. Kawadia, R. S. Sreenivas, and P. R. Kumar, "Power Control in Ad-Hoc Networks: Theory, Architecture, Algorithm and Implementation of the COMPOW Protocol," in *in European Wireless Conference*, 2002, pp. 156-162.

[33] S. Agarwal, R. H. Katz, S. V. Krishnamurthy, and S. K. Dao, "Distributed power control in ad-hoc wireless networks," in *Personal, Indoor and Mobile Radio Communications, 2001 12th IEEE International Symposium on*, 2001, pp. F-59-F-66 vol.2.

[34] A. Nandi and S. Kundu, "Optimal transmit power and energy level performance of random WSN in Rayleigh fading channel," in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on*, pp. 556-561.

[35] G. Yu and H. Tian, "Data forwarding in extremely low duty-cycle sensor networks with unreliable communication links," in *Proceedings of the 5th international conference on Embedded networked sensor systems* Sydney, Australia: ACM, 2007.

[36] R. R. Brooks, "Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems," *International Journal of Distributed Sensor Networks,* vol. 4, pp. 369-369, 2012/08/02 2008.

[37] J. T. Adams, "An introduction to IEEE STD 802.15.4," in *Aerospace Conference, 2006 IEEE*, 2006, p. 8 pp.

[38] M. Geoff, "The 6LoWPAN architecture," in *Proceedings of the 4th workshop on Embedded networked sensors* Cork, Ireland: ACM, 2007.

[39] Z. Shelby and C. Bormann, "6LoWPAN: The wireless embedded internet - part 1 " in *EE Times*, 2011.

[40] F. Liu, K. Xing, X. Cheng, S. Rotenstreich, M. Cardei, I. Cardei, and D.-Z. Du, "Energy-Efficient MAC Layer protocols in Ad Hoc Networks" Resource Management in Wireless Networking." vol. 16, D.-Z. Du and C. Raghavendra, Eds.: Springer US, 2005, pp. 300-341.

[41] C. Francesca, C. Emanuele, and A. Anna, "Performance analysis of IEEE 802.15.4 wireless sensor networks: An insight into the topology formation process," *Comput. Netw.,* vol. 53, pp. 3057-3075, 2009.

[42] B. Vaduvur, D. Alan, S. Scott, and Z. Lixia, "MACAW: a media access protocol for wireless LAN's," in *Proceedings of the conference on Communications architectures, protocols and applications* London, United Kingdom: ACM, 1994.

[43] D. Tijs van and L. Koen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems* Los Angeles, California, USA: ACM, 2003.

[44] D. Jin, K. Sivalingam, R. Kashyapa, and C. Lu Jian, "A multi-layered architecture and protocols for large-scale wireless sensor networks," in *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, 2003, pp. 1443-1447 Vol.3.

[45] P. Karn, "MACA a new channel access method for packet radio," in *Computer Networking Conference*, 1990, pp. 134-140.

[46] L. Koen and H. Gertjan, "Energy-efficient medium access control," in *Embedded systems hand book*, C. press, Ed., 2005.

[47] H. Wendi Rabiner, C. Anantha, and B. Hari, "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*: IEEE Computer Society, 2000.

[48] K. T. Kim and H. Y. Youn, "PEACH: Proxy-Enable Adaptive Clustering Hierarchy for Wireless Sensor network," in *The 2005 International Conference On Wireless Network*, 2005, pp. 52-57.

[49] J. Kyu-Tae and C. Dong-Ho, "A new MAC algorithm based on reservation and scheduling for energy-limited ad-hoc networks," *Consumer Electronics, IEEE Transactions on,* vol. 49, pp. 135-141, 2003.

[50] N. Arastouie, M. Sabaei, and H. S. Shahreza, "A novel approach for trade-off between computation and communication cost in wireless sensor networks," in *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, pp. 820-825.

[51] S. K. Singh, M. P. Singh, and D. K. Singh, "Energy Efficient Homogenous Clustering Algorithm for Wireless Sensor Networks," *International Journal of Wireless & Mobile Networks (IJWMN),* vol. 2, pp. 49-61, 2010.

[52] T. Pham, K. Eun Jik, and M. Moh, "On data aggregation quality and energy efficiency of wireless sensor network protocols - extended summary," in *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, 2004, pp. 730-732.

[53] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables," Journal of the American Statistical Association, vol. 58, pp. 13-30, 1963.

[54] O. Maron and A. W. Moore, "Hoeffding races: Accelerating model selection search for classification and function approximation," in Advances in Neural Information Processing Systems, 1994, pp. 59-66.

[55] B. Brian, B. Shivnath, D. Mayur, M. Rajeev, and W. Jennifer, "Models and issues in data stream systems," in Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems Madison, Wisconsin: ACM, 2002.

[56] S. Muthukrishnan, "Data streams: algorithms and applications," in Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms Baltimore, Maryland: Society for Industrial and Applied Mathematics, 2003.

[57] S. Muthukrishnan, "Data Streams: Algorithms and Applications," Foundations and Trends® in Theoretical Computer Science, vol. 1, pp. 117-236, 2005.

[58] H. H. O. Nasereddin, "Stream Data Mining," *International Journal of Web Applications (IJWA),* vol. 3, pp. 90-97, 2011.

[59] M. Kholghi and M. Keyvanpour, "An analytical framework for data stream mining techniques based on challenges and requirements," International Journal of Engineering Science and Technology (IJEST), vol. 3, pp. 2507-2513, March 2011.

[60] G. Mohamed Medhat, Z. Arkady, and K. Shonali, "Mining data streams: a review," *SIGMOD Rec.,* vol. 34, pp. 18-26, 2005.

[61] B. G. Phillip and M. Yossi, "Synopsis data structures for massive data sets," in *External memory algorithms*, M. A. James and V. Jeffrey Scott, Eds.: American Mathematical Society, 1999, pp. 39-70.

[62] C. C. Aggarwal and P. S. Yu, "A Survey of Synopsis Construction in Data Streams.," in *Data Streams - Models and Algorithms*. vol. 31: Springer, 2007, pp. 169-207.

[63] D. Pedro and H. Geoff, "A General Method for Scaling Up Machine Learning Algorithms and its Application to Clustering," in *Proceedings of the Eighteenth International Conference on Machine Learning*: Morgan Kaufmann Publishers Inc., 2001.

[64] T. Nesime, U, ur, etintemel, Z. Stan, C. Mitch, and S. Michael, "Load shedding in a data stream manager," in *Proceedings of the 29th international conference on Very large data bases - Volume 29* Berlin, Germany: VLDB Endowment, 2003.

[65] S. Muthukrishnan, "Data streams: algorithms and applications," in *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms* Baltimore, Maryland: Society for Industrial and Applied Mathematics, 2003.

[66] C. G. Anna, K. Yannis, S. Muthukrishnan, and S. Martin, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," in *Proceedings of the 27th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 2001.muth

[67] C. A. Charu, H. Jiawei, W. Jianyong, and S. Y. Philip, "A framework for clustering evolving data streams," in *Proceedings of the 29th international conference on Very large data bases - Volume 29* Berlin, Germany: VLDB Endowment, 2003.

[68] M. Gurmeet Singh and M. Rajeev, "Approximate frequency counts over data streams," in *Proceedings of the 28th international conference on Very Large Data Bases* Hong Kong, China: VLDB Endowment, 2002.

[69] C. Yixin, D. Guozhu, H. Jiawei, W. W. Benjamin, and W. Jianyong, "Multi-dimensional regression analysis of time-series data streams," in *Proceedings of the 28th international conference on Very Large Data Bases* Hong Kong, China: VLDB Endowment, 2002.

[70] C. A. Charu, H. Jiawei, W. Jianyong, and S. Y. Philip, "A framework for projected clustering of high dimensional data streams," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* Toronto, Canada: VLDB Endowment, 2004.

[71] J. Gao, W. Fan, J. Han, and P. S. Yu, "A general framework for mining concept-drifting data streams with skewed distributions," in *2007 SIAM Int. Conf. Data Mining (SDM'07)*, Minneapolis, MN, April, 2007.

[72] A. Rakesh and S. Ramakrishnan, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 1994.

[73] J. Z. Mohammed, P. Srinivasan, O. Mitsunori, and L. Wei, "New Algorithms for Fast Discovery of Association Rules," University of Rochester 1997.

[74] H. Jiawei, P. Jian, and Y. Yiwen, "Mining frequent patterns without candidate generation," *SIGMOD Rec.,* vol. 29, pp. 1-12, 2000.

[75] H. Jiawei, P. Jian, and Y. Yiwen, "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* Dallas, Texas, United States: ACM, 2000.

[76] I. R. Mohammed, J. O. C. Martin, and K. D. Amar, "Computational Method for Temporal Pattern Discovery in Biomedical Genomic Databases," in *Proceedings of the 2005 IEEE Computational Systems Bioinformatics Conference*: IEEE Computer Society, 2005.

[77] S. Hou and X. Zhang, "Alarms Association Rules Based on Sequential Pattern Mining Algorithm," in *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, 2008, pp. 556-560.

[78] T. Syed Khairuzzaman, A. Chowdhury Farhan, J. Byeong-Soo, and L. Young-Koo, "Efficient frequent pattern mining over data streams," in *Proceedings of the 17th ACM conference on Information and knowledge management* Napa Valley, California, USA: ACM, 2008.

[79] C. K. S. Leung and Q. I. Khan, "DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams," in *Data Mining, 2006. ICDM '06. Sixth International Conference on*, 2006, pp. 928-932.

[80] L. Carson Kai-Sang and H. Boyu, "Mining of Frequent Itemsets from Streams of Uncertain Data," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*: IEEE Computer Society, 2009.

[81] L. Carson Kai-Sang and J. Fan, "Frequent pattern mining from time-fading streams of uncertain data," in *Proceedings of the 13th international conference on Data warehousing and knowledge discovery* Toulouse, France: Springer-Verlag 2011, pp. 252-264.

[82] L. Carson Kai-Sang and J. Fan, "Frequent itemset mining of uncertain data streams using the damped window model," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, TaiChung, Taiwan, 2011, pp. 950--955.

[83] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining Frequent Patterns in Data Streams at Multiple Time Granularities," in *Data Mining: Next Generation Challenges and Future Directions*: AAAI/MIT Press, 2004, ch. 6, 2002.

[84] Y. Jeffery Xu, C. Zhihong, L. Hongjun, and Z. Aoying, "False positive or false negative: mining frequent itemsets from high speed transactional data streams," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30* Toronto, Canada: VLDB Endowment, 2004.

[85] D. Pedro and H. Geoff, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* Boston, Massachusetts, United States: ACM, 2000.

[86] H. Geoff, S. Laurie, and D. Pedro, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* San Francisco, California: ACM, 2001.

[87] L. Feixiong and L. Quan, "An Improved Algorithm of Decision Trees for Streaming Data Based on VFDT," in *Proceedings of the 2008 International Symposium on Information Science and Engineering - Volume 01*: IEEE Computer Society, 2008.

[88] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*: Pearson Prentice Hall, 2007.

[89] S. Benson Edwin Raj and A. A. Portia, "Analysis on credit card fraud detection methods," in *Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on*, pp. 152-156.

[90] T. Siu-Keung and X. Liming, "Outlier Detection in Clinical Research," in *Encyclopedia of Biopharmaceutical Statistics*, pp. 933-939.

[91] C. Manuel, L. n, Joaqu, B. O. M. n, J. M. Francisco, P. nez De, A. n, P. V. G. Eliseo, and lez, "Outlier Detection and Data Cleaning in Multivariate Non-Normal Samples: The PAELLA Algorithm," *Data Min. Knowl. Discov.,* vol. 9, pp. 171-187, 2004.

[92] T. N. David, M. Gokhan, and C. Alok, "A reconfigurable architecture for network intrusion detection using principal component analysis," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays* Monterey, California, USA: ACM, 2006.

[93] Z. Jiang, L. Chang-Tien, and K. Yufeng, "Detecting region outliers in meteorological data," in *Proceedings of the 11th ACM international symposium on Advances in geographic information systems* New Orleans, Louisiana, USA: ACM, 2003.

[94] L. Anselin, "Local Indicators of Spatial Association—LISA," *Geographical Analysis,* vol. 27, pp. 93-115, 1995.

[95] V. Barnett and T. Lewis, *Outliers in Statistical Data*: John Wiley & Sons, 1994.

[96] R. Sridhar, R. Rajeev, and S. Kyuseok, "Efficient algorithms for mining outliers from large data sets," *SIGMOD Rec.,* vol. 29, pp. 427-438, 2000.

[97] R. Sridhar, R. Rajeev, and S. Kyuseok, "Efficient algorithms for mining outliers from large data sets," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* Dallas, Texas, United States: ACM, 2000.

[98] T. N. Raymond and H. Jiawei, "Efficient and Effective Clustering Methods for Spatial Data Mining," in *Proceedings of the 20th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 1994.

[99] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226-231.

[100] Z. Tian, R. Raghu, and L. Miron, "BIRCH: an efficient data clustering method for very large databases," *SIGMOD Rec.,* vol. 25, pp. 103-114, 1996.

[101] G. Sudipto, R. Rajeev, and S. Kyuseok, "CURE: an efficient clustering algorithm for large databases," *SIGMOD Rec.,* vol. 27, pp. 73-84, 1998.

[102] A. Fabrizio and F. Fabio, "Detecting distance-based outliers in streams of data," in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management* Lisbon, Portugal: ACM, 2007.

[103] B. Sabyasachi and M. Martin, "Automatic outlier detection for time series: an application to sensor data," *Knowl. Inf. Syst.,* vol. 11, pp. 137-154, 2007.

[104] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proceedings of the 32nd international conference on Very large data bases* Seoul, Korea: VLDB Endowment, 2006.

[105] D. l. Curiae, O. Banias, F. Dragan, C. Volosencu, and O. Dranga, "Malicious Node Detection in Wireless Sensor Networks Using an Autoregression Technique," in *Networking and Services, 2007. ICNS. Third International Conference on*, 2007, pp. 83-83.

[106] I. Kozue and K. Hiroyuki, "Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams," in *Proceedings of the 19th international conference on Database and Expert Systems Applications* Turin, Italy: Springer-Verlag, 2008.

[107] L. Duan, L. Xu, Y. Liu, and J. Lee, "Cluster-based outlier detection," *Annals of Operations Research,* vol. 168, pp. 151-168, 2009.

[108] T. N. Raymond and H. Jiawei, "Efficient and Effective Clustering Methods for Spatial Data Mining," in *Proceedings of the 20th International Conference on Very Large Data Bases*: Morgan Kaufmann Publishers Inc., 1994.

[109] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, 2000, pp. 359-366.

[110] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-data algorithms for high-quality clustering," in *Data Engineering, 2002. Proceedings. 18th International Conference on*, 2002, pp. 685-694.

[111] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *In 2006 SIAM Conference on Data Mining*, 2006, pp. 328-339.

[112] R. Carlos, M. Ernestina, and S. Myra, "C-DenStream: Using Domain Knowledge on a Data Stream," in *Proceedings of the 12th International Conference on Discovery Science* Porto, Portugal: Springer-Verlag, 2009.

[113] R. Jiadong and M. Ruiqing, "Density-Based Data Streams Clustering over Sliding Windows," in *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*, 2009, pp. 248-252.

[114] L.-x. Liu, J. Kang, Y.-f. Guo, and H. Huang, "A three-step clustering algorithm over an evolving data stream," in *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, 2009, pp. 160-164.

[115] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The ClusTree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems,* vol. 29, pp. 249-272.

[116] C. A. Charu, H. Jiawei, W. Jianyong, and S. Y. Philip, "A framework for clustering evolving data streams," in *Proceedings of the 29th international conference on Very large data bases - Volume 29* Berlin, Germany: VLDB Endowment, 2003.

[117] G. Sudipto, M. Adam, M. Nina, M. Rajeev, and O. C. Liadan, "Clustering Data Streams: Theory and Practice," *IEEE Trans. on Knowl. and Data Eng.,* vol. 15, pp. 515-528, 2003.

[118] Z. Tian, R. Raghu, and L. Miron, "BIRCH: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data* Montreal, Quebec, Canada: ACM, 1996.

[119] F. Doug, "Iterative optimization and simplification of hierarchical clusterings," *J. Artif. Int. Res.,* vol. 4, pp. 147-179, 1996.

[120] F. Brian and S. Mihaela Van Der, "A rules-based approach for configuring chains of classifiers in real-time stream mining systems," *EURASIP J. Adv. Signal Process,* vol. 2009, pp. 1-17, 2009.

[121] E. S. Robert, "A brief introduction to boosting," in *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2* Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999.

[122] Z. Yongluan, "Scalable and Adaptable Distributed Stream Processing," in *Data Engineering Workshops, 2006. Proceedings. 22nd International Conference on*, 2006, pp. x148-x148.

[123] B. Magdalena, B. Hari, M. Samuel, and S. Michael, "Fault-tolerance in the Borealis distributed stream processing system," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* Baltimore, Maryland: ACM, 2005.

[124] H. Maria, B. Yannis, and V. Michalis, "On Clustering Validation Techniques," *J. Intell. Inf. Syst.,* vol. 17, pp. 107-145, 2001.

[125] K. Matthias, L. Stefano, and M. Gianluca, "Distributed clustering based on sampling local density estimates," in *Proceedings of the 18th international joint conference on Artificial intelligence* Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003.

[126] S. Alexander and G. Joydeep, "Cluster ensembles --- a knowledge reuse framework for combining multiple partitions," *J. Mach. Learn. Res.,* vol. 3, pp. 583-617, 2003.

[127] B.-H. Park, H. Kargupta, and N. Ye, "Distributed Data Mining: Algorithms, Systems, and Applications," in *The handbook of data mining*: Routledge, 2003, pp. 341-358.

[128] D. Souptik, B. Kanishka, G. Chris, W. Ran, and K. Hillol, "Distributed Data Mining in Peer-to-Peer Networks," *IEEE Internet Computing,* vol. 10, pp. 18-26, 2006.

[129] G. Cormode, S. Muthukrishnan, and Z. Wei, "Conquering the Divide: Continuous Clustering of Distributed Data Streams," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, 2007, pp. 1036-1045.

[130] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science,* vol. 38, pp. 293-306, 1985.

[131] K. Hillol, H. Weiyun, S. Krishnamoorthy, and J. Erik, "Distributed clustering using collective principal component analysis," *Knowl. Inf. Syst.,* vol. 3, pp. 422-448, 2001.

[132] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta, "Clustering distributed data streams in peer-to-peer environments," *Information Sciences,* vol. 176, pp. 1952-1985, 2006.

[133] G. Mohamed Medhat and S. Y. Philip, "A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering," in *Proceedings of the 2006 ACM symposium on Applied computing* Dijon, France: ACM, 2006.

[134] Z. Qi, L. Jinze, and W. Wei, "Approximate Clustering on Distributed Data Streams," in *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, 2008, pp. 1131-1139.

[135] M. Halkidi and I. Koutsopoulos, "Online Clustering of Distributed Streaming Data Using Belief Propagation Techniques," in *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, pp. 216-225.

[136] D. Henriksson, A. Cervin, and K.-E. Arzen, "TrueTime: Real-time Control System Simulation with MATLAB/Simulink," in *Proceedings of the Nordic MATLAB Conference*, Copenhagen, Denmark, October 2003.

[137] A. Cervin, D. Henriksson, and M. Ohlin, "TrueTime 2.0 beta 5 - Reference Manual," Department of Automatic Control, Lund University Jun 2010.

[138] M. De Biasi, C. Snickars, K. Landernas, and A. J. Isaksson, "Simulation of process control with WirelessHART networks subject to packet losses," in *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, 2008, pp. 548-553.

[139] A. Cervin, M. Ohlin, and D. Henriksson, "Simulation of Networked Control Systems Using TrueTime " in *3rd International Workshop on Networked Control Systems: Tolerant to Faults* Nancy, France Jun 2007.

[140] M. Andersson, D. Henriksson, A. Cervin, and K. Arzen, "Simulation of Wireless Networked Control Systems," in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, 2005, pp. 476-481.

[141] C. G. Cassandras, C. Seatzu, and D. Henriksson, "Truetime: Simulation of networked computer control systems," in *2nd IFAC Conference on Analysis and Design of Hybrid Systems* Hotel Calabona, Alghero, Italy, 2006, pp. 272-273.

[142] Texas Instruments "MSP430F22x4 Mixed signal Microcontroller (Rev. F) Datasheet" www.ti.com.product/msp430f2274, 14 July, 2011.

[143] Texas Instruments "CC2530 (Rev. B) Datasheet" www.ti.com.product/CC2530, 05 October, 2010.

[144] L. L. d. A. Andr, M. S. F. Carlos, F. N. Eduardo, S. B. Luciana, A. F. L. Antonio, F. Antnio Otvio, and J. N. J. C. Claudionor, "Data Stream Based Algorithms For Wireless Sensor Network Applications," in *Proceedings of the 21st International Conference on Advanced Networking and Applications*: IEEE Computer Society, 2007.

[145] S. Ullah, J. J. Ahmad, J. Khalid, and S. A. Khayam, "Energy and distortion analysis of video compression schemes for Wireless Video Sensor Networks," in *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pp. 822-827.

[146] M. Ilyas and I. Mahgoub, *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*: Taylor & Francis, 2004.

[147] C. Graham, D. Mayur, I. Piotr, and S. Muthukrishnan, "Comparing Data Streams Using Hamming Norms (How to Zero In)," *IEEE Trans. on Knowl. and Data Eng.,* vol. 15, pp. 529-540, 2003.

[148] J. C. Dunn, "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters," *Journal of Cybernetics,* vol. 3, pp. 32-57, 2011/11/23 1973.

[149] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*: Kluwer Academic Publishers, 1981.

[150] H. Sabit, A. Al-Anbuky, and H. Gholamhosseini, "Data stream mining for wireless sensor networks environment: energy efficient fuzzy clustering algorithm," *International Journal of Autonomous and Adaptive Communications Systems,* vol. 4, pp. 383-397, 2011.

[151] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*: Morgan Kaufmann Publishers Inc., 2005.

[152] H. Sabit, A. Al-Anbuky, and H. Gholam-Hosseini, "Distributed WSN Data Stream Mining Based on Fuzzy Clustering," in *Ubiquitous, Autonomic and Trusted Computing, 2009. UIC-ATC '09. Symposia and Workshops on*, 2009, pp. 395-400.

[153] A. Rakesh, G. Johannes, G. Dimitrios, and R. Prabhakar, "Automatic subspace clustering of high dimensional data for data mining applications," *SIGMOD Rec.,* vol. 27, pp. 94-105, 1998.

[154] A. Rakesh, G. Johannes, G. Dimitrios, and R. Prabhakar, "Automatic subspace clustering of high dimensional data for data mining applications," presented at the Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Seattle, Washington, United States, 1998.

[155] S. Chiu, "Fuzzy Model Identification based on cluster estimation," *Journal of Intelligent Fuzzy Systems,* vol. 2, pp. 267-278, 1994.

[156] R. R. Yager and D. P. Filev, "Generation of Fuzzy Rules by Mountain Clustering," *Journal of Intelligent and Fuzzy Systems,* vol. 2, pp. 209-219, 1994.

[157] J. C. Bezdek, "A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. PAMI-2, pp. 1-8, 1980.

[158] J. Chen, Z. Qin, and J. Jia, "A Weighted Mean Subtractive Clustering Algorithm," *Information Technology Journal,* vol. 7, pp. 356-360, 2008.

[159] P. Yang, Q. Zhu, and X. Zhong, "Subtractive Clustering Based RBF Neural Network Model for Outlier Detection," *Journal of Computers,* vol. 4, pp. 755-762, 2009.

[160] J. Tian, L. Zhu, S. Zhang, and L. Liu, "Improvement and Parallelism of k-Means Clustering Algorithm," *Tsinghua Science and Technology,* vol. 10, pp. 277-281, 2005.

[161] M. N. Halgamuge, S. M. Guru, and A. Jennings, "Energy efficient cluster formation in wireless sensor networks," in *Telecommunications, 2003. ICT 2003. 10th International Conference on*, 2003, pp. 1571-1576 vol.2.

[162] Commonwealth of Australia 2011, Bureau of Meteorology "Climate Data Online Service." ABN 92 637 533 532, 2011. [Online]. Available: http://www.bom.gov.au/climate/data-services/. [Accessed Feb. 18, 2011

[163] A. G. McArthur, "Fire Behaviour in Eucalypt Forests.," CommonW. Aust. For. And Timber Bur. Leaflet Number 107. 36pp, 1967.

[164] G. Resta and P. Santi, "A Framework for Routing Performance Analysis in Delay Tolerant Networks with Application to Noncooperative Networks," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 23, pp. 2-10, 2012.

[165] P. Kumar, Gu, x, nes, M., Q. Mushtaq, and J. Schiller, "Optimizing Duty-Cycle for Delay and Energy Bound WSN Applications," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, 2010, pp. 692-697.

[166] C. E. v. Wagner, *Development and structure of the Canadian forest fire weather index system / C.E. van Wagner*. Ottawa :: Canadian Forestry Service, 1987.

[167] M. Hefeeda and M. Bagheri, "Forest Fire Modeling and Early Detection using Wireless Sensor Networks," *Ad Hoc and Wireless Sensor Networks,* vol. 7, pp. 169-224, 2009.

[168] T. Kevin, J. San-Miguel-Ayanz, S. Richard, D. Murray, A. Martin, J. D. Carlson, and M. Gary, "Current Methods to Assess Fire Danger Potential," in *Wildland Fire Danger Estimation And Mapping*. vol. Volume 4, ed: WORLD SCIENTIFIC, 2003, pp. 21-61.

[169] B. D. Amiro, K. A. Logan, B. M. Wotton, M. D. Flannigan, J. B. Todd, B. J. Stocks, and D. L. Martell, "Fire weather index system components for large fires in the Canadian boreal forest," *International Journal of Wildland Fire,* vol. 13, pp. 391-400, 2004.

[170] C. E. VanWagner and T. L. Pickett, *Equations and FORTRAN program for the Canadian forest fire weather index system*. Ottawa: Minister of Supply and Services Canada, 1985.

[171] B. Leblon, M. Alexander, J. Chen, and S. White, "Monitoring fire danger of northern boreal forests with NOAA-AVHRR NDVI images," *International Journal of Remote Sensing,* vol. 22, pp. 2839-2846, 2001.

[172] Wikipedia article on sensor node, http://en.wikipedia.org/wiki/Sensor_node, [cited 2011-08-22].

[173] H. Qi, P. T. Kuruganti, and Y. Xu, "The development of localized algorithms in wireless sensor networks," *Sensors,* vol. 2, pp. 286-293, 2002.

[174] EPA(1987). On-Site Meteorological Program Guidance for Regulatory Modeling Applications, EPA-450/4-87-013. Office of Air Quality Planning and Standards, Research Triangle Parks, North Carolina 27711.

[175] O. World Meteorological, *Guide to meteorological instruments and methods of observation*. Geneva, Switzerland: Secretariat of the World Meteorological Organization, 1996.

[176] The state Climatologist (1985). Publication of the American Association of State Climatologists: Height and Exposure Standards for Sensor on Automated Weather Stations, v. 9, No, 4, October, 1985.

[177] L. Golab and M. T. Özsu, "Issues in data stream management," *SIGMOD Rec.,* vol. 32, pp. 5-14, 2003.