Prolonging Network Life Time and Running Multi-Sequence Applications in Wireless Sensor Networks

By

Amjed Abdul Majeed

B.Sc. Electronics and Communications Engineering, Mosul University; (1992)

Submitted to the School of Computing and Mathematics

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at

Charles Sturt University

April 2014
To my parents, Nadia Baker and Dr. Abdul Hameed Abdul Majeed, to my wife Sally Ibrahim who has always been supportive and patient during my years of study, and finally to my children who were all along understanding and forgiving for me being away working on my study.
Abstract

Traditionally, wireless sensor networks’ deployment is meant for a single application, which limits the commercial deployment of a sensor network. In many scenarios, however, it has become very desirable to use a sensor network to run multiple applications where the demand for applications that can harness the capabilities of a sensor-rich environment increases. With advancement in sensor technology, the size and cost of a sensor node have been reduced significantly, allowing a large number of these resource-constrained nodes be deployed to form a dense network. This thesis proposes a network design model that is based on utilizing a dense wireless sensor network to achieve running multiple applications in a predefined sequence for scenarios requiring multiple applications. We also claim that the design model proposed when applied for a single running application would result in an energy consumption improvement that would result in prolonging network life time.

The proposed network design model fundamentally works on dividing the region of interest into multiple sets called dynamic switching sets (DSS), each of which has a number of sensor node subscribers allocated by the base station (BS). These DSS are then re-organized by the base station into a number of sets equivalent to the number of applications intended to run in a pre-defined sequence over the entire network life. The research model was implemented and tested to run three different applications on the network such that there were three different sets of DSS. Each DSS set would run one application in a predefined sequence while the remaining sets transit to energy saving sleep mode. Hence only one set would be active running one application while maintaining coverage and connectivity until new instructions are disseminated by the base station for the next in line application sequence (second application) to go active and running. Similarly, when the third set comes on active running the third application, the first two sets would transit to sleep mode after completing their assigned roles in running the first two applications. The experimental results show that the proposed design model also serves to improve energy consumption significantly and prolong network life time when implemented for single applications running over the entire network life time.

The proposed research focuses on two dominating factors pertaining to the cost of a wireless sensor network (WSN). The first is the return of investment, that is the ‘usefulness’ of a WSN when deployed and hence the importance for the network to perform multi-application tasks over its intended lifetime. The second is the improvement of energy consumption when the model is
applied for a single running application of the unattended network resulting in prolonging network life time. Both factors constitute a WSN that features modular and flexible infrastructure.

Thesis Supervisor: Dr. Tanveer A Zia
Associate Head, School of Computing and Mathematics, Charles Sturt University
Acknowledgments

I am deeply indebted to my advisor, Dr. Tanveer A Zia, for making more than five years of my graduate study one of the best periods of my life. He has been an advisor, a mentor and a good friend who continued support and guidance have made me excel in my study. He was always available and around when I needed him, his continuous motivation and encouragement for publications has led me to publish a number of papers from my study and attend and present them at reputable conferences.

I would also like to thank all the people whom I met at CSU when I first came for a visit to the School of Computing and Mathematics and during my consecutive visits who made adjusting to a new place easy and created a friendly working environment.

I thank my beloved wife and children for being patient and providing continued support and encouragement during the years of my study.

My father and mother whose love and support have been the true inspiration in my study journey and whom every time I made that long distance call, they would tell me how proud they were to see me pursuing my graduate study.

My friends and family, named and unnamed, you know who you are and please you know how much I appreciate you all.
Publications from this Thesis

This idea has been accepted and published in the following well-reputed international conferences:


# Table of Contents

ABSTRACT .................................................................................................................. III

ACKNOWLEDGMENTS ................................................................................................. V

PUBLICATIONS FROM THIS THESIS .......................................................................... VI

TABLE OF CONTENTS .................................................................................................. VII

LIST OF FIGURES ......................................................................................................... X

LIST OF TABLES ........................................................................................................... XII

GLOSSARY ....................................................................................................................... XIII

CERTIFICATE OF AUTHORSHIP .................................................................................... XVI

1 INTRODUCTION ........................................................................................................... 1
  1.1 SENSOR NETWORKS ............................................................................................... 1
  1.1.1 Sensor Node Architecture .................................................................................. 2
  1.1.2 Network Architecture ....................................................................................... 5
  1.1.3 Wireless Sensor Networks – Application Examples ............................................ 7
  1.2 MOTIVATING SCENARIOS ................................................................................... 11
  1.3 SECURITY DOMAIN ............................................................................................ 12
  1.4 AIMS OF STUDY AND RESEARCH QUESTION ................................................... 12
  1.5 CONTRIBUTION OF THIS THESIS .................................................................. 15
  1.6 THESIS ORGANIZATION .................................................................................... 15
  1.7 SUMMARY ............................................................................................................ 16

2 OVERVIEW OF WSN AND RELATED WORK ........................................................... 17
  2.1 CONCURRENCY IN RUNNING MULTIPLE APPLICATIONS IN WSN .................... 18
  2.2 RUNNING AND SUPPORTING MULTIPLE APPLICATIONS IN WSN ................... 27
  2.3 DETERMINISTIC AND STOCHASTIC COVERAGE ............................................. 31
  2.4 PROLONGING NETWORK LIFETIME IN WSN ...................................................... 33
     2.4.1 Node-level Power Management ..................................................................... 35
     2.4.2 Efficient Data Delivery .................................................................................. 47
     2.4.3 Network Level – Topology Control Protocols ............................................... 50
     2.4.4 Energy Dissipation Model ............................................................................ 53
     2.4.5 Sensing Model of a Sensor Node ................................................................. 54
  2.5 WIRELESS PROPAGATION MODELS .................................................................. 56
     2.5.1 The Free Space Propagation Model .............................................................. 56
     2.5.2 The Two-Ray Ground Model ....................................................................... 57
     2.5.3 The Log-Distance Path Model ..................................................................... 58
  2.6 HARDWARE AND SOFTWARE PLATFORMS FOR WSNs .................................... 59
3 NETWORK TOPOLOGY STRUCTURE WITH DYNAMIC SWITCHING SETS RUNNING MULTI-APPLICATIONS IN WSN

3.1 GENERAL NETWORK TOPOLOGY

3.2 CONNECTIVITY IN NETWORK TOPOLOGY

3.3 DYNAMIC SWITCHING SETS BASED TOPOLOGY STRUCTURE

3.4 CLUSTERING IN WIRELESS SENSOR NETWORKS

3.5 SUMMARY

4 RUNNING MULTI-SEQUENCE APPLICATIONS IN WSN – MODELING AND ALGORITHM

4.1 MODELING OF WIRELESS SENSOR NETWORKS

4.2 MODELING OF MULTI-SEQUENCE APPLICATIONS RUNNING ON WSN

4.3 SYSTEM ALGORITHM AND FLOWCHART

4.4 SUMMARY

5 SIMULATION - EXPERIMENTAL RESULTS AND ANALYSIS

5.1 RESEARCH WORK SIMULATOR - OMNET++ AND CASTALIA

5.2 TECHNICAL PRELIMINARIES

5.3 SIMULATION RESULTS

5.3.1 Simulation Set Up

5.3.2 First Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s Simulation Time | Node Initial Energy 100J | PLD0 55dB

5.3.3 Second Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s Simulation Time | Node Initial Energy 18720J | PLD0 55dB
5.3.4 Third Simulated Scenario – Three sets of DSS groups running multiple applications.
Nodes | 150sec SimulationTime | NodeInitialEnergy 18720J | PLDO 62dB

5.4 SUMMARY

6 CONCLUSIONS

6.1 SUMMARY OF CONTRIBUTIONS

6.2 FUTURE WORK

BIBLIOGRAPHY
List of Figures

**Figure 1.1:** Components of a typical wireless sensor node (adapted from [56]) .................... 2

**Figure 1.2:** A MicaZ sensor node (mote) with labeled devices. From Mimsic Powerful Sensing Solutions. http://www.memsic.com/wireless-sensor-networks/), February, 2014 .................... 5

**Figure 1.3:** Architecture of general wireless sensor network (source from [89]) ..................... 6

**Figure 2.1:** Middleware layer Architecture of the C-agent and S-agent ...................................... 22

**Figure 2.2:** Wireless Sensor Network Model (adapted and modified from [56]) ....................... 34

**Figure 2.3:** General classification of energy saving in WSN (adapted and modified from [56]). 34

**Figure 2.4:** String Topology Network (adapted from [56]) .................................................. 38

**Figure 2.5:** Pipelined wakeup procedure in pipelined tone wakeup (PTW), (adapted from [56]).38

**Figure 2.6:** Radio triggered power management (adapted from [56]) .................................... 39

**Figure 2.7:** The listening technique of preamble sampling (adapted from [74]) ....................... 43

**Figure 2.8:** Asynchronous sleep using TICER/RICER (adapted from [74]) ............................. 44

**Figure 2.9:** B-MAC Components with memory requirements (adapted from [74]) .................... 45

**Figure 2.10:** S-MAC, Sleep/wake duty cycles (adapted from [74]) ....................................... 45

**Figure 2.11:** D-MAC, receive-transmit- sleep cycle (adapted from [74]) ............................. 47

**Figure 2.12:** Virtual grids in GAF (adapted from [56]) .................................................. 51

**Figure 2.13:** First order energy dissipation model (adapted from [89]) ................................. 53

**Figure 2.14:** The probabilistic sensing model (adapted from [89]) .................................... 55

**Figure 2.15:** The two-ray propagation model (adapted from [89]) .................................... 57

**Figure 2.16:** (a) MicaZ sensor mote (b) Mica2 senor board (c) Mica2Dot sensor (From Mimsic: Powerful Sensing Solutions. http://www.memsic.com/wireless-sensor-networks/ ). February, 2014 .................................................... 60

**Figure 2.17:** MTS300 data acquisition board (Source: [7]) ................................................ 61

**Figure 2.18:** MTS 310 data acquisition board (Source [7]) .................................................. 62

**Figure 2.19:** A TelosB Sensor. (From Mimsic: Powerful Sensing Solutions. http://www.memsic.com/wireless-sensor-networks/) February, 2014 .................................................... 62

**Figure 2.20:** Imote2 Sensor (Source: [7]) ........................................................................... 63

**Figure 2.21:** Taxonomy of programming models for WSNs (adapted from [19]) ................... 64

**Figure 3.1:** Large dense wireless sensor network (adapted and modified from [74]) ............. 70

**Figure 3.2:** Star-connected single-hop topology (adapted from [74]) ................................ 71

**Figure 3.3:** (a) Flat multi-hop mesh, (b) Structured grid (adapted from [74]) ....................... 72

**Figure 3.4:** Two-tier hierarchical cluster topology (adapted from [74]) ............................ 73

**Figure 3.5:** G (n,R) geometric random graph : (a) Small R value results in sparse graph and (b) Large R value results in dense network connectivity (adapted from [74]) ....................... 74

**Figure 3.6:** Probability of connectivity for a geometric random graph with respect to transmission radius (adapted from [74]) ............................................................... 75

**Figure 3.7:** Probability of connectivity for a geometric random graph with respect to the number of nodes in a unit area (adapted from [74]) ...................................................... 76

**Figure 3.8:** Phase 1 – Sensor node deployment and initialization ........................................... 78

**Figure 3.9:** Phase 2 – Formation of network dynamic switching sets (DSS) ....................... 79

**Figure 3.10:** Running first application on a DSS set ................................................................ 80

**Figure 3.11:** Running second application on a DSS set ...................................................... 81
Figure 3.12: Inner clustering structuring with cluster heads (CH) communicating with sink of the first DSS set.................................................................................................................................................. 82
Figure 3.13: Inner clustering structuring with cluster heads (CH) communicating with sink of the second DSS set ................................................................................................................................................. 83
Figure 3.14: Graph representation of a WSN (adapted from [96]) .................................................................................................................. 85
Figure 4.1: A graph representing a distributed system ................................................................................................................................. 92
Figure 4.2: A directed graph .................................................................................................................................................................................. 93
Figure 4.3: Sensing Field for wireless sensor nodes deployment ................................................................................................................... 99
Figure 4.4: Dynamic Switching sets for three WSN applications ................................................................................................................ 100
Figure 4.5: An example of clustering procedure/finding area of each dynamic switching set, P0 is the centroid location of sensing field, P1 and P2 are the locations of two boundary sensor nodes, PV belongs to P1P2, and ACL = DSS is the cluster area (adapted from [109]) ................................................................................................................................. 106
Figure 4.6: The angle \( \theta \) shown for a Dynamic Switching Set partition .................................................................................................................................................. 107
Figure 4.7: Node allocation procedure to each DSS ........................................................................................................................................... 108
Figure 4.8: Cluster Head location calculation in each partition/cluster (adapted and modified from [109]) ........................................................................................................................................................................... 110
Figure 4.9: Running multi-sequence applications in WSN. Application-1 is active and running, application-2, and-3 are in sleeping mode ................................................................................................................................................................. 111
Figure 4.10: Running multi-sequence applications in WSN. Application-2 is active and running, applications-1, and -3 are in sleeping mode ................................................................................................................................................................. 112
Figure 4.11: Running multi-sequence applications in WSN. Application-3 is active and running, applications-1, and -2 are in sleeping mode ................................................................................................................................................................. 113
Figure 4.12: Flowchart for the system algorithm simulation ........................................................................................................................................ 116
Figure 5.1: OMNeT++ Model, simple and compound modules (adapted from [112]) .................................................................................................................................................................................. 119
Figure 5.2: Node internal structure in Castalia Simulator (adapted from [113]) ............................................................................................ 121
Figure 5.3: Active Nodes in first DSS running application one .......................................................................................................................... 139
Figure 5.4: Active Nodes in second DSS running application two ...................................................................................................................... 140
Figure 5.5: Active Nodes in third DSS running application three ...................................................................................................................... 141
Figure 5.6: Total network packets and individual group packets exchanged during the execution of three applications on WSN ................................................................................................................................................ 142
Figure 5.7: a) Remaining energy in joules of a node, b) Average energy consumption in Joules of a node with and without DSS mechanism ........................................................................................................................................ 152
List of Tables

Table 1.1: Research Work Stake Holders ........................................................................................................... 14
Table 2.1: Comparison of related work on concurrency at different node levels (adapted and modified from [19]) ................................................................................................................................. 25
Table 2.2: Mica Family of Sensors (adapted from [7]) ......................................................................................... 61
Table 2.3: Telos Platform and specification (adapted and modified from [7]) .................................................... 63
Table 2.4: Imote2 Specification (adapted from [7]) ......................................................................................... 64
Table 5.1: Castalia output files for all simulations, with and without DSS implementation....................... 130
Table 5.2: List of outputs produced by modules comprising the node during simulation.................... 131
Table 5.3: Average number of packets received by the base station from each DSS group and from each sensor node (packet rate of 1 packet/sec) ............................................................................. 132
Table 5.4: First Simulation, trace sample of time stamp of packets sent to the base station ...... 133
Table 5.5: Running three applications on wireless sensor network with Dynamic Switching Sets packets exchange ................................................................................................................................. 136
Table 5.6: Energy consumption in Joules for node transmission with packet rate 1 pkt./sec. 145
Table 5.7: Energy consumption in Joules for node transmission with packet rate 5 packets/sec ............... 146
Table 5.8: Energy consumption in Joules for node transmission with packet rate 10 packets/sec .......... 146
Table 5.9: Energy results for running a single application with DSS implementation ......................... 149
Table 5.10: Energy results of average node consumption when running a single application without DSS implementation ............................................................................................................................................. 150
Table 5.11: Energy performance comparison with 20 seeds (repetitions) simulation with and without DSS implementation for running single application on a WSN .................................................. 153
Table 5.12: Average packets received by the base station per node with wireless channel parameter variation, sigma = 4.0, collision model = 2.0, path loss = 55dB ................................................................. 156
Table 5.13: Average energy consumed by a node and remaining energy in Joules with wireless channel parameter variations, sigma = 4.0, collision model = 2.0, path loss = 55dB ................................................................. 156
Table 5.14: Second Simulation, trace sample of time stamp of packets sent to the base station under wireless channel parameter variations, sigma = 4, collision model = 2.0, path loss = 55dB ................................................................................................................................. 157
Table 5.15: Average packets received by the base station from each DSS set with variation of wireless channel parameters and setting disk range of the node to 15m................................................................. 160
Table 5.16: Average node energy consumed and remaining energy (in Joules) after 150s simulation time with path loss value 62dB ............................................................................................................ 160
Table 5.17: Third Simulation, trace sample of time stamp of packets sent to the base station under wireless channel parameter variations, sigma = 4, collision model = 2.0, path loss = 62dB ................................................................................................................................. 161

xii
Glossary

**ACK**  Acknowledgement, a signal that is used as part of a hand-shaking protocol between sender and receiver.

**BCL**  Boundary of a cluster, which is the boundary of each partitions of the dynamic switching set.

**BFS**  Breadth First Search, is a search algorithm used in graph theory. Normally it begins at the root node and explores all the neighboring nodes. Then of each explored node, it explores their unexplored neighbor nodes and so on until it finds its final goal.

**CCA**  Clear Channel Assessment is one of many carrier sense mechanisms used by the radio module in the node when determining whether the wireless channel medium is clear.

**CDMA**  Code Division Multiple Access, one of the standard technologies used in communication and phone systems. CDMA uses a ‘Wide-Spectrum’ technique whereby electromagnetic energy is spread to allow for a signal with a wider bandwidth. This allows multiple people on multiple phones to be ‘multiplexed’ over the same channel to share a bandwidth of frequencies.

**Channel**  There are many different meanings for channel in communications. Primarily used here to indicate the path over which information can travel (communication takes place).

**CPU**  Central Processing Unit, referred to as the brain of the computer where all the calculations, instructions and execution of the program code take place.

**CSMA**  Carrier Sense Multiple Access, one of the protocols in carrier transmissions for 802.11 networks. In this protocol when a node receives a packet, it checks the channel and if it is clear (no other node is transmitting at the same time), the packet is sent. If the channel is not clear, then the node waits for a random period of time called backoff time, then checks the channel again.

**DAG**  Directed Acyclic Graph in computer science and mathematics is a directed graph with no directed cycles. It is formed by a collection of vertices with directed edges, each edge connecting one vertex to another. However, if you start at a certain vertex \( v \), then there is no way that you follow a sequence of edges that eventually loops you back to \( v \) again.

**DFS**  Depth First Search. This is one of the algorithms that is used for traversing or searching a tree structure or a graph. In a graph for instance, it starts by selecting a node as root and explores as far as possible along each branch before backtracking, page.

**DSS**  Dynamic Switching Sets in our proposed work defined as virtual partitions formed by virtual radial lines through an algorithm run at the base station. Each DSS is a partition of actual size that, together with a number of DSS forms one set that eventually runs one application in the network. These partitions are centrally coordinated by the base station.
to dynamically switch them for multiple application execution on the network. The summation of all DSS over the entire field gives the total area of the field of interest.

**EEPROM**  Electrically Erasable Programmable Read Only Memory, one of many non-volatile memory types used to store intermediate data exchanged between nodes. These are also programmed at the byte level.

**FSK**  Frequency Shift Keying: a method used to encode binary data using frequency modulation (FM).

**GPS**  Global Positioning System: a satellite-based navigation system made up of a network of 24 satellites placed in orbit. GPS works in any weather conditions, anywhere in the world 24 hours a day.

**GUI**  Graphical User Interface: a program interface that utilizes the computer’s graphical capabilities to make the program easier to use and user friendly.

**IoT**  Internet of Things: a system in which unique identifiers are provided to every available thing in life from objects, animals or humans and the ability to transfer data on these things without human to-human or human to-machine interaction. The evolution of IoT came from the convergence of the internet, MEMS and wireless technologies.

**Joule**  SI unit of electrical energy equal to the work done when a current of ampere passes through a resistance of one ohm for one second.

**MAC**  Medium Access Control. It is also referred to as a protocol for data communication and is part of the data link layer. It is primarily responsible for providing control mechanisms for channel accessing that make it possible for a number of nodes to share the wireless channel medium for sending and receiving packets in the network. It is also responsible for controlling the radio module in a sensor node.

**MCU**  Micro Controller Unit, much like a CPU, but usually with additional peripheral devices and hardware such as input/output, serial communication and built-in memory.

**MEMS**  Micro-Electro-Mechanical Systems, the integration of mechanical elements, sensors, actuators, and electronics on a common silicon layer through precise technology fabrication.

**NED**  Network Description Language. This is used in the OMNeT++ platform models to describe the structure of the underlying components, sub-modules and modules and provide the proper interfacing between them.

**Omnidirectional Antenna**  An antenna that radiates uniformly in all directions in at least one plane.

**OMNeT**  Object-Oriented Modular Network Test bed. It is written in C++ and as discrete event network simulation framework, it is designed to simulate distributed systems and can be extended for modeling other systems [112].
**PSK**  Phase Shift Keying: a type of digital modulation scheme that conveys data by changing the phase of the carrier wave.

**RAM**  Random Access Memory: a volatile memory that temporarily stores data and would lose them if power is shut down.

**RF**  Radio Frequency: a carrier or alternating current that can radiate or propagate in an electromagnetic field. In the earth’s atmosphere, radio frequencies are between 3 KHz TO 300 GHz.

**RFID**  Radio Frequency Identification. This uses radio frequency electromagnetic fields to transfer data for automatically identifying tags attached to objects for tracking purpose.

**RSSI**  Received Signal Strength Indicator: a measurement of the power level of the signal received and seen by the receiver side at a certain time and location.

**SINR**  Signal Interference plus Noise Ratio: normally considered as a measure of the clarity of the signal and can be found by dividing wanted power by the unwanted power consisting of all noise and interference.

**ROM**  Read Only Memory: a permanent type of memory that is used to store the program code.

**SQL**  Structured Query Language: used to communicate with a database and uses statements for updating and retrieving data from the database.

**RTS/CTS**  Request to Send/Clear to Send: a two way communication protocol for sending data between nodes. In this protocol when one node needs to send, it first initiates an RTS command, then the corresponding node when not busy will acknowledge by CTS.

**TCL**  Tool Command Language: a scripting language developed for scientific and engineering applications.

**TDMA**  Time Division Multiple Access: a method that provides a channel access for shared medium networks. TDMA divides the signal into different time slots that enables nodes to share the same transmission medium (the wireless channel).
Certificate of Authorship

I hereby declare that this submission is my own work and to the best of my knowledge and belief, understand that it contains no material previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any other degree or diploma at Charles Sturt University or any other educational institution, except where due acknowledgement is made in the thesis [or dissertation, as appropriate]. Any contribution made to the research by colleagues with whom I have worked at Charles Sturt University or elsewhere during my candidature is fully acknowledged. I agree that this thesis be accessible for the purpose of study and research in accordance with normal conditions established by the Executive Director, Library Services, Charles Sturt University or nominee, for the care, loan and reproduction of thesis, subject to confidentiality provisions as approved by the University.

Name: Amjed Abdul Hameed Abdul Majeed

Signature: [Signature]

Date: 8 October 2014
Chapter 1

1 Introduction

This chapter provides an introduction to wireless sensor networks (WSNs). It begins with the internal architecture of a typical sensor node as the main building block of the network followed by a brief description of the network architecture and the most important challenges faced by these WSNs.

Thereafter, we present a number of sensor network applications, examples of deployments and operations from which we draw our motivating scenarios. We then identify a number of open problems that the sensor network research community is working on, two of which are addressed in our research question. We conclude with a thesis contribution and organization.

1.1 Sensor Networks

Advances in wireless networking, micro-fabrication and integration (e.g. sensors and actuators manufactured using micro-electromechanical systems technology, or MEMS), and embedded microprocessors have enabled a new generation of massive scale sensor networks suitable for a range of commercial and military applications. The technology promises to revolutionize the way we live, work and interact with the physical environment [1].

A wireless sensor network is composed of a large number of sensor nodes that are densely deployed either inside the field of interest where a phenomenon to be monitored or very close to it [2]. When deployed, they collectively monitor and disseminate information about a variety of phenomena of interest. Each sensor node is a battery-operated device capable of sensing physical quantities and made up of several parts: a radio transceiver module for wireless communication, memory integrated chip for data and program storage, microcontroller for computation and signal processing, sensing embedded board (temperature, humidity, light, vibration, sound etc.) with electronic interfacing circuit and a battery or an embedded form of energy harvesting. These nodes have emerged as a result of recent advances in low-power digital and analog circuitry, low-power RF design and sensor technology [3]. Sensor networks are distinct from traditional computing domains; their design assumes being embedded in common environments, instead of dedicated ones. A vast number (hundreds or even thousands) of these sensors constitute a wireless sensor
network and when these devices are deployed in large numbers, they need the ability to assist each other to communicate data back to a centralized collection point for further data processing and analysis.

Advances in integrated circuit design are continually shrinking the size, weight and cost of sensor devices, while simultaneously improving their resolution and accuracy. They are now coming equipped with many add-on boards, such as a sensor board that contains application-specific sensors. The availability of a wide variety of sensor boards and sensors such as temperature, air quality, pressure, magnetometers, light, acoustic, accelerometers etc. make WSN popular to serve as a general platform for many application domains, solving many practical problems. A WSN has one or more sinks (base-stations) that collect data from all devices. These sinks are the interface through which the WSN interacts with the outside world.

### 1.1.1 Sensor Node Architecture

The architecture of a sensor node’s hardware comprises five main components [4]: microcontroller, memory, sensors and actuators, communication device and power supply, as shown in Figure 1.1 below:

![Figure 1.1: Components of a typical wireless sensor node (adapted from [56])](image.png)
**MCU**

A Microcontroller Unit (MCU), or just known as a microcontroller, is used to process all the relevant data and execute the code. It is also referred to as the central processing unit (CPU) of a wireless sensor node. It collects and processes data from sensor nodes from which decisions and actions are made. It also executes programs, ranging from time-critical signal processing and communication protocols to application programs. These microcontrollers are conveniently suited to wireless sensor nodes as they play a major role in managing power consumption by going into sleep mode and only certain parts of the controller are active.

**Memory**

Some form of memory is required to store programs and intermediate data: usually, different types of memory are used for programs and data. The two main memory types used are Random Access Memory (RAM) and Read Only Memory (ROM). RAM is needed to store intermediate sensor readings, packets from other nodes etc. and although it is considered fast, its main disadvantage is that it loses data if the power supply is interrupted. Therefore sometimes flash memory is used to serve as intermediate storage of data in case RAM is insufficient or when the power supply could be shut down for some time. ROM, on the other hand, is used to store program code; another version of ROM is the Electrical Erasable Programmable Read Only Memory (EEPROM), which is typically used nowadays to store the code or even use flash memory. However, flash memory tends to be slower when it comes to data access, therefore we need to take into account the long access time delay required for read/write as well as the high energy consumption.

**Sensors/Actuators**

Sensors and actuators are the actual interface to the physical world: devices that can observe or control physical parameters of the environment. The three main categories where sensors can be classified [5] are:

**Passive and omnidirectional sensors**: sensors that can measure a physical quantity at the point of the sensor node without the need to actively probe the environment and hence are passive. There are many examples of these sensors such as temperature and humidity sensors, light sensors, chemical sensors sensitive for given substances, smoke detectors, vibration and microphones.

**Passive, narrow-beam sensors**: although these sensors are also considered passive, they feature direction of measurement, i.e. they are equipped with a motorized platform. One example is a
camera that can take measurements in a given direction and the ability to rotate to that direction when need be.

**Active Sensors**: radar and sonar are two examples of active sensors where a probing environment is needed to get the intended measurement. With regard to the sensing area, it has been occasionally assumed in literature that each sensor node has a certain coverage area that can be used reliably for the particular quantity that it is observing. The sensing metric is basically the detection probability measured based on the distance between a sensor and an object or the event to be detected.

Actuators in general can be as simple as a node opening or closing a switch or controlling a relay, controlling a motor or setting a value in some way. Actuators are now widely researched to be integrated with sensor nodes as part of their overall structure.

**Communication**

Turning a node into a network requires a device (radio module) for sending and receiving information over a wireless channel. Data exchange between individual nodes is via wireless communication and the type used is Radio-Frequency (RF) based communication. RF best fits WSN applications as it provides relatively long range and high data rates, acceptable errors and reasonable data expenditure and, most importantly, does not require line of sight between receiver and sender. A transceiver module that is used in communication is a device that combines both transmitter and receiver in one package and would normally be a half-duplex operation in that it can either transmit or receive as both transmitting and receiving at the same time (full-duplex) on a wireless medium is impractical in most cases in WSN. The prime function of a transmitter/receiver or transceiver is to convert a bit stream coming from a microcontroller (or a sequence of bytes or frames) and convert them to and from radio waves.

**Power Supply**

Power supply for a wireless sensor node is a crucial element of the device as the node’s lifetime depends solely on how much power remains. Batteries are used to provide the necessary energy. Sometimes, sensor nodes are designed to replenish consumed energy by some form of recharging or ‘scavenging’, which is accomplished by obtaining energy from the environment (e.g. solar cells).

Figure 1.2 below shows a typical sensor node from Crossbow (later acquired by Memsic) which integrates all the aforementioned parts. Power is normally provided by two AA batteries in a compartment at the bottom of the mote.
1.1.2 Network Architecture

Wireless sensor network architectures have evolved over time and continue to evolve as technology progresses with new devices and capabilities being built. When WSN was first implemented, it consisted of a flat topology of several sensor node devices measuring a single variable. Nowadays the demand for more sophisticated applications has made a requirement for considerably more sensor nodes, hundreds or even thousands equipped with more powerful capabilities. These applications required new architectures for the efficient transmission of wireless sensor data, which involved layered and clustered topologies in their design that also implemented multiple sinks (base stations). Also, large networks needed different points of connectivity with the outside world, which resulted in a lot of research invested in the design and implementation of communication protocols as seen in Figure 1.3 below: two flat small sensor networks with one base station connected to a wireless ad hoc network, cellular network and the internet which can also serve as an interconnection hub among them.
The incorporation of several base stations evolved network architectures to include heterogeneous devices with more powerful capabilities especially for the Cluster Heads (CH) that would perform complex functions as well as save additional energy.

Despite the fact that a considerable amount of research and efforts over the past decade or so have been directed towards WSNs, there are still many challenges yet to be worked out which would require even more research efforts to be invested. Many of these challenges are a direct consequence of the constrained resources available in these sensor nodes, and hence specific to WSN, with other common challenges faced by most networking technologies. Some important challenges faced by WSNs nowadays are listed below:

**Network lifetime:** sensor nodes are battery powered, therefore the network lifetime depends on how wisely energy is consumed. The importance of extending network life time or minimizing the number of times batteries change comes in large scale WSNs or in applications where it is difficult or dangerous to reach. A life time in the scale of a year or more is very desirable in this case and can be achieved by operating sensor nodes in very low duty cycles.
**Scalability:** in applications where hundreds or even thousands of nodes are used, such as monitoring oil pipelines new challenges are presented that are not seen in small scale WSN since algorithms and protocols used by the small scale ones do not necessarily work in large-scale ones. One example is the routing protocols and clustering based on the LEACH protocol [6]

**Interconnectivity:** sensor nodes need to be fully connected in a network in order to guarantee the reach of data to its destination for possible storage, analysis and action if needed. Since WSNs are envisioned to be connected with many different networking technologies such as cellular networks, ad hoc networks and the internet, there needs to be a new design paradigm that achieves interconnections for the transfer of data to and from WSNs. Basically, gateway devices are used to implement some of the required interconnections such as base stations.

**Reliability:** node reliability is also a crucial factor that has been addressed in the literature. Sensor nodes are prone to failure and damage especially cheap ones that are airborne or thrown from a vehicle to the field of interest. Even if they are partially broken that would still affect their normal functionality.

**Heterogeneity:** when sensor nodes of different functionalities and capabilities (higher in energy level and functions) are embedded into WSN, new algorithms and communications protocols are required. One good example is in clustering as in normal situations the sensor node with a high energy level and more functionality is used as a cluster head to aggregate sensor data and sends them to the base station.

**Privacy and Security:** security mechanisms are very resource demanding which does not come in line with the limited resources available in a sensor node and hence security algorithms and less computational complexity and low energy requirements are needed.

### 1.1.3 Wireless Sensor Networks – Application Examples

The claim of wireless sensor network proponents is that this technological vision will facilitate many existing application areas and bring into existence new ones. Apart from the need to build cheap, simple to program and network, potentially long-lasting sensor nodes, a crucial and primary ingredient for developing actual applications is the actual sensing and actuating faculties with which a sensor node can be endowed [4]. Sensor networks make it possible to monitor, instruct, or control various domains such as homes, buildings, warzones, cities and forests. Sensor networks can observe the sensing environment at a close range and thus have many advantages such as the
ability to monitor the smallest details, and proximity to places that are difficult to reach by humans, for example difficult terrain or hazardous environments [7]. For many physical parameters, appropriate sensor technology exists that can be integrated in a node of a WSN. The applications of wireless sensor network technology can be classified into four main categories:

1.1.3.1 Environmental Monitoring

One of the first and most common uses of WSN is for environmental applications which are becoming more and more important given that climate change, global warming and diminishing natural resources are crucial and common issues worldwide [8]. Natural disaster systems, energy-monitoring systems and hazard response systems are among just a few instances where WSN can contribute. Environmental monitoring can be further broken down into the sub-groups given below:

**Meteorological Monitoring:** The goal in meteorological monitoring is to study several physical and atmospheric magnitudes. Compared to the traditional weather stations that would normally provide information about rainfall, wind speed and direction, temperature, pressure and relative humidity, WSN when used in weather forecasting can provide the acquisition of a large amount of data that would be too difficult to obtain otherwise. These data are then stored in databases for further analysis to improve predictability and reliability, and to assist in preventing the loss of human lives and economical losses as a result of natural disasters such as hurricanes, floods and droughts [9]. In cases of flooding, a WSN may provide real-time information on rainfall and water levels, and in cases of drought forecasting, sensor nodes are deployed to monitor and collect special and temporal ground surface information etc.

**Geological Monitoring:** This is mainly to study the phenomena related to earth state that would assist in the predictions of earthquake happenings, tsunamis and volcanic eruptions. Sensor nodes can be deployed underground and do not need any wired connections to send real-time data that allows quick and instant analysis.

**Habitat Monitoring:** The long-term data collection ability coupled with high resolution that sensor network solutions have for habitat monitoring can provide great potential for both industrial and scientific communities. Habitat monitoring is regarded as a driver application of wireless sensor networks, eliminating the potential impact of human presence when monitoring plants and animals in field conditions and enables data at a scale and resolution that are difficult to achieve through traditional instrumentation.
Pollution Monitoring: Air pollution, water pollution, nose pollution and radioactive contamination are some of the many types of pollution that need to be monitored and controlled in an attempt to reduce the devastating damaging consequences that they may cause.

Energy Monitoring: Energy resource management has become the highest priority as it plays a significant factor in the performance of economies. Wireless sensor networks play a fundamental role in reducing energy waste. Starting from simple temperature measurement or measurement of the number of people present in a room where light and heat are adjusted accordingly.

1.1.3.2 Health Care

Wireless sensor networks play a major part in many health care applications, such as medical treatment, patient monitoring, people rescue etc. There are many other applications involving elderly people and people with mental and physical disabilities. The health care domain can be divided into the following sub-groups.

Patient Monitoring: This is primarily to observe and monitor the patient’s state of health while in hospital or even at home after being discharged. The measurements of patient vital signs can be useful not only for medical records and treatments, but also for later rehabilitation [8].

Disability Assistance: In this application smart sensors are injected inside the human body to help monitor and manage a patient’s diabetes, blood glucose level etc.

People Rescue: In this application, sensors come as tiny wireless badges that people are outfitted with in order to relay continuous data to nearby paramedics as well as to help guide a rescue team much faster to victims in cases of emergency or disaster.

1.1.3.3 WSN Applications in Other Domains

WSNs have also entered into new important applications pertaining to civil infrastructures. They are used to monitor bridges and buildings for the safety of aging structures. Events such as earthquakes can cause invisible damage to the civil infrastructure that could result in life-threatening conditions in the structure long after the actual event has occurred. The embedding of sensors into the civil structure can prevent and reduce the loss of human lives by warning appropriate people about hazardous structures and impending collapses as well as providing information to emergency response services [8]. Other applications being researched and implemented are in vehicle management, where inexpensive wireless sensor networks can be used in applications such as traffic control, vehicle tracking and detection, monitoring car theft and
parking management. Moving towards smart environments has also attracted considerable research attention; many potential applications where deploying sensor networks in indoor environments such as buildings, homes, offices, and laboratories can make the environment seem ‘alive’. [7]. WSNs are also becoming an important part of industrial process control where the need for real-time access to information about the environment pertaining to industrial plants and equipment is required. Unlike traditional wired networks, sensors can be deployed in the bearings of the motors, oil pumps, whirring engines, packing crates, and many other inaccessible or hazardous environments that are inaccessible with normal wired systems [10]. Sensors may also be used to monitor and track assets such as trucks or other equipment for areas where fixed networking infrastructure is not available.

Many WSN applications share some basic characteristics. In most of them, there is a clear difference between sources of data (the actual nodes that sense the data) and sinks (nodes where the data should be delivered to). Following are common operation types of WSN:

**Event Detection**

Sensor nodes should report to the sink(s) once they have detected the occurrence of a specified event. A few examples in this class of application are intruder detection as part of military surveillance, detection of forest fires, and detecting anomalous behavior or failures in the manufacturing process. Here, the infrequency of occurrence of the events of interest is the common characteristic of all these applications. When a WSN is deployed for one of the aforementioned applications, it is expected to be in a state of inactivity for most of the time and only becomes active and conveys report to the sink when an event is detected. [11]

**Periodic Measurements**

Sensors can be tasked with periodically reporting measured values. The reporting period is application dependent. Some application examples in this class include monitoring temperature, light and humidity in buildings, and monitoring environmental conditions affecting crops etc. Actuator modules could also be combined with the monitoring modules to control, for example, the amount of fertilizer in the soil, or the amount of heating or cooling in the building based on the measurements gathered from the distributed sensors. In these cases, sensors would constantly produce some amount of data and be delivered to the base station (BS) [11].
Function application and edge detection

The way a physical value like temperature changes from one place to another can be regarded as a function of location. A WSN can be used to approximate this unknown function, using a limited number of samples taken at each individual sensor mode.

Tracking

The source of the event can be mobile (e.g. an intruder in surveillance scenarios). The WSN can be used to report updates on the event source’s position to the sink(s), potentially with estimates about speed and direction as well. To do so, typically sensor nodes have to cooperate before updates can be reported to the base station.

1.2 Motivating Scenarios

Sensor nodes are usually deployed to support a single application per network to form a dedicated network. This limits the commercial deployment of sensor networks. Consider a wireless sensor network in applications such as road traffic monitoring, disaster or forest fire detection, health care applications, military and security applications etc.

For example, a sensor network deployed to monitor road traffic detects traffic congestion in peak hours or traffic congestion due to an accident. Once the detection application is complete, these sensors would now need to run a new application diverting vehicles towards alternative roads by activating traffic signs accordingly. Another example is in a forest fire detection application, when fire is detected and the fire fighters arrive, they may want to reprogram the sensor network to initiate rescue and search operations. In these applications, nodes perform multiple functions; however, these nodes only support one application throughout the network. Most wireless sensor networks are limited in that the application must be installed prior to deployment and once deployed it is rarely that the application will be changed. These are just a few examples where there is urgency and need for the WSN to be able to execute multiple applications.
1.3 Security Domain

In this application domain wireless sensor networks are used in a wide variety of challenging problems such as detection of toxic chemicals, rescue, target tracking and localization etc. WSNs have also gained wide attention in military applications as they can also be deployed to detect information about enemy movements, explosions, and other phenomena of interest. The increasing use of WSN in the aforementioned applications makes network security a critical issue and one of the pressing concerns in wireless sensor networks. Providing security in a large scale sensor network in not an easy task as there are severe challenges exist since sensor nodes have limited processing capability, wireless links have limited bandwidth, and scarce amount of energy [11]. Despite the above challenges, security is important and even critical to many applications as previously mentioned. Several recent works have addressed security issues in sensor networks [118-121]. However, the privacy and security issues posed by sensor networks still present a rich filed of research problems. In wireless sensor networks, in order to implement and achieve security, it is essential the cryptographic operations are performed, including encryption, and authentication. Communication nodes need to set up these cryptographic operations prior to exchanging data. Most security requirements, such as privacy, authenticity, and integrity, can be addressed by building up on a solid key management framework which is considered a pre-requisite for the security of other primitives and achieving secure infrastructure in WSN [11].

1.4 Aims of Study and Research Question

When wireless sensor networks were first used, they were typically purpose-built, designed to run a single application. Following the recent rapid advances in technology coupled with the increased demand for applications that can harness the capabilities of a sensor-rich environment, there are clear benefits to design a model focused on sharing an infrastructure amongst multiple applications on a single wireless sensor network and give a better return of investment and hence “usefulness” of the network. Considering the aforementioned issues and demands, and after having analyzed the related work related to my research area, I have formulated the following question that sums up the direction of my investigation.
“How can the scale density of small low-cost sensor nodes be used such that a single wireless sensor network is partitioned into smaller dynamic switching sets to run multiple applications in sequential order, as well as when in the case of only a single application needed to run, how can the multiple switching set mechanism achieve enhanced performance in energy saving that would prolong the network life time?”

The literature review and related work carried out in Chapter 2 indicate that a significant amount of research work has been invested and explored in this area. A lot of work was conducted on the possibility of reprogramming sensor nodes after being deployed to the field of interest, others on modifying the internal architecture of sensor nodes in order to be able to share the common infrastructure of the network and manage executing multiple applications. However, the result outcomes of all that has been researched are still limited due to the constrained nature of the sensor nodes and their scarce available resources. In this proposed research we are taking into consideration the constrained resources of these sensor nodes as a given fact on which our design model and algorithms are constructed. It is also worthwhile listing some stake holders who may be affected by the result outcomes of our research outcome as described in Table 1.1 below. I intend to satisfy the expectations of the different groups listed in the table by investigating the answers to my research question.
<table>
<thead>
<tr>
<th><strong>Table 1.1: Research Work Stake Holders</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Running Multiple Sequence Applications on Wireless Sensor Networks and Prolonging Network Life Time</strong></td>
</tr>
<tr>
<td><strong>Literature Review</strong></td>
</tr>
<tr>
<td><strong>Researcher</strong></td>
</tr>
<tr>
<td><strong>Academic Community</strong></td>
</tr>
<tr>
<td><strong>Industry</strong></td>
</tr>
</tbody>
</table>
1.5 Contribution of this Thesis

The contribution of this thesis is primarily a novel idea of running multiple sets of sensor nodes (sub-sets) in a predefined sequence by exploiting the advantages of dense networks as well as prolonging the network life time when the design model is only used to run a single application.

There are a number of significant accomplishments that justify running multiple applications on a single network such as optimization of node redundancy, meeting new demands of sophisticated applications, and better return of investment that is directly related to the “usefulness” of the deployed network. As well the multi-layer concept in our proposed work when applied on a single network to run single application achieves 58% of energy saving compared to traditional network and thus prolonging network lifetime.

The design strategy is primarily focused on dividing the wireless sensor network into multiple dynamic switching sets and allocating sensor nodes to each set (sensor nodes become subscribers and members as part of that particular set) while maintaining the necessary connectivity and coverage. These are formulated in the two listed objectives:

a- One of the most essential researched issues in wireless sensor networks is the ability to run multiple applications on a single network. The prime objective is to achieve full usability of the network with high a return of investment efficiency as well as reducing deployment and administrative costs. The proposed algorithm is designed such that multiple applications run in a predefined sequential mechanism.

b- The second goal is related to energy consumption enhancement and performance, a metric that has become of utmost priority due to the need for a long operational life time of distributed sensing systems particularly when deployed in areas that are difficult to reach. When the need is only for a single application to run over the entire network life time, the proposed strategy of multiple dynamic switching sets is an efficient design that saves a significant amount of energy. This is accomplished by switching sets between active and sleep modes enhancing overall network energy usage to prolong network life time.

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 provides the background information related to the researched work on both running multiple sequenced applications on a single network and on node energy consumption improvement and efficiency for prolonging network life times. Chapter
3 introduces network topology with running multiple applications by the implementation of dynamic switching sets. Chapter 4 describes the conceptual systems design and architecture for the proposed algorithm on Prolonging Network Life Time and Running Multi-Sequence Applications in Wireless Sensor Networks. Chapter 5 describes the simulation of different network scenarios and the experimental results obtained and analysis is provided. Chapter 6 summarizes the work in this thesis, with conclusions and future research directions.

1.7 Summary

A wireless sensor network is a distributed computing system comprising of a number of small sensor nodes, hundreds or even thousands that collaborate together to perform one ultimate goal of data gathering. They also perform in-network processing, have limited energy and hardware, and addressing is based on logical attributes.

A sensor node is characterized by being relatively small, preferably cheap, easy to deploy, energy constrained and using wireless communications. Nodes also include a sensor/actuator board with one or more different sensors, employ a microcontroller with limited computing and storage capabilities. Each of these characteristics introduce problems which need to be solved in order to efficiently use the network.

Wireless sensor networks have a tremendous potential number of applications, from environmental monitoring, health care, industrial process control to security and military applications. In fact, many current deployments provide important data that would have been almost impossible to obtain using traditional techniques.

Wireless sensor networks promise to enable dense, long-lived embedded sensing of the environment. The unprecedented degree of information about the physical world provided by WSNs can be used for in situ sensing and actuation. These sensor networks can also provide a new level of context awareness to other back-end applications, making them an integral part of the vision of pervasive, ubiquitous computing with the long term objective of seamlessly integrating these sensor networks into larger multi-tier systems [12].
Chapter 2

This chapter contains information about the literature review and recent work related to this research. It is mainly divided into six major categories, the first two being on the work conducted on concurrency and running multiple applications on wireless sensor networks. Strengths, weaknesses and trade-offs made in scholarly published research works are discussed and compared to our novel algorithm on running multiple applications in WSN. Then we proceed to the second part of the research problem, energy issue and power management, one of the most important issues for a sensor network. In addition to being restrained in power, batteries (and energy sources) are also quite expensive, increasing the cost of the sensor node substantially. This is another good argument for exploiting the energy source as effectively as possible. Thus, we investigate the most recent work and techniques presented on improving node energy consumption and ultimately prolonging network life time and how our proposed model can serve towards saving overall network energy consumption.

Next, we touch on the two most common deployment types in wireless sensor networks, deterministic and stochastic, and the difference between them. The wireless channel (and the models available) is an important topic that needs to be presented for its impact on packet delivery and hence network communication. The chapter concludes with a review of the most widely available software and hardware platforms for wireless sensor networks.

2 Overview of WSN and Related Work

Our proposed model and implemented algorithms are the result of an intensive literature review of current and past research work in the field of wireless sensor networks. It specifically focuses on research that has been done both on supporting running multiple applications on a single network and prolonging network life time. Our novel algorithm for running multiple sequenced applications takes into account the coverage and connectivity issues of the field of interest for the deployed wireless sensor network. The literature review and related work in this chapter is mainly divided into the following parts:

a. Concurrency in running applications in WSN
b. Running and supporting multiple applications in WSN
c. Deterministic and stochastic coverage in WSN.
d. Prolonging network life time in WSN
e. Wireless Propagation Models in WSN
f. Hardware and Software Platforms for WSN.
2.1 Concurrency in Running Multiple Applications in WSN

Concurrency and multithreading do not get along well [13], mainly because of the costs and possible deadlocks. In sensor networks concurrency raises bigger issue because of the resource limitations. Much work has been carried out on concurrency and running multiple applications in wireless sensor networks. Concurrency in wireless sensor networks can be divided into the following categories:

I. Applications concurrency at Node level
II. Application concurrency at Group level
III. Applications concurrency at Network level.

Concurrency in the sense of executing multiple tasks at node level is an essential feature requirement for a sensor node in order to perform and be part of a distributed network system. The extreme limited resources of sensor nodes make achieving concurrency a challenging task faced by all software developers and system designers etc. A lot of research work has been invested into implementing an event-driven paradigm or multi-threading approach when it comes to node level software development. The fact that the event-driven approach has proven to be efficient in implementing concurrency came at the expense of code writing complexity and the associated development time required. Hence, researchers started looking into the possibility of implementing a procedural programming approach that uses the multi-threading concept for sensor nodes with which all traditional personal computer software developers are familiar. This would not only make sensor code development easier but also invites and attracts a wide number of developers to contribute to wireless sensor nodes application development and specifically to accomplish concurrency. The concurrency concept in this case is about multiple simultaneous tasks that are achieved through threads or processes. However, implementing threading inside of deeply embedded systems such as sensor nodes in a wireless sensor network brings some additional challenges and complexities. A few to mention are: 1) memory resources are limited which imposes a large problem as each thread requires its own task 2) in order to avoid overflow stack, they need a sufficient amount of memory to be allocated and 3) the cost of context switching. Preemptive multi-threading gives each thread a time-slice to run, after it forces the context to switch. This is a positive thing when running a node with a microcontroller of some high frequency range, however with microcontrollers running at only few MHz of frequency this preempting action would negatively impact the node performance.
If we look more into the concurrency concept at node level, it basically involves seemingly concurrent execution of tasks at the processor of the node. For example, a system could check a sensor to decide whether data are available and process the data right away, and at the same time check the transceiver module for data packet availability, and immediately process the packet and so on. TinyOS [14, 15], written in nesC [16] language, allows concurrency at node level to be implemented, however the rigorously event-based style and exclusion of blocking operations are often the source of complexity for TinyOS.

To deal with the above-mentioned complexity issues of the event-driven model, an alternative approach that has been the focus of much research is to use thread abstraction. When it comes to operating systems for WSN, as previously stated there is a long-standing argument over the event-driven model versus the thread model. One major constraint for thread abstraction in sensor network programming is the limited hardware resources. In terms of memory space, it is often too expensive to statically allocate per-thread stacks. However, the event-driven model is more suitable for letting the processor sleep as much as possible, hence achieving energy-efficiency. On the other hand by having blocking execution contexts, thread abstraction simplifies the programs significantly. Y-threads [17] propose a model to support concurrency at node level as an alternative option to the event-driven TinyOS by realizing preemptive multithreading. They capture applications as a combination of computation and control behaviors, and have a shared stack for the non-blocking computations and multiple separate stacks for the control behaviors that are blocking. Also a library-based approach enables multi-threading programming called TinyThread [18] that has been suggested would allow developers to write programs for Wireless Sensor nodes in the more traditional procedural programming style. TinyThreads implements cooperative multithreading, in which an execution context is yielded explicitly by calling yield () or implicitly by waiting on a blocking I/O routine. The latter approach includes a scheduler being active only when a thread is scheduled and going into sleep mode when not active, an approach that we have considered as part of our suggested model.

The idea of combining features of the event-driven approach with that of the threading concept for sensor node operation enables efficient concurrency and makes program development less time consuming. As previously explained, TinyThread implements cooperative multithreading as opposed to preemptive multithreading that supports concurrency analysis of TinyOS. This allows the release of having threads contend for the processor at all times and allows developers to choose the time length of the thread to run. Hence, although TinyThreads are good for normal operation runs, the cooperative multithreading feature falls behind when it comes to computationally expensive operation and long-running calculations. As well, the multithreading concept also comes
at the expense of memory consumption of program and data memory, response time and power consumption.

A great deal of work has also been done with the focus on re-programmability (to run different applications), that is, the capability of injecting new codes into each node on site dynamically [19]. Some of these research works are based on virtual machines and middleware. Mate [20], a tiny virtual machine for sensor networks was the first virtual machine architecture proposed. It is stack-oriented virtual machine implemented on top of TinyOS. The architecture proposed had very limited flexibility and minimal support for concurrency which was then addressed by ASVM [21], Application Specific Virtual Machine.

Melete system [22], describes a model to support concurrency at both node and network levels. The grouping in this case is dynamic on the fly deployment of applications and is based on the contemporary status of the sensor nodes. The Melete system is based on the virtual machine concept of Mate [20], and extends it to support multiple concurrent applications and provides a high-level interface for ease of programming and also allows complex programs to be very short reducing the energy cost of transmitting new programs. Trickle [23] has also been used for code dissemination based on a ‘polite gossip’ technique where nodes periodically broadcast a code summary to local neighbors and stay quiet if they have recently heard a summary identical to theirs. Although Melete the system [22] seems a promising approach when it comes to concurrent applications at network level, it is not clear at node level how much memory would be dedicated for each application as this would limit the number of possible applications that could be running especially with the scarce amount of memory available to these sensor nodes.

Other work has suggested using scoping building blocks [24] as a general concept for creating subsets within the network. Scoping suggests an important point of separation of different tasks both at node and networking level. In our approach, separation of applications at network level would ultimately result in separation at node level, since each set of nodes is allocated to one dynamic switching set and assigned to run one application at a time. For instance, in a wildfire scenario if there are two subsets of nodes running two different applications in a single network, each set of nodes would utilize different available sensing devices equipped on their sensor boards for the intended application, and this would provide separation at node level. One set could be provided with temperature and humidity sensors, and another set might be equipped with motion detectors, vibration or GPS.

Other research work has explored the possibility of building a re-programmability feature in some operating systems. For example, some operating systems have started to feature a dynamic
reprogramming capability, as with TinyOS, where full image replacement takes place that overwrites the existing image of the sensor node. This is enabled using Deluge [25], a dissemination protocol, with TOSBoot Loader for TinyOS. However, this method has proved to be quite inefficient as it requires the entire image to be transferred at the expense of energy usage consumed by the long transmission time of the radio modules that also come with high latency.

The re-programmability feature is sometimes realized in middleware, a layer that acts as a bridge between the operating system and the applications: one of its aims is to support and coordinate multiple applications at run time environment. IMPALA [26] and SensorWare [27] are examples where middleware is used.

IMPALA is a middleware architecture that enables application modularity, adaptivity and reparability in wireless sensor networks. It allows software updates to be received via the node’s wireless transceiver and to be applied to the running system dynamically. SensorWare supports TCL-based control scripts for the language used for reprogramming. Both IMPALA, which was successfully designed to be part of the Zebranet mobile sensor network and SensorWare which supports easy programming with scripting language as well as concurrent multi-tasking of a node so that multiple applications can concurrently execute on WSN, require the luxury of memory availability as they were designed for a richer hardware platform such as iPAQ pocket PC handheld systems.

Our approach [28] aims to cover functionality modification at node level that would lead to the ability to run multiple applications on a single deployed network. The work was based on developing a middleware layer embedded on top of TinyOS that would manage and coordinate two mobile network agents; a configuration agent (C-agent) and a switching set agent (S-agent). Agents are the primary means of communications amongst the nodes in the network by acting as couriers: they tend to carry various pieces of data such as instructions, tasks, updates, network status, etc. These two agents are injected into the network by the base station to be directly picked up and discovered by the cluster heads, which then disseminate to their sensor node cluster members. In the case of C-agent, this is used when minor modifications or updates are needed in the functionality of sensors such as changing the temperature sensor threshold or the sampling frequency rate etc. However, the S-agent is only injected to the network when the current application needs to be halted and switched to a new, different application, provided that all sensor nodes have been programmed prior to deployment with multiple applications that would sequentially be actively running at a particular time, based on the particular demand phase of the network life time. The multiple set structure of the network allows the active sets of nodes, once
they receive the agents, to go to energy-saving sleep mode and at the same time activates the sleeping sets of nodes to run the new requested application. Figure 2.1 below shows a simplified internal structure of the aforementioned middleware.

![Figure 2.1: Middleware layer Architecture of the C-agent and S-agent.](image)

The work suggested in [29] introduces multi-agent support for multiple concurrent applications and dynamic data-gathering in wireless sensor networks. The proposed work presents a middleware solution called Sensomax that would run on a Java platform. It is component-based and multithreading software that would run on java-enabled embedded devices. It provides many promising features such as; multi-tasking, multiple applications support, re-programming at run time for task modifications and supporting various scalability mechanisms. However, with the all promising features of the work presented, there was no clear direction on the network coverage being implemented and how to maintain it.

The paper discusses dividing the region of interest into separate regions such that each would run one application: this arrangement means that coverage is only confined to that specific region for which a particular application is running and not for the entire network. As well, one of the features introduced allows nodes to act as members of multiple groups that they can also switch between applications, based on the agent they receive or their saved tasks. This approach is useful to allow
nodes to take different roles based on the required task, but on the other hand when a particular node working on a certain application receives an agent requesting to take a new role in a different application, this node would need to wait before it can join the new set of nodes running the new application. This clearly raises the issue of high latency as well, there should be a clear measure to the remaining residual energy of that particular node before any decision is made to switch to the next application. The last feature of scalability support is implemented through one-hop re-broadcasting of received multi-cast type agents. The normal approach recipient relays every agent that has been flooded throughout the network in a multicast fashion after its ID is saved in that node, this mechanism practically leads to increased network traffic causing congestion and packets collisions. We argue in our proposed model that the previous work achieved with middleware was more platform-specific such as on MICA2 and TinyOS, and would bring obstacles and challenges to developers when it comes to portability, that is modifying the existing middleware to run onto other hardware platforms (cross-platforms compatibility). However, although the previously developed middleware application/platform specific, it ran onto low cost sensor nodes with scarce amount of memory and limited processing capability in contrast to the one used in the paper by Microsystems such as Sun Spot, with luxury amount of data and program memory required of about 1MB and 8MB respectively as well as high processor speed of 400MHz.

In [30], the authors proposed a number of algorithms for the dynamic allocation of sensor nodes in WSN hosting multiple applications. The system model suggested is comprised of computing grid, WSN, and an interface proxy that provides protocols and API to access and manages the sensors in the underlying WSN. The allocation decision requires that prior information is available on both characteristics of an application and the state of the sensor node, most importantly the energy parameters pertaining to the energy status levels for the radio module and the CPU which will also aim towards improving the lifetime of WSN. Different initial energy statuses of sensor nodes were examined and it was revealed that sensor nodes with an equal amount of initial energy performed best. Latency may be introduced because continuous evaluation and feedback knowledge of sensor node energy levels need to be communicated back to the computing grid for allocation decision making; this would also affect the number of messages being transmitted in the network. It is also not known how the dynamic allocation decision will affect the network coverage when different sensors are being relocated to take up new roles in different applications.

Run-Time Dynamic Linking [31] is another proposed work that focuses on the reprogrammability in wireless sensor networks where system functionality update is required. The intended design is to update native code in a heterogeneous network. The proposed work implements an in-situ run time dynamic linker and loader that uses the standard Executable Linking File object (ELF Object)
file format. Although run-time dynamic linking is a sought-after approach for WSN, especially when deployed in inaccessible areas, its main drawback lies in the need for a special compressor utility for creating a compact ELF (CELF), as the ELF object file needs to be compressed to accommodate the 8-bit and 16-bit processors. Part of our proposed work is based on Agilla [32, 33] mobile agent middleware for sensor networks that establish in-network programming. Their architecture is based on tuple spaces and neighbor lists for agent coordination. Our proposed model follows a similar approach to Agilla but with some modifications related to introducing two agents for two specific functions.

The agent manager in Agilla architecture can handle a maximum of three agents due to limitation of processor speed and memory availability. Also Agilla uses two types of operations: WEAK and STRONG. Since the STRONG operation suffers higher overheads we have only adopted the WEAK Operation and also limited our architecture design to two agents to free memory space and to be in line with processor speed.

Since Agilla’s mobile agents are continuously on the move, migrating and cloning as required for the purpose of updates, configuration and new application code, they are heavily dependent on wireless links in the network which is somehow unreliable and could affect its efficiency and reliability. In our proposed design presented in [28], we have implemented lightweight mobile agents that would only act as an ON-OFF control agent in most cases. This uses less energy transmission when compared to Agilla’s mobile agents and serves the required purpose. In Table 2.1 below, the analytical work on concurrency at node level is summarized.
<table>
<thead>
<tr>
<th>Concurrency &amp; Programming at Node Level</th>
<th>TinyOS/nesC</th>
<th>TinyThread</th>
<th>Y-Thread</th>
<th>Mate/ASVM</th>
<th>Melete</th>
<th>IMPALA</th>
<th>SensorWare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allows concurrency at Node level and lets processor sleep as much as possible as it uses event-driven paradigm that enhances energy efficiency.</td>
<td>Efficient per-thread stack allocate as it uses stack estimator. It is often too expensive in terms of memory to statically allocating per-thread stack. With the blocking execution contexts, simplifies programs and can use procedural programming style.</td>
<td>Shared stack for computation and separate stack for control. It is often too expensive in terms of memory to statically allocate per-thread stack. With the blocking execution contexts, simplifies programs.</td>
<td>A tiny virtual machine architecture. Stack oriented on top of TinyOS. Had limited flexibility and minimal support for concurrency. These issues were then addressed by the application of a specific virtual machine.</td>
<td>Supports concurrency at both node and network level. It is also based on the Mate virtual machine concept.</td>
<td>Both are middleware based. IMPALA allows software to be updated dynamically. Sensorware supports TCL-based script language for reprogrammability However, they require the luxury of more memory usage since they were designed for iPAQ pocket PC devices.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Concurrency &amp; Programming at Group Level</th>
<th>Neighborhood-based Group</th>
<th>Logical Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>These are defined by physical closeness. Abstract Regions and Hood provide programming primitives based on neighborhood. Neighborhood based groups are mostly static.</td>
<td>These are based on sharing some logical properties such as node type and sensor reading. These logical groups are more dynamic as the group membership is often determined by dynamic properties such as sensor input.</td>
<td>Group level abstractions feature good collaboration among sensors as they support aggregation at group level through data sharing as well as being efficient within the group, however, we think when it comes to multi-application running, it would be very sophisticated to control and manage multi-applications due to the nature of distributed groups.</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of related work on concurrency at different node levels (adapted and modified from [19])
<table>
<thead>
<tr>
<th>Concurrency &amp; Programming at Network Level</th>
<th>Database</th>
<th>Other Network level reprogramming and concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cougar and TinyDB are two examples of networking level. While their in-network query processing and collaboration are declaratively described by query, these are only suitable for describing query operations to a sensor network.</td>
<td>Run-Time Dynamic Linking is another proposed work for reprogrammability in wireless sensor networks where system functionality update is achieved. The intended design is to update native code in heterogeneous network. The proposed work implements an in-situ run time dynamic linker and loader that uses the standard Executable Linking File object (ELF Object) file format. Although run-time dynamic linking is a great approach for WSN, its main drawback lies in the need for a special compressor utility for creating a compact ELF (CELF) as the ELF object file needs to be compressed to accommodate the 8-bit and 16-bit processors. Part of our proposed work is based on Agilla’s mobile agent middleware for sensor networks that establish in-network programming. Their architecture is based on tuple spaces and neighbor lists for agent coordination. Our proposed model follows a similar approach to Agilla but with some modifications. The agent manager in Agilla architecture can handle a maximum of three agents due to limitations of processor speed and memory availability. Also Agilla uses two types of operations, WEAK and STRONG. Since the STRONG operation suffers a higher overheads we have only adopted the WEAK Operation and also limited our architecture design to two agents to free memory space and to be in line with processor speed. Since Agilla’s mobile agents are continuously on the move migrating and cloning as required for example for updates, configuration and new application code, they are heavily dependent on wireless links in the network which is somehow unreliable and hence affects its efficiency and reliability. In our proposed design we have implemented lightweight mobile agents that would only act as an ON-OFF control in most cases. This uses less energy transmission when compared to Agilla’s mobile agents.</td>
<td></td>
</tr>
</tbody>
</table>
2.2 Running and Supporting Multiple Applications in WSN

Initial research work on WSN was primarily focused on networks running single-application or application-specific networks: these application-specific networks were the start of the first generation to be implemented. The many challenges posed by WSN with deployment, size and communication protocols as well as scarce resources related to power, memory and processor speed have attracted much research work. The new emergence in technology associated with the advances in semiconductor technology has made sensor nodes become more multi-functional devices, as well, coupled with the increased demand to run multiple applications on the sensing field, a new research direction and paradigm have shifted towards running multiple applications on a single network for the ultimate goal of a better return on investment. Most of the WSNs are designed to only run a single application over the entire network lifetime, even when it comes to large scale implementation. Our research has been primarily focused on those large scale dense networks that should not only serve to run a single application but are also capable of executing multiple applications when needed. Running only one application results in limiting the use of numerous versatile sensor nodes and is a major source of system inefficiency even for the purpose of redundancy. Resource wastage in this case is not the only factor, but also such design does not fully exploit the enormous capabilities that these WSNs possess of self-adapting, self-configuring etc.

These were some of the motivating factors that led us to work on the proposed research topic that aims to utilize a dense network for running multiple applications on what we called Dynamic Switching Sets (DSS), implemented by a new algorithm to alternately switch different parts of the network to different application needs. To that end we have also published four research papers from our work on running multiple applications on WSN and prolonging network lifetime [34, 28, 35, 36]. The new concept of dynamic switching sets (DSS) is based on partitioning a single network to multiple sets of nodes, each set comprising a number of DSS spread over the entire network such that coverage and connectivity are maintained. Each DSS is to run one single application at a predefined sequence controlled by the base station. The idea of running multiple applications on a single network has gained significant momentum in recent years with much research work investigating different effective approaches on the design concepts of both
hardware and software systems to achieve the goal of running multiple applications on a single network.

In [37] the work proposed adds many new challenges to the design and implementation concepts of WSN since it is not only suggested to run different multiple applications but also with heterogeneous sensor nodes deployed in multiple networks employing multiple base stations and a web server. It is also suggested to have these sensor nodes be part of the new emerging technology internet of things (IoT) where sensor data can be accessed through the web. The main challenge in this new trend of multiple applications is the ability to effectively and dynamically coordinate the sharing of available resources for best optimization while at the same time maintaining the goal of meeting application requirements. The authors propose the use of a middleware platform solution that can provide high flexibility for adding advanced and sophisticated functions to implement the required algorithms for their approach, called the service oriented approach. In this approach sensor nodes are considered as service providers and the middleware layer acts as an interface between the applications. This would access a set of services provided by abstractions of the complex underlying layers through some standard interfaces based on the service-oriented architecture [38]. The work presented in [37] is an algorithm called resource allocation in heterogeneous sensor networks (SACHSEN) which is the core of the resource allocation component of their middleware layer. This algorithm addresses the aforementioned challenges by running multiple applications onto heterogeneous WSN. The approach suggested by SACHSEN would seem to be effective only when the different applications could share the same sensing data when these data have common characteristics in time and space and similar quality of service requirements (QoS). It is not clear how SACHSEN is applied when different data are gathered from different sensors. A similar work on scheduling of multiple applications in distributed and heterogeneous networks was also presented in [39]. The authors proposed several resource allocation algorithms that carefully constrain the amount of resources that can be allocated to each running application. In [40], the work carried out was for supporting multiple concurrent applications by working out the best trade-off between fidelity-aware resource allocation and the sensor node selection. Fidelity can be defined to represent a concept for an application that can take a variety of operational measures such as data quality and communication latency. The trade-off in this case arises when the algorithm starts to reduce the value of fidelity of several tasks in the applications in order to be able to run more simultaneous applications in the WSN, system and this results in reducing the quality for that application. The work presented in [41] is an interesting different direction for running multiple application in WSN. These deployed networks are aimed to be integrated to the internet of things (IoT). The internet of things is a
networking paradigm evolved on the premise of large scale deployments of two important well-established technologies: radio-frequency identification (RFID) and WSN [42]. With the recent research directions on scalability in WSN and the emergence of IoT as a new platform that is primarily realized on meeting scalability demands, the integration between the two would make a good optimization of resources. This work would allow WSN to benefit significantly from the resource pool that IoT utilizes, which comprises all pervasive technologies and services such as cell phones, municipal cameras and data collectors. Also, resource sharing and cross-utilization between WSNs and nearby architectures would also give additional benefit to utilize their resources. However, the biggest challenge presented in the paper is how to interweave WSN in the greater paradigm of IoT rather than just to tailor it to work for it. The work proposed aims to move away from WSNs as dedicated systems for sensing tasks to generic platforms of dynamically assigned resources. Nodes in this scenario are viewed as resource providers and the ability to assign measurable attributes to these resources would make it possible to better use them and utilize them to leverage operational capacity across multiple WSN platforms and hence multiple applications could be adopted to run concurrently on different WSNs. The approach idea is divided into three phases: 1) the first phase looks into measuring the available resources in the network and their usability 2) the second phase involves representing applications into finite sets of functional requirements and 3) the last phase looks into optimal mapping between applications and resources. It aims for dynamically accommodating varying resources being introduced or removed as well as utilizing any passing by transient resources in their vicinity. The aforementioned work presented on running multiple concurrent applications with the integration of IoT is a promising trend that will have much research work invested. However there are still many obstacles (IP address space and allocation to things, availability of sensor nodes on the internet and adapting to large scale and control overhead [43]) and challenges that need to be addressed and resolved first in order to translate this into realistic work for implementation. In [44], the authors present a framework to run two applications on a single deployed WSN. The proposed work is for a deterministic deployment scenario of wireless sensor nodes in an indoor office building, which assumes the luxury of power replenishment availability to any individual sensor node. The two applications, Collection and Firecam need to run at different timings. Collection: is the default application in the sense of being executed continuously during normal network operation primarily to monitor building environment parameters such as temperature, humidity, light, and motion PIR detectors. The second application, Firecam, is executed at emergency events only, specifically when a smoke detector or security sensor goes off in one of the building’s zones. Sensor nodes that happened to be installed in that zone would take a series of picture shots and send them back to the base station.
In the second application it can be seen that a large amount of data stream needs to be transmitted and routed back to the base station. These two scenarios obviously require two different configurations with regard to the MAC and network protocols as the first application operates with low-rate reliable many-to-one collection versus the requirement of the second, point-to-point, low-latency, and bulk-data streaming. The authors propose a framework that at design time, allows the settings of Network, MAC and Radio protocols for each application as well as the threshold and events that would trigger the WSN reconfiguration. Hence, at run time, the WSN automatically reconfigures itself in response to these events and to the criteria at which they were set at the design stage.

In [24], the authors present scoping, a general concept to create node subsets on a network for the purpose of achieving multi-purpose task network. Scoping is achieved by selecting a group of nodes through a membership condition such as node properties, for example, nodes that share the same temperature-sensing feature can subscribe to one group and so on. This is similar to our work presented since each of the dynamic switching set nodes would run different sensing application. In order to delimit the scope of applications to a subset of nodes, the paper mentions that it is required that separation of applications is made at both node and network levels but it is not clear on how this is implemented. Although this concept is useful to run multiple tasks on a network, these groups will eventually remain geographically isolated and confined only to their areas in terms of coverage and connectivity and not to the entire network. As well, the concept presented may only be applied for a deterministic deployment strategy as it is not clear how this can be implemented when a stochastic type of deployment is required. A similar research study on hosting multiple applications in WSNs is presented in [30]. The paper suggests the dynamic allocation of sensor nodes constituting a number of groups, each of which hosts one application. This concept depends on prior knowledge of application characteristics and the state of the network. The information needed in this case will serve as input is collected from queries and application requests issued by the users of applications. This information is also modeled using a traffic source with stream of application requests following a Poisson arrival process. The requests are then submitted to a proxy stage to process and send request messages to cluster heads in the network.

Another work in [45] is carried out on supporting multiple concurrent applications using multi-agents. The authors propose a middleware solution called ‘SensoMax’ that runs on Java-based platforms in a modulated architecture that supports running multiple concurrent applications. Although the proposed SensoMax middleware solution aims to provide high standard features
needed for WSN such as macroprogramming, multi-tasking and multiple concurrent application support, it requires the availability of luxury and rich resources. These resources in hardware and software platforms which are rarely available when designing WSN because these sensors are characterized by their constrained resources and cost efficiency. The hardware targeted for the Sensomax such as Sun Spot and Raspberry Pi [46] are resource-rich hardware platforms, for example, Sun Spot devices feature a 400MHz ARM processor, 1MB of data and 8MB of program memory etc. As well, the paper discusses dividing the entire area to be monitored into separate regions, each of which runs a different application, similar to the concept of scoping discussed in [24], this would limit network connectivity and coverage to a specific region where tasks are being executed and not necessarily the entire network.

Some new research work has also been focusing on running multiple applications in WSN for multimedia purpose. As we see in [47], work is presented to support multiple applications in wireless multimedia sensor networks. The proposed research aims to achieve enabling nodes to transmit audio and video streams along with scalar data. The framework concept is primarily focused on the cross-layer architecture for the internal node design concerning network and MAC layers and the associated communication and routing protocols. The design concept works on reserving a sufficient amount of memory as a separate space where application-specific parameter settings for different layers of the protocol stack are stored and managed by an additional middleware layer called Cross Layer Optimization Middleware (CLOM). Those saved parameters and their values are then used by different layers of processing data packets for the applications that requested the settings. Despite that fact that Multimedia WSN is a promising future work, however, the feasibility of implementing in a harsh sensor environment is still in question. This is because video data streaming in particular requires a robust and reliable wireless channel with sufficient bandwidth and data rate as well as back-up power sources for the high energy consumption in this case. Also the cost of deploying a mass number of these sensor nodes is significantly high, which defeats the purpose of utilizing a dense network.

2.3 Deterministic and Stochastic Coverage

One fundamental issue in sensor networks is the coverage problem to the region of interest which is measured by how well a sensor network is monitored or tracked by sensors. Since sensors may be spread in an arbitrary manner, a fundamental issue in WSN is to ensure that coverage and connectivity are maintained. Given a sensor network, the coverage issue is concerned with how well the sensing field is monitored by sensors [48]. Coverage is one of the essential issues we have taken into account in our proposed algorithm to run multiple applications in WSN. The fact that
our sensing field is divided into multiple dynamic sets of which only one set is active at a time while other two sets are idle/in sleeping mode, it is imperative that the active running set (comprised of multiple dynamic switching groups of nodes spread over the entire field) takes care of the adjacent sleeping sets in terms of sensing coverage and connectivity during its life cycle.

The first step to decide upon is whether the coverage is deterministic or stochastic. The coverage problem has been studied under different objectives, depending on the requirements and constraints of the applications. If the location of the deployed sensors can be preselected, the coverage problem reduces to the problem of finding the optimal placement for sensors such that target coverage is met and this type is the deterministic coverage [49].

However, for large sensor networks, it is impractical to perform deterministic coverage for the field of interest: first, the number of sensors that need to be placed is often prohibitively large, and second, due to the type of environment where they are deployed. As an example, sensors may be dropped off an aircraft into a forest in order to monitor environmental parameters such as humidity, temperature, air quality etc. Instead, sensors are deployed in the field of interest (FOI) according to a preselected distribution. For stochastically deployed sensor networks, the coverage problem quantifies how well the FOI is monitored when a number of sensors are deployed according to a known distribution [50].

There has been much work done on the coverage problem. In [51] the approach tries to look at how the perimeter of each sensor’s sensing range is covered which leads to a polynomial algorithm. Basically it determines whether every point in the service area of the sensor network is covered by at least \( k \)-sensors, where \( k \) is a predetermined value. When it comes to a large number of sensor deployments, the polynomial algorithm becomes computationally expensive. Here, the solution for determining perimeter coverage requires each sensor to communicate with all the sensors within twice its sensing range, then that solution is used to determine redundancy and schedule inactive periods for redundant sensors. However, to determine its redundancy, a sensor \( s \) has to ask all the sensors within twice its sensing range to reevaluate the coverage of their perimeter without \( s \). Thus, a sensor has to run the perimeter coverage \( N \) times, making the complexity of the protocol \( O(N^2 \log N) \). Reference [2] proposes a heuristic to select mutually exclusive sets of sensor nodes such that each set of sensors can provide a complete coverage to the monitored area. Some works are targeted at particular applications, but the central idea is still related to the coverage issue. In [3], a distributed node-scheduling protocol is proposed that also increases network lifetime by turning off some redundant nodes. This is basically done by a concept of an effective sensing area (ESA), where it refers to the sensing area that is not overlapping with other sensors’
sensing area. Here a sensor node determines whether it will be active or not after calculating its own ESA.

Our proposed algorithm, when it comes to the coverage problem, needs to consider large dense networks and we have therefore adopted the work proposed in [52] where Zhang and Hou proved the interesting results that if the transmission range of sensor nodes is equal or exceeds twice the sensing range, coverage implies connectivity of the sensor network. In [53] we see that a global framework is proposed for both deterministic and stochastic sensing models of the sensors. Their proposal goes into K-coverage characterization, which is based on Helly’s Theorem and Reuleaux triangle and K-coverage preserving scheduling and this checks each sensor if its eligible to turn itself on.

2.4 Prolonging Network Lifetime in WSN

Wireless sensor networks, as shown in Figure 2.2 being self-configuring networks can be invaluable in many civil and military applications for collecting, processing, and disseminating a wide range of complex environmental data. Sensor nodes are battery-operated devices, and hence operate on an extremely constrained power source. Additionally, they must have a lifetime of the order of months to years, since battery replacement is not feasible nor is economical as an option for a network with thousands of physically embedded nodes. This transforms energy consumption into the most important factor that determines sensor node life time [54].

Conventional low power design techniques [55] and hardware architectures only provide point solutions which are insufficient for these highly energy constrained systems. Energy optimization in the case of sensor networks is far more complicated, since it involves not only reducing the energy consumption of a single sensor node, but also maximizing the life time of the entire network. The network life time can be maximized only by incorporating energy-awareness into every stage of wireless sensor network (node and network levels) design and operation [54].
In past decades, energy conservation and power efficiency of wireless sensor networks have attracted the attention of a vast number of researchers looking into different aspects and challenges of system and network design, the prime goal being to prolong network life time. Our literature and related work on this crucial topic has mainly classified energy saving in WSN into three categories as shown in Figure 2.3 below.

**Figure 2.2:** Wireless Sensor Network Model (adapted and modified from [56])

**Figure 2.3:** General classification of energy saving in WSN (adapted and modified from [56])
2.4.1 Node-level Power Management

A great deal of research has been carried out investigating the best possible internal node architecture (microcontroller, radio module transceiver, memory storage, and sensor board) design towards optimum energy conservation. In [56], Anastasi et al list some generic aspects of energy breakdown that apply across many different sensor nodes, although still heavily dependent on the specific node [57]. These are:

- The communication module has an energy consumption much higher than the processing microcontroller module. In [58], work has shown that transmitting one bit of data may consume as much as some few thousands of instructions executed by the processor.

- The radio module consumes roughly the same magnitude of energy when receiving, transmitting or in an idle state. It is therefore the duration length at any of the three states that ultimately counts and hence should be turned off (put to sleep) whenever possible.

- The sensing module and depending on the specific application could also be consuming significant amount of energy that would need to be reduced.

As previously mentioned, the node’s radio module is the most energy-consuming part that needs to be effectively controlled. One way is duty-cycling for the transceiver module such as putting it in low-power sleep mode whenever communication is not needed. Ideally, the radio should be switched off as soon as there is no data to send/receive and put back active (turned on) once a new data packet becomes available. In this way, nodes alternate between active and sleep periods depending on network activity. This duty-cycling approach operated on active nodes plays an important part of the effectiveness of power management at node level. This approach can be implemented as independent protocol running along the MAC layer or integrated part of the MAC protocol itself [56].

Node power management can be classified into two categories: sleep/wakeup protocols and MAC protocols adapting low duty cycles. The sleep/wake up protocols can be further divided into three main categories: on-demand, synchronized wakeup/sleep, and asynchronous schemes.
2.4.1.1 Sleep/Wake-up Protocols

On-demand Scheme

The basic idea of on-demand schemes is that a node should only wake up when it has to receive a packet from a neighboring node. This clearly minimizes energy consumption and makes this scheme particularly suitable for sensor network applications with a very low duty cycle, such as fire detection, surveillance of machine failures and generally for all event-driven applications. In these cases, sensor nodes are only sensing the environment, that is they are in the monitoring state, and as soon as an event is detected, nodes transit to the transfer state. While the main aim of these scenarios is reducing energy consumption in the monitoring state, they also need to ensure limited latency when transitioning from sleep to transfer state. The issue in this case is how to let the sleeping node know that some other node needs to communicate with it. To achieve the best implementation for this type of scheme, two different channels are required, a data channel for normal data communication and a wakeup channel to wakeup nodes when needed. This suggests that two different radios would be required on the same node, one for signaling that there is a node that wants to communicate, and this type of radio tends to be with low power and low rate type, and the second radio needs to be with higher power for data communication once the node transits from sleeping mode. Despite the fact that this would incur additional cost for the second radio, however, it would allow not to defer the transmission of a signal on the wake-up channel if a packet transmission is in progress on the other channel and thus reducing the wake up latency.

Sparse Topology and Energy Management (STEM) [59] uses two different radios for the wakeup signal and data packet transmissions respectively. Both radios are of the same power to avoid problems associated with different transmission ranges. Hence, an asynchronous duty cycle scheme is used on the wakeup radio as well. Each node, for every T-duration, would periodically turn on its wakeup radio for $T_{active}$ time. In this case, every time a node (initiator) wants to communicate with another neighbouring node (target), it sends a stream of periodic beacons on the wakeup channel. Once the target node receives a beacon, it sends back a wake-up acknowledgment, and turns on its data radio. In addition to the STEM-B approach, another new approach, suggested by [60] and referred to as STEM-T, uses a wakeup tone instead of a beacon. The difference between the two is that in STEM-T all the nodes in the neighbourhood of the initiator are awakened. When a combination of either STEM-B or STEM-T is used with topology control protocol energy saving is achieved, for example, in a practical case the combination of Geographically Adaptive Fidelity (GAF) [61] and STEM can reduce the energy consumption to about 1% of that of a sensor network with neither topology control nor power management [60].
That shows network life time is increased, but on the other hand a tradeoff for path setup latency is taking place. In STEM the inter-beacon period is such that there is enough time to send the wakeup beacon and receive the related acknowledgment.

Let $T_{\text{wakeup}}$ and $T_{\text{wack}}$ denote the time required to transmit a wakeup beacon and the related acknowledgment, respectively. Since nodes are not synchronized, the receiver must listen on the wakeup radio for a time $T_{\text{active}}$ at least equal to $2T_{\text{wakeup}} + T_{\text{wack}}$ to ensure the correct reception of the beacon. The $T_{\text{active}}$ period on the other hand depends on the bit rate of the nodes in the network, therefore for low bit rate networks, the time between successive active periods (T) must be very large to allow a low duty cycle on the wakeup channel. This results in large wakeup latency, especially in multi-hop networks with a large hop-count [62].

To achieve tradeoff between energy saving and wakeup latency, [63] proposes a Pipelined Tone Wakeup (PTW) scheme. Similar to STEM, PTW relies on two different channels for transmitting wakeup signals and packet data, and uses a wakeup tone to wake neighboring nodes. PTW uses a different approach in that unlike STEM, the burden of tone detection is shifted from the receiver to the sender and that result in the duration of the wakeup tone is long enough to be detected by the receiver that turns on its radio periodically. The main reason behind this solution is that the sender only sends a wakeup tone when an event is detected, while receivers wakeup periodically. As well, the wakeup procedure is pipelined with the packet transmission so as to reduce the wakeup latency, and hence the overall message latency. Figures 2.4 and 2.5 below illustrate the main idea of PTW. In Figure 2.4, let us assume that node A needs to transmit a message to node D through nodes B and C. At time $t_0$, A begins the procedure by sending a tone on the wakeup channel that awakens all A’s neighbours. At time $t_1$, A sends a notification packet to B on the data channel to inform that the next data packet will be destined for B. When the notification message is received by all of A’s neighbours, they will learn that the message is not intended for any of them but only for B and they will turn off their radio. B will then realise to be the destination of the next data message and replies with a wakeup acknowledgment on the data channel. Then, A starts transmitting the data packet on the data channel and at the same time, B starts sending a tone on the wakeup channel to awaken all its neighbors as shown in Figure 2.5 below. The packet transmission from A to B on the data channel and B’s tone transmission on the wakeup channel are done concurrently. The data transmission and control are regulated by the underlying MAC protocol.
Since the energy consumption of the wakeup radio is generally not negligible, we can see that both STEM and PTW use an asynchronous sleep/wakeup scheme for enabling a duty cycle on the wakeup radio as well. Using a different approach to minimize the wakeup latency, a low power wakeup radio is proposed. In this case the low power radio is continuously in stand-by mode and whenever it receives a signal, it wakes up the data radio [62, 64, 65, 66]. However, the main drawback in this approach is that the transmission range of the wakeup radio is significantly smaller than that of the data radio. This would limit the applicability of such a technique as a node may not be able to wake up a neighbouring node even if it is within its data transmission range. For example, in [66], the low power radio operates at 915 MHz (ISM band) and has a transmission range of approximately 332ft in open space, while the IEEE 802.11 card operates at 2.4 GHz with a transmission range up to 1750ft.
In all the previously discussed approaches of using a second radio for the wakeup channel, there is always a side effect of the additional power consumption even when using a low power radio. To tackle such problems associated with the extra energy consumed by the wakeup radio, a Radio-Triggered Power Management scheme is investigated [67]. The basic idea in this case is to use the energy contained in wake-up messages (beacon in STEM-B, and tone signal in both STEM-T and PTW) to trigger the activation of the sensor nodes. This is similar to the principle used in active Radio Frequency Identification (RFID) systems [68]. Figure 2.6 shows the radio-triggered scheme in its simplest form where a special hardware circuit is used to capture the energy contained in the wake up message which can then be used to trigger an interrupt for waking up the node. The approach of radio triggering is significantly different than using a stand-by radio to listen to possible wake up messages from neighboring nodes. The stand-by radio consumes energy from the node while listening, whereas the triggered-radio circuit is powered by the wakeup message. The limitation of the maximum distance from which the wakeup message can be sent is the main drawback of the radio-triggered approach.

![Figure 2.6: Radio triggered power management (adapted from [56])](image)

**Synchronized wakeup/sleep**

The concept here is to control the wakeup and sleep timings of all nodes at the same time. This would guarantee that when all nodes are active they can communicate and broadcast messages to each other effectively. However the drawback in this scheme is the need for a network to be synchronized with clock synchronization, which is an ongoing research area and since it is beyond the scope of our work, it can referred to in [69, 70]. There could also be some network message collisions as the traffic congestion builds up as all nodes will start to send messages at the same time. There are different scheduling protocols for the sleep/wake-up concept and they differ in
the way these sensors wake and sleep during their life cycle. In [71], a Fully Synchronized Pattern scheme is suggested, which is regarded as the simplest way and has been widely used in practical implementations. In this case a periodic pattern is applied to all nodes in the network where they all wake-up at the same time. Sensor nodes periodically wake-up for every time span $T_{\text{wakeup}}$ and remain active for another time slot $T_{\text{active}}$ after which they return back to the sleep state until the next wakeup instant arrives. Although it is a simple scheme, its effectiveness lies in the fact that it provides low duty cycle provided that the active time $T_{\text{active}}$ is significantly smaller than the wakeup time given by $T_{\text{wakeup}}$. Even when these nodes come to the active time slot they can have some timeout, after which their radios can be switched off, further enhancing energy conservation.

Our proposed algorithm adopts a similar approach with regard to dynamic switch sets and the nodes wake and sleep states. The only difference in the scheme pattern is that our adopted algorithm gives the flexibility of not only timing those sets of nodes for when they wake-up and sleep but also can be controlled by the base station to adjust their active and sleeping times to accommodate the duration needed by each application to run during the network life cycle. The sleep/wake-up mechanism we used follows in line with the internal network organization that leads to the creation of Dynamic Switching Sets (DSS).

A different approach [72] is used based on the Time Division Medium Access (TDMA) scheme, called Flexible Power Scheduling (FPS). In FPS time is divided into a number of slots that add up to make a cycle that becomes periodic. Each slot is of duration $T_s$ and with a cycle made up of $m$ slots will have a duration time of $T_c = m \times T_s$. A node will also need to maintain a power schedule of what operations to perform during a cycle. Controlling its radio when to receive and transmit is another important operation a node adds to its schedule. Having said that, these schemes suffer from the fact that a strict synchronization among all nodes needs to be maintained.

**Asynchronous Scheme**

In the asynchronous approach, we mainly discuss research work that has been implemented for large density scale networks to be in line with our proposed work that applies only to wireless sensor networks characterized by a high node density. The work presented in [73] is a protocol named Random Asynchronous Wakeup (RAW). The RAW protocol is a combination of a routing protocol and random wakeup mechanism where the routing in this case is a variant of geographic routing. The similarity between the geographic routing and the routing implemented in RAW is basically the approach in which packets are sent. In geographic routing packets are normally sent to a neighbour that is closest to the destination, and in the case of RAW, packets are forwarded to
any of the active neighbour nodes that belong to a set called the Forwarding Candidate Set, membership of which must meet pre-selected criteria. When it comes to the random wakeup scheme, the idea is that each node wakes up randomly once in every time interval of fixed duration $T$, and remains active for a predefined time $T_a$ where $T_a \leq T$ after which it sleeps again. When a node wakes up, it runs a neighbouring discovery procedure to see if there is any available active node in the nearby neighbourhood. If, for example, node A needs to transmit a packet to a destination node Y, and that is within the forwarding set of A, there are $m$ neighbours as possible forwarders to Y, then the probability that at least one of these neighbors is awake along with A is given by:

$$P = 1 - \left(1 - \frac{2T_a}{T}\right)^m$$

(2.1)

In a dense network, the number ($m$) of neighbour in the forwarding candidate set is also large and hence equation (2.1) would result in large probability $P$ to find an active neighbor to which to forward the packet.

This scheme is also suitable for frequent topology change networks in dense networks, however it is not suitable for sparse networks. Nor does RAW does not guarantee the packet forwarding within one time frame ($T$), however, due to the dense network it is very likely that some of the neighbours are awake when a packet is needed to be forwarded. One other approach is to have to force the receiver to periodically wake up and listen in case one potential asynchronous sender needs to communicate with it. Hence the receiver wakes up for a short time to see if there is a sender that needs its attention, otherwise it returns to the sleep state.

### 2.4.1.2 MAC Protocols

An essential characteristic of wireless communication is that it provides an inherently shared medium. All medium access control (MAC) protocols for wireless networks manage the usage of radio interface to ensure efficient utilization of the shared bandwidth. MAC protocols for wireless sensor networks have been massively researched for an additional goal of managing radio activity to conserve and save energy. Energy efficiency is one of the critical emphases for MAC protocols while at the same time balancing throughput, delay and channel access. One common theme through all MAC protocols is putting radios to a low-power ‘sleep mode’ either periodically or whenever no activity of receiving or transmitting is detected [74].
Asynchronous Sleep Techniques

As previously discussed nodes need to be able to sleep when there is no communication activity for the sake of saving energy and need to only be able to wake up and participate in any necessary communication. A number of research projects [74] have been carried out into a hardware solution where each sensor would be equipped with two radios, one for data and the second is a low-power wake-up radio. The data radio is the primary and main module that would remain asleep by default, whereas the wake-up radio remains on all the time. When the wake-up radio receives a wake-up signal from another node, it responds by triggering and waking up the main data radio to start receiving. With that configuration energy saving is achieved, however since all nodes are equipped with the same radio configuration, this means when a signal is received those nodes in the broadcast domain of the transmitting node may also wake up as they receive the same signal.

A different approach for waking up sleeping radios using a preamble RF pulse was developed [75, 76]. In this technique, which is also called preamble sampling or low-power listening, receivers periodically wake up to sense the channel to see if there is any activity and if not then go back to sleep. When a particular node needs to transmit, it sends a preamble signal prior to packet transmission, once this preamble is detected by the receiving node it changes to full active receive mode, (Figure 2.7). Traditionally, the wake-up signal is sent within the packet frame, however a more efficient approach developed by [75, 76] is to implement this directly in the physical layer which could simplify the wake-up signal to just a long RF pulse. In this technique the detecting node would monitor the channel to check for radio energy on the channel to determine whether the signal is present. This could potentially wake up receivers in a given transmitter’s neighbourhood while on the other hand embedding such information in the header could allow mechanisms to be used to put nodes back to sleep if the communication is not intended for them. Another drawback of this technique is that the long preamble that the transmitter needs to send can sometimes cause reduction in the throughput as well as energy wastage for both sender and receiver.
A similar technique to lower-power listening/preamble sampling is suggested by [66] and is called TICER/RICER Techniques. In the transmitter-initiated cycle receiver technique (TICER), the receiver node wakes up periodically to check if there are signals from the sender, i.e. wake-up request to send (RTS) signal. The sender as shown in Figure 2.8 (a) sends a sequence of such RTS signals and as soon as the receiver detects the signal it responds with clear to send signal (CTS). The transmission begins when the sender receives the CTS signal. It can be seen that the main difference here between the preamble sampling and the TICER is that, instead of sending a long preamble, the TICER sends a sequence of interrupted signals and awaits a response from the receiver, as handshake acknowledge protocol, before initiating transmission.

The second technique shown in Figure 2.8 (b) is RICER, where a receiving node periodically wakes up to execute a three-stage sequence, monitor-send, wake-up, and beacon-monitor. A node that wishes to transmit stays active monitoring until it hears a beacon from the receiver, after which it begins data transmission. On the other side when a receiver sees packet transmission it remains on until the packet reception is completed.
The implementation of the TICER/RICER on a low – low power RF analog level is the main challenge facing this technique, as it needs to match the transmission to the correct receiver, and hence the receiver needs to uniquely identify itself to the transmitter [74].

The reconfigurable MAC protocol [77] (B-MAC) is another implementation of low-power MAC protocols. This protocol provides the functionality and interface that allows necessary tuning to take place depending on higher layer needs. The main features of B-MAC that can be tuned on/off and used in any combination are:

- Low Power Listening (LPL) – this implements the preamble wakeup technique and allows selection by higher layers to different channel sampling and preamble durations that contribute to conserving energy.

- Clear Channel Assessment (CCA) – this checks on the channel to determine when it is busy or free. When CCA is disabled, a scheduling protocol may be implemented and, if enabled, the back-off duration can be adjusted by the higher layer when the channel is found to be busy.

\[\text{Figure 2.8: Asynchronous sleep using TICER/RICER (adapted from [74])}\]
• Acknowledgments (ACK) – this feature, if enabled, allows an immediate response to be sent after receiving any unicast packet.

The B-MAC has proved useful in many practical settings for its ease of configuration and high efficiency. Figure 2.9 shows components of B-MAC and their memory requirements.

![B-MAC Components Table](image)

**Figure 2.9:** B-MAC Components with memory requirements (adapted from [74])

**Sleep Scheduled Techniques**

There have been many MAC protocols implemented with the sleep scheduling techniques for energy efficiency goals, three of which we will be discussing: Sensor MAC (S-MAC), Timeout MAC (T-MAC), and Data-gathering MAC (D-MAC) protocols.

The S-MAC [78, 79] was specifically designed for WSN where all the nodes follow a listen-sleep schedule based on a scheduled duty-cycle, as shown in Figure 2.10, which guarantees reduction of energy consumption.

![S-MAC Duty Cycle](image)

**Figure 2.10:** S-MAC, Sleep/wake duty cycles (adapted from [74])
In this technique, a node can either become a follower when adapting a neighbour’s schedule or a synchronizer if it uses its own schedule, which will also broadcast it to the neighboring nodes. When at the initialization stage, nodes wait for a random amount of time expecting to receive a message providing the sleep-listen schedule if they do receive such a message, then as mentioned previously, they become a follower. If after a certain amount of time nothing is received then the node becomes a synchronizer and uses its own schedule. In order to accommodate any new nodes joining the network, the exiting nodes sustain a constant periodic schedule transmission while at the same time need to exchange synchronization packets. This is only achieved because of the very large listening period when compared to the clock drifts. While this protocol does contribute to the reduction of energy consumption, however, a potential problem is the latency resulting from significant sleep time as the packet travelling across the network may have to pause during the sleep period of intermediate nodes and at every few hops.

The T-MAC protocol [80] is similar to the S-MAC but with an added adaptability feature of duty-cycle modification. The length of the cycle is constant but the end of the active period is dynamically determined by the use of a time-out mechanism. A node waits for the time-out to elapse and if nothing is received then it goes to sleep, however if it receives a control or data message then the timer starts fresh after the reception of the message. The early sleep problem is the drawback of this protocol and also has an impact on the data throughput.

It can be concluded that both B-MAC and T-MAC provide energy savings at the expense of delay and latency, especially for packets traversing multiple hops. As they move from one hop to another, these packets will eventually get interrupted and hence delayed when they reach a node that must go to sleep. A protocol called data-gathering MAC [81] or D-MAC was developed to overcome this problem, but it is only an application-specific protocol, which is not a general purpose protocol since it applies to one data gathering tree as explained below. This protocol applies only to flows on a predetermined data-gathering tree moving from different nodes up towards the base station. A staggered sleep schedule is applied, where nodes at different tree levels follow a sequence of receive-transmit-sleep shifted to the right, as shown in Figure 2.11 below. The cycles here are aligned so that a node at certain level $k$ is in receiving mode while another node below it on the tree at $k+1$ is in transmitting mode. This allows minimum delays for data and control packets to sequentially traverse all the way up a tree.
2.4.2 Efficient Data Delivery

Data driven approaches play an additional and important role in improving energy efficiency. The sample data may not all be needed and those unneeded ones result in useless energy consumption, even if the cost of sampling is negligible, because they result in unneeded communications. In [56], several data reduction schemes are discussed, one of which is in-network processing, where average data values are computed resulting for data aggregation at intermediate nodes between the sources and the sink. Data compression is another scheme that can be applied to reduce the amount of information sent by source nodes. This scheme involves encoding information at nodes that generate data, and decoding at the sink. The gains from in-network compression can be significant in terms of energy consumption reductions. A common case could be that the data from a number of sensor nodes (sources) can be combined into a single packet (e.g. suppressing identical data generated from the nearby sources). In this case, if there are $k$ sources all located close to each other and far from the base station, then a method of combining their information can achieve $k$-fold reduction in transmission as compared with each node sending its information separately without compression. Another aggregation service called Tiny Aggregation (TAG), presented in [82], is an aggregation service for low-power distributed wireless ad hoc networks, such as sensor networks. In this method, an SQL-like interface is provided to query and aggregate the interested attributes from the network. Through the base station, TAG distributes the aggregation queries into the network, and sensors route requested attributes of matched data back to the base station through a routing tree rooted at the base station. As packets are sent towards the base station they get aggregated along the tree by the intermediate nodes (aggregation is where...
information is condensed from information provided by nodes further away from the sink. Because of the in-network aggregation and disregarding irrelevant data, TAG reduces the number of transmissions, thus achieving energy efficient data gathering. The TAG algorithm consists of two phases, the distribution phase and the collection phase. In the distribution phase, aggregate queries are propagated throughout the network. In the collection phase, aggregate values are gathered from the network. The aggregate queries are SQL-like syntax language. One common application example is to collect the in-door temperature in a building, in this case the query we may issue is similar to the following:

SELECT AVG (temperature), room FROM sensors
WHERE floor = 2
GROUP BY room
HAVING AVG (temperature)>80
DURATION 60s

This query collects the average temperature and room number of each room located on the second floor, with average temperature greater than 80 degrees.

In the collection phase, sensors report the sensed data back to the base station after travelling along the routing tree path created by the underlying protocol. Instead of aggregating packets at the base station, TAG aggregates packets in-network whenever possible to reduce the number of transmissions. In order to achieve a reduction of the number of transmissions in this case nodes should wait until they have received all the packets from their children before they begin to transmit. While the data packets are routed up the root of the tree towards the base station, the intermediate nodes keep aggregating the data. One example is that two packets can be aggregated if they are generated by the sensors that are located in the same room. Since TAG uses SQL-like queries this can be easily done, and the layouts of the SQL results have the same attributes if they are generated by the same query. With the in-network aggregation taking place, TAG not only is suitable for energy conservation but also reduces the required bandwidth and number of message transmissions that would all save energy. The in-network aggregation is very efficient as TAG uses a simple SQL-like query interface, which results in the same layout of the data. The properties that TAG possesses make it a suitable aggregation service particularly for low-power, distributed wireless sensor networks.

A third method involves data prediction techniques that consist of building an abstraction of a sensed phenomenon (model describing data evolution). The model can predict the values sensed by sensor nodes within certain error bounds, and reside both at the sensors and at the sink. If the needed accuracy is satisfied, queries issued by users can be evaluated at the sink through the model
without the need for communication to get the exact data from nodes, thus reducing energy consumption. On the other side, explicit communication between sensor nodes and the sink is needed when the model is not accurate enough, i.e. the actual sample has to be retrieved or the model has to be updated. In the meantime, sensor nodes sample data as usual and compare the data obtained against the predication. If the sensed value falls within the range of an application-dependent tolerance, then the model is considered valid. Otherwise, the sensor nodes (source node) must transmit the sampled data or start updating the model at the sink.

A new class of applications using active sensing modules that are power hungry is emerging. Effective data acquisition techniques need to be applied for energy efficiency and better consumption. Sensing by sensor nodes’ was always assumed to consume almost negligible power when compared to the power consumer by the radio module in the node. This perception is not valid with the new technology advances that have managed to integrate active sensing devices as well as actuators, which are all power-hungry devices. Therefore, energy consumed by the sensing modules needs to be considered because of many factors [83]. For example, image and multimedia sensors [84] require high power resources to perform their sampling task. Sensors like acoustic and seismic transducers generally require high rate and high-resolution A/D, which tend to be power-hungry devices. These converters can account for significant power consumption of the sensing module. Another class of sensors is the active sensors such as sonar, radar and laser rangers. For these to acquire information about the observed quantity they need to send out probing signals and that would constitute a quite big portion of the consumer by the sensor. In these cases, reducing communication would not only reduce energy consumption but, most importantly, would require reduction in the number of acquisitions, i.e. data samples. Many techniques are used in this case, such as adaptive sampling techniques, and these techniques exploit similarities and correlation in the data measures which can then decide on reducing the amount of data acquired from the transducer. As an example, data of interest may change slowly with time and in this case temporal correlations may be exploited to reduce the number of acquisitions, knowing that subsequent samples do not differ very much between each other. The hierarchical sampling approach assumes that nodes are equipped with different types of sensors from high to low resolution devices with different power consumption and requirements. In most cases simple sensors are energy efficient, but their resolution is limited. However, advanced and complex sensors can give a more detailed characterization of the sensed data at the expense of higher energy consumption. This is where a compromise and trade-off between accuracy and efficiency need to be made. Accuracy, can be traded off for energy efficiency by using low power sensors to get a coarse-grained information about the sensing field. Then, in the case of an event that needs to be
detected or greater detail phenomena that need to be observed, the high resolution and accurate sensors are activated. A good example is for target tracking, where often magnetometers and low power passive acoustic devices are used. These would normally be sufficient to track an object, however many times they tend to give false positives. In addition, even when the detection is successful, they cannot be accurate enough to identify what type of object is detected. In that case, high resolution acoustic beam forming or image capturing sensors can be activated to help provide greater accuracy. The combination use of these two techniques can significantly reduce the power consumption of the sensing module by first using the less accurate ones to detect possible targets, and once the target is detected, the more accurate sensors can be activated [85].

2.4.3 Network Level – Topology Control Protocols

The concept of topology control is mainly associated with dense wireless sensor networks that would be networks of redundancy. In most cases network deployment is done at random, by dropping a large number of sensor nodes from an airplane. Therefore, it is convenient to deploy more nodes than necessary to cope with possible node failures occurring during or after deployment [56]. Topology control protocols aim at dynamically adapting the network topology, based on the application needs, so as to allow network operation while minimizing the number of active nodes and thus prolonging network life time.

The dynamic switching sets model divides the scattered deployed nodes in the field into multiple DSS sets. The algorithm takes into account the selection of each DSS when active and running an application to maintain coverage of the adjacent area where DSS is in sleeping mode. Hence at any time, the entire sensing field of interest is covered by a number of active running sensors constituting a number of DSS while at the same time an equivalent number are in sleep mode. The coordination of the alternate active/inactive states of the DSS is operated by the base station (sink). The topology control protocols are generally classified into location-driven and connectivity-driven protocols, however, our proposed model combines both topologies such that the end results are multi-layer sets, in which each layer is a replicate of the other.

In [86], a location-driven protocol called geographical adaptive fidelity (GAF) is described that reduces energy consumption while maintaining a constant level of routing fidelity. In this case the sensing area where the nodes are distributed is divided into small virtual grids as shown in Figure 2.12 below.
Each virtual grid is defined such that, for any two adjacent grids A and B, all nodes in A are able to communicate with nodes in B and vice versa. All nodes within the same virtual grid are equivalent for routing, and just one node at a time needs to be active. Therefore, nodes have to coordinate with each other to decide which one can sleep and for how long. Initially a node starts in the discovery state where it exchanges messages with other nodes, and then it enters the active state and periodically re-broadcasts its discovery message. In this case, when a node detects that some other node can handle routing, it can change its state to sleeping from the active or discovery state. Once the node goes to the sleeping state it stays there for an amount of time, after which it goes to the discovery state. In GAF, load balancing is achieved through a periodic re-election of a leader that will remain active to manage routing in the virtual grid. The leader is chosen through a rank-based election algorithm which considers the nodes’ residual energy, thus allowing the network lifetime to increase in proportion to node density. The frequent process of transitioning the node to sleep and back to the discovery state requires the radio module of the node to transit to different power states constantly, and if the sleeping time is small, then, the amount of current drawn and hence power consumed when the radio is awake may defeat the purpose of the power conservation goal in general when added to the power consumed as a result of continuous broadcasting during the discovery phase. GAF is independent of the routing protocol, which can make it useable with any existing solution. However, the structure imposed over the network may lead to underutilization of the radio coverage areas. Since all nodes within a virtual grid must be able to reach any node in an adjacent virtual grid, this would mean nodes are forced to cover less than half the distance allowed by the radio range. Compare that to our centralized algorithm where the distribution of dynamic switching sets and the alternate switching are all handled by the central
base station in a coordinated sequential method resulting in a significant power saving and prolonged network life time.

A connectivity-driven protocol is suggested by SPAN [87] that adapts coordinators in the network. In this case, elected coordinators stay awake continuously, performing tasks of multi-hop routing, while other nodes stay in sleeping mode and periodically check if they are needed to wake up and become coordinator. The criteria span used in selecting a coordinator is as follows: if two neighbors of non-coordinator node cannot reach each other, either directly or via one more coordinator, then that node should become a coordinator. In a case where several nodes discover the lack of coordinator at the same time and they all decide to become coordinator this could create drift in the synchronization process. To avoid such cases, nodes that decide to become a coordinator defer their announcement by a random back-off time delay. The random back-off time delay is generated by the node using a function that takes into account the remaining residual energy and the number of neighbors that can be connected by a potential coordinator node. The idea is to select a coordinator with a higher expected lifetime and at the same time try to minimize their numbers in the network. A node must withdraw from being a coordinator when every pair of its neighbour can communicate directly, or through some other coordinator. The coordinator transition phase might cause loss of connectivity and in order to avoid this scenario from happening, the old coordinator continues its service until the new one is available up and running [56]. The span election algorithm requires knowing neighbor and connectivity information to decide whether a node should become a coordinator or not. Such information is provided by the routing protocol which makes SPAN depend on it and that may require modification in the routing process of the network.

Another protocol presented by [88], called Adaptive Self-Configuring sEnsor Network (ASCENT) topology, a connectivity-driven type that does not depend on the routing protocol. However, a node in this case decides whether to join the network or continue to sleep based on the connectivity information status and the packet loss that is measured by the node. The basic idea of ASCENT is that initially only some nodes are active, while all other ones are passive, listening to packets but not transmitting. When the situation encounters high message loss from sources due to the fact that not enough nodes are active, the sink starts sending help messages asking neighboring nodes that are in the passive state to join the network by becoming active. Although, these passive neighbor nodes keep their radio on listening to all packets transmitted by the active node, they do not cooperate to forward data packets or exchange routing control information. They only collect information about the network status without interfering with other nodes.
However, active nodes forward data and routing messages until their energy is depleted. Active nodes can also send help messages when they find that local data loss reaches an unacceptable level. Once it joins the network, a node signals its presence as an active node via announcement messages and begins monitoring the condition of the network. This process continues until the number of active nodes is sufficient to contribute to decreasing the message loss experienced by the sink to a value below a pre-defined threshold. While the ASCENT protocol’s primary goal is saving energy by initially allowing only a few nodes to be active and remaining once in sleeping mode, the actual tradeoff side significantly affects the accuracy of the data collected. We feel that waiting until the network experience unacceptable levels of packet loss in order for passive nodes to join would poorly affect the overall collected data precision and accuracy.

2.4.4 Energy Dissipation Model

In order to compare energy performance in WSN for different communication protocols, different energy models are utilized. The first order radio model introduced in [89], is a simple model that is commonly used for energy dissipation, as shown in Figure 2.13.

![First order energy dissipation model](image)

**Figure 2.13**: First order energy dissipation model (adapted from [89])

For the system to transmit $k$-bit packet from transmitter to receiver over a distance $d$, it would spend

$$E_{Tx}(k, d) = E_{elec-Tx}(k) + E_{amp-Tx}(k, d) \tag{2.2}$$

$$E_{Tx}(k, d) = E_{elec} \times k + E_{amp} \times k \times d^2 \tag{2.3}$$

Where: $E_{Tx}(k, d)$ is the energy consumed by the transmitter for a packet of length $k$-bit over a distance $d$. $E_{elec-Tx}(k)$, $E_{amp-Tx}(k, d)$ represent consumed energy by the electronic circuitry of the transmitter and the amplifier respectively.
\[ E_{Rx}(k) = E_{elec-Rx}(k) \]  
(2.4)

\[ E_{Rx}(k) = E_{elec} \times k \]  
(2.5)

Where: \( E_{Rx}(k) \) is the energy consumed by the receiver for the \( k \) - bit packet.

The paper in [6] considers the following values: \( E_{elec} = 50\text{nJ}/\text{bit} \) as the energy consumed by the circuitry of both transmitter and receiver. \( E_{amp} = 100\text{Pj}/\text{bit}/\text{m}^2 \) as the amplifier energy consumption and finally the path loss exponent \( \alpha = 2 \) for the free air environment. It is clear that the receiving packet is a major power consumption factor and hence energy-efficient routing is important as it may turn out that many short links can consume more energy than paths with fewer but longer links.

### 2.4.5 Sensing Model of a Sensor Node

In our research proposal we have used the implementation of equation (4.3) where the connectivity transmission coverage of a sensor node is equal to or greater than twice the sensing coverage [52]. In literature the two most commonly known and utilized sensing coverage models are 1) the binary sensing model and 2) the probabilistic sensing model, as given below:

#### 2.4.5.1 Binary Sensing Model

In this model the assumption of a disk with a fixed radius \( r \) is used to represent the sensing coverage of a sensor \( s \), given by \( C(s) \). This model is explained with reference to equation (2.6) where any event occurring at point \( p \) is detected by sensor \( s \) in the two dimensional plane, \((x_p, y_p)\) only if the Euclidean distance between \( s \) and \( p \), \( d(s,p) \) is within the sensing range of \( s \) as follows [90]:

\[ C(s) = \begin{cases} 
1, & \text{if } d(s,p) \leq r \\
0, & \text{otherwise} 
\end{cases} \]  
(2.6)

Equation 2.6 indicates that the sensor \( s \) will detect an event \( e \) with a probability of 1, if \( d(s,p) \) is less than or equal to \( r \), otherwise it will not detect it. This model is a simplified rather than an accurate one given the imprecise sensor directions and the fact that in reality sensor coverage is not a perfect circle. A more realistic model for the sensor coverage is the probabilistic model.
2.4.5.2 Probabilistic Sensing Model

This model is shown in Figure 2.14 and shows that sensors have three defined areas:

1- Inner Area

This area is given by the distance \( r - r_u \) which clearly ensures that an event \( e \) is detected by sensor \( s \) with probability 1. The uncertainty distance is represented by \( r_u \)

2- Exterior Area

This area has a detection probability of zero. The sensing event \( e \) cannot be detected as it is outside the distance range \( d(s, p) \) and hence being greater than \( r + r_u \)

3- Uncertainty Area

In this region the sensor will detect an event \( e \) with a probability that decays exponentially as the distance increases. This is shown in equation (2.7) below:

\[
C(s) = \begin{cases} 
1, & \text{if } r - r_u \geq d(s, p) \\
\ e^{\lambda \alpha \beta}, & \text{if } r - r_u < d(s, p) \leq r + r_u \\
0, & \text{if } r + r_u < d(s, p) 
\end{cases} 
\]

Figure 2.14: The probabilistic sensing model (adapted from [89])

Where: \( \alpha = d(s, p) - (r - r_u) \) and \( \beta \) and \( \alpha \) are parameters that yield different detection probabilities which can also help to model different types of physical sensors particularly range sensing devices such as infrared and ultrasonic sensors.
2.5 Wireless Propagation Models

When signals are transmitted via a transmitter module with a certain power in a wireless communication, they propagate through a radio channel until they get to the other end and received by the receiver module. During their travel signals suffer attenuation (the power density of the signal decreases) in the radio channel that degrades the initial power they were sent at and the receiver at the other end can only receive the transmitted signals if they arrive with power levels greater than the sensitivity of its transceiver. By definition, the receiver sensitivity is the lowest power level at which the receiver can detect a Radio Frequency (RF) signal and demodulate data, receiver sensitivity is purely receiver specification. The attenuation affecting the travelled signal is commonly known as the path loss of the channel and has a direct dependency on the distance travelled between sender and receiver, the frequency of operation and other factors related to the medium of travel. Path loss models exist to predict if there is a radio channel between two nodes. The three most commonly used path models in the literature [91] are discussed below:

2.5.1 The Free Space Propagation Model

The Friis’ free space propagation model applies when there is a direct and unobstructed path between sender and receiver, i.e. there is line-of-sight. The received power at a distance \( d \geq d_0 \) meters between sender and receiver is given by Friis’ path loss equation below:

\[
P_{rx}(d) = \frac{P_{tx} \times G_{tx} \times G_{rx} \times \lambda^2}{(4\pi)^2 \times d^2 \times L} = C_f \times \frac{P_{tx}}{d^2}
\]

(2.8)

Where \( P_{rx}(d) \) is the power received at the receiver over distance \( d \), \( P_{tx} \) is the power at which the signal is transmitted, \( G_{tx} \) and \( G_{rx} \) are the gain of the antennae of the transmitter and receiver, respectively, \( L \geq 1 \) represents the losses in the circuits of the transmitter and receiver, \( \lambda \) is the wavelength in meters, and \( C_f \) is the constant that depends on the transceivers.

Equation 2.8 shows the signal attenuates in proportion to the square of distance (\( d \)) travelled. When looking at the above equation and from the transmitter’s side, it can be translated that a disk of radius below is created at the center of the node equivalent to its area of coverage.

\[
r = \sqrt{C_f \times P_{tx}}
\]
2.5.2 The Two-Ray Ground Model

In this model the receiver takes into account both the signals traveled from the sender as well as other signals that reach the receiver as shown in Figure 2.15. This of course gives more accuracy than the free-space model described in the previous section. The power received at the receiver over distance $d$ is now given by the following equation:

$$P_{rx}(d) = \frac{P_{tx} \times G_{tx} \times G_{rx} \times h_{tx}^2 \times h_{rx}^2}{d^4} = C_t \times \frac{P_{tx}}{d^4} \quad (2.9)$$

Where: $h_{tx}, h_{rx}$ represent the heights of the antenna of the transmitter and receiver respectively. $G_{tx}, G_{rx}$ represent the gain of the antennae of the transmitter and receiver, respectively. $P_{tx}$ is the power at which the signal is transmitted, $C_t$ is a constant that depends on the transceivers. Equation 2.9 reflects that the signal in this case is attenuated to the fourth power of the distance. Here again we have a transmitter with a disk of radius given in equation below equivalent to the coverage area.

$$r = \sqrt[4]{C_t \times P_{tx}}$$

![Figure 2.15: The two-ray propagation model (adapted from [89])](image)
2.5.3 The Log-Distance Path Model

This model is used as an approach for environments that are uncommon or places where abundant obstacles or reflecting materials are present. It was created from real field measurements that collected data and curve fitting to the data was applied. The log-distance model is given by the following equation:

\[ P_{rx}(d) \propto \frac{P_{tx}}{d^\alpha} \]  \hspace{1cm} (2.10)

In equation 2.10 we see that the path loss is proportional to the transmission power and to the distance between sender and receiver raised to the path loss exponent \( \alpha \) that depends on the environment. The transmitter in this case has a disk of radius \( r \) below equivalent to its area of coverage:

\[ r = \frac{a}{\sqrt{P_{tx}}} \]

Equation 2.10 can also be expressed in decibels as follows:

\[ PL_{r,\text{dB}}(d) = PL_{\text{dB}}(d) + 10\alpha \log \left( \frac{d}{d_0} \right) + X_{\alpha,\text{dB}} \]  \hspace{1cm} (2.11)

Where \( PL_{r,\text{dB}}(d) \) is the received power in dB, \( PL_{\text{dB}}(d) \) is the path loss in dB from sender to receiver over a distance \( d \), \( d_0 \) is a reference distance, \( \alpha \) is the path loss exponent, and \( X_{\alpha,\text{dB}} \) is a zero-mean Gaussian random variable in dB with standard deviation \( \sigma \).

From the above equations, the received power is directly proportional to the distance travelled \( d \) according to the power law and decreases with the frequency of the signal.
2.6 Hardware and Software Platforms for WSNs

2.6.1 Sensor Technology

As was previously introduced, a typical wireless sensor network consists of spatially distributed sensors that work together to monitor some physical phenomena and achieve the required goals. With different available industry technology in the market nowadays, different networks could be formed using different types of platform-based sensor nodes offering unique differentiators such as sensor size, power consumption, nature of operating systems or basic sensing abilities etc. [7].

A sensor node (mote) is a device that measures some physical phenomenon or quantity such as temperature, humidity, sound, light intensity, or motion and converts it into some quantifiable form that can then be read and interpreted by devices or human observers. Along with the main CPU board, each sensor’s mote contains an onboard power, communication, sensing, and processing module that allows it to perform the sensing task. There are many available sensor platforms in the market, and we will list and briefly touch base on the most commonly used ones and their features that make them unique.

2.6.1.1 Mica Platforms

The Mica family is considered one of the first platforms founded for sensor networks and most commonly used sensor platforms. It is supported by a number of operating systems and sensing modules such as TinyOS, Contiki, and Mantis OS. This family includes MicaZ, Mica2, and Mica2Dot series of sensors as shown in Figure 2.16, below. Each of these motes has unique and functional capabilities; Table 2.1 shows the similarities and differences of these devices. These motes are also provided with a special expansion connector in the case of Mica motes and a general purpose interface in the case of Mica2Dot motes which extends the functional capability of these motes by enabling them to interface to some additional boards such as sensor data acquisition boards. Commonly used data acquisition boards are the MTS300 and MTS310 as shown in Figure 2.17 and Figure 2.18 respectively.
Figure 2.16: (a) MicaZ sensor mote (b) Mica2 sensor board (c) Mica2Dot sensor (From Mimsic: Powerful Sensing Solutions. [http://www.memsic.com/wireless-sensor-networks/](http://www.memsic.com/wireless-sensor-networks/)). February, 2014.
Table 2.2: Mica Family of Sensors (adapted from [7])

<table>
<thead>
<tr>
<th>Property</th>
<th>MicaZ</th>
<th>Mica2</th>
<th>Mica2Dot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash memory, kB</td>
<td>128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Measurement memory, kB</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>EEPROM, kB</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>A/D channels</td>
<td>10 bits (8)</td>
<td>10 bits (8)</td>
<td>10 bits (8)</td>
</tr>
<tr>
<td>Frequency, MHz</td>
<td>1400–2483.5</td>
<td>433/868/916</td>
<td>433/868/916</td>
</tr>
<tr>
<td>Data rate, kbps</td>
<td>250</td>
<td>19.2</td>
<td>19.2</td>
</tr>
<tr>
<td>Outdoor range, m</td>
<td>100</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Size</td>
<td>6×3×1 cm</td>
<td>6×3×1 cm</td>
<td>2.5×0.6 cm</td>
</tr>
</tbody>
</table>

Figure 2.17: MTS300 data acquisition board (Source: [7])
2.6.1.2 Telos Platforms

This family is a new generation of motes when compared to the Mica family as they are equipped with a universal serial bus (USB) interface for programming and data collection. There are two main types of Telos sensors, TelosA and TelosB motes [92], as shown in Figure 2.19 below, (Table 2.3 gives their specifications). Similar to the Mica family, the Telos family can be equipped with additional sensor boards such as data acquisition board to extend their functional capabilities.

Figure 2.18: MTS 310 data acquisition board (Source [7])

Table 2.3: Telos Platform and specification (adapted and modified from [7])

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Telos Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program flash memory, kB</td>
<td>48</td>
</tr>
<tr>
<td>RAM, kB</td>
<td>10</td>
</tr>
<tr>
<td>ROM, kB</td>
<td>16</td>
</tr>
<tr>
<td>A/D converter, bits</td>
<td>12</td>
</tr>
<tr>
<td>Frequency band, MHz</td>
<td>2400–2483.5</td>
</tr>
<tr>
<td>Data transmit rate, kbps</td>
<td>250</td>
</tr>
<tr>
<td>Outdoor range, m</td>
<td>75–100</td>
</tr>
<tr>
<td>Size, mm</td>
<td>65×31×6</td>
</tr>
<tr>
<td>Light-sensing range, nm</td>
<td>320–720</td>
</tr>
<tr>
<td>Temperature range, °C</td>
<td>−40–123.8</td>
</tr>
<tr>
<td>Humidity range, % RH°</td>
<td>0–100</td>
</tr>
</tbody>
</table>

2.6.1.3 Imote2 Platforms

This is a powerful sensor platform with a high computation capability as it is built around an Intel PXA271 Xscale processor that ranges from 13 to 416MHz with a built in 2.4GHz antenna. They also come equipped with sufficient memory of around 32MB and a high data rate of less than 250Kbps. All these features make them suitable for tasks such as digital image processing. Figure 2.20 shows a typical Imote2 sensor and Table 2.4 their specifications.

Figure 2.20: Imote2 Sensor (Source: [7])
Table 2.4: Imote2 Specification (adapted from [7])

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDRAM memory</td>
<td>32 MB</td>
</tr>
<tr>
<td>Flash memory</td>
<td>32 MB</td>
</tr>
<tr>
<td>Frequency band</td>
<td>2400–2483.5 MHz</td>
</tr>
<tr>
<td>Data rate</td>
<td>250 kbps</td>
</tr>
<tr>
<td>Range of sight</td>
<td>30 m</td>
</tr>
<tr>
<td>I/O ports</td>
<td>3 UART, 2 SPI, SDIO, GPIO</td>
</tr>
<tr>
<td>Size</td>
<td>36 × 48 × 9 mm</td>
</tr>
<tr>
<td>Operating systems</td>
<td>TinyOS, Microsoft .NET Framework</td>
</tr>
</tbody>
</table>

2.6.2 Software Technology

Intensive study and research has been carried out on the challenges posed by WSNs to the software that run them. The challenges and difficulties of software development for sensor networks are not only due to their inherently distributed nature but also to the need for mechanisms to address harsh operating conditions such as extremely constrained resources of energy, unreliable communications as well as nodes’ vulnerability to faults, especially when deployed on a harsh terrain field. As application software and programming of sensor nodes are also related to our research work, we herewith introduce the most significant software-related aspects of WSNs that are currently the object of intensive research. In [93], Sugihara and Gupta summarized the taxonomy of the overall programming models of WSN as shown in Figure 2.21.

![Figure 2.21: Taxonomy of programming models for WSNs (adapted from [19])](image-url)
Generally, WSN programming models can be classified into three categories namely, Node-level, Group-level and Network-level.

In a Node-level programming model, the focus is on hardware abstraction and allowing access and node control to local sensing interfaces, maintaining application level state in the local memory as well as sending and responding to messages to and from other nodes. One of the earliest examples of this class is the TinyOS [15] operating system with the nesC [16] programming language, which has been the standard software platform for sensor network programming. TinyOS is defined as a lightweight operating system specifically designed for low power wireless sensors. It features in simplifying the building of sensor network applications by providing a set of important services and abstractions such as timers, sensing, storage and communication. Another important feature is its concurrent execution model where developers can build applications of re-usable components and services without the need to be fully aware of unforeseen interactions [15]. At a high level, TinyOS provides three important features that make supplication and system software development easier, these are:

- A component model, which defines how you write small, reusable pieces of code and compose them into larger abstractions,
- A concurrent execution model, which defines how components interleave their computations as well as how interrupt and non-interrupt codes interact,
- Application Programming Interfaces (APIs), services, component libraries and an overall component structure that simplifies writing new applications and services.

The TinyOS itself, applications and systems are all written in the nesC programming language which is a C dialect. It features reducing RAM and code size, enables significant optimization and works around low-level bugs such as race condition.

As can be seen from Figure 2.21, virtual machines in this class provide platform-independent execution models on which the developers can write their programs. On a different objective, virtual machines can also be run on each node that provides an execution environment for scripts that are much smaller than binary codes for TinyOS. Virtual machines as well as middleware are particularly focused for the re-programmability of sensor nodes after being deployed via a wireless channel. This class is sometimes referred to as node-centric programming despite that fact that it allows manual cross-layer optimization which leads to efficient implementation, the required expertise and efforts makes this approach insufficient for developing sophisticated application
behaviors for large scale sensor networks [94]. So we can view the approach of node-centric as the decomposition of the desired global application behavior by the programmer into subsequent codes programmed into individual nodes. This node only, as previously mentioned, is time and effort consuming but also prone to error for complex applications. Adding to the code complexity is the strong coupling of application level logic and system level services such as resource management, routing, localization etc.

The second category of programming and software model is Group-level, in which multiple nodes are grouped as a set on which some set of operations are handled. This provides a language construct that allows developers to program the behavior of the group as a single entity. In this respect, the grouping is established based on either neighborhood relations or some logical node commonality properties. In the neighborhood grouping, the physical closeness defines the belonging of a node in a group, whereas the logical group is based on nodes sharing some logical properties such as node type and sensor reading. This class of group level abstraction facilitates the collaboration among nodes that are particularly useful in implementing “localized algorithms”, [93].

The third category group is the Network-level programming model, which is sometimes referred to as macroprogramming. In this group, the entire sensor network is considered as a single abstract machine where every node and datum can be accessed without considering low-level communications among nodes, this is of course looked at from the macroscopic viewpoint and hence the term ‘macroprogramming’. One interpretation of macroprogramming for WSN is to write one centralized program for the entire network, and the compiler and run time system are responsible for the translation of this program into node level behaviors with the implementation of data coherence respectively. The second approach to Network-level programming is database abstraction where, for instance, the programmer who initiates an SQL-like declarative aggregate query over sensor data is not responsible for the details of in-network processing that are responsible for data collection and processing [94].

2.7 Summary

Sensor networks were first designed to run one specific application when deployed and hence their “usefulness” is limited to that specific application. With the new advances in technology, sensor nodes have become cheaper in price and smaller in size making them ideal for large numbers of deployment. Hence, many applications have started to be deployed on dense networks leading to node redundancy. The availability of these redundant nodes coupled with
the increasing demand for better WSN utilization, have led much research work to be directed towards new techniques to effectively manage dense networks for running multiple applications and hence better returns on investment. Many different techniques and algorithms have been suggested and presented in the literature review, however, most of them assume the availability of luxury-rich resources in order for the suggested techniques to be functional/operation and do not take into account the actual real source constraints that these sensor node possess. In our paper [35], we propose a practical solution called dynamic switching sets (DSS), not only to implement multiple applications running but also to solve the problem of network congestion (after a large number of nodes are deployed) and enhance energy consumption for single application running to achieve prolonging network life time.

We also investigated an important issue faced by most sensor networks: power management and network life time. In our proposed work, our prime energy awareness adopted is controlling nodes operation during network life time and managing them appropriately between active and energy saving sleep nodes. This would only be effectively achieved when considering large dense networks to make full utilization of the excess nodes that would otherwise serve as redundant. Power management and energy conservation are important issues in order to reach the lifetime goal of the network. This issue has become even more critical for networks deployed to regions (fields of interest) where in many cases it is impossible for humans to reach or where it would be impractical to carry out any power replenishment mechanism. Intensive research has been investigated on the effective design of node internal architecture pertaining to energy consumption. Particular attention was paid to the radio module, regarded as the most power-hungry module, and suggested methods to turn it off and hence put the node into an energy saving mode have been explored. This is our prime goal for improving average node energy consumption in our design model. We adopted a multiple-layer approach at network design level, utilizing the availability of large numbers of nodes to get the network operation in multiple sets of nodes called dynamic switching sets. In that case each set would run the intended application needed while all other remaining sets go into sleep mode for power conservation. Also, massive research work has been conducted on topics related to MAC and Routing protocols and ways of re-designing them for effectively improving energy consumption. We believe the wireless channel through which signals propagate is an important topic to explore as it has a major impact on overall network communication and packet delivery as well as energy for the attenuated signals. Hence we introduced common models of the wireless channel used and ones that we adopted for our simulation. Also we touched upon sensing models, particularly the common disk range model that we also used in our simulation environment.
We conclude the chapter with the most widely available and used hardware and sensor platforms in the field in wireless sensor networks and their specifications. Mica and Telos families have led the industry of sensor nodes which were originally manufactured by Crossbow before it was acquired by Mimsic. TinyOS an event-driven operating system has been the de facto operating system for sensor nodes and is based on the programming nesC language.
Chapter 3

3 Network Topology Structure with Dynamic Switching Sets Running Multi-Applications in WSN

In this chapter, we investigate the flat structure and hierarchal structure topologies of a sensor network and their connectivity establishment. The topology of a dense wireless sensor network needs to be effectively managed to utilize the redundant available nodes, not only for energy enhancement but also for enabling running multiple applications such as we propose in our research work. We devise a three-phase cycle for our proposed model through which the topology of each set of nodes, called dynamic switching sets (DSS), is shaped to enable running three different applications. In the first phase, the network initialization process takes place, in the second phase dynamic switching set formation for the entire networks are established. The third phase is the execution of the applications as they start running on different multiple dynamic switching sets. In the proposed work, the topology of the network changes dynamically with the launch of each new application and once the application begins executing, these active nodes ultimately define the topology of the network. The selection of nodes in those DSS are such that when they are active and running a particular application, their collective communication constitutes a topology that ensures coverage and connectivity of all the adjacent sleeping sets and thus the overall network. We conclude the chapter with clustering in wireless sensor networks and cluster heads formation for each cluster. We particularly focus on the LEACH protocol [6] being widely used, however we do not assume any restrictions to any clustering algorithm to be used. Clustering is an important part of the topology in which energy conservation is achieved through in-network processing and data aggregation.

3.1 General Network Topology

One typical listed characteristic of wireless sensor networks is the possibility to deploy a large number of sensor nodes into the region of interest. This ensures sufficient coverage of an area or to have redundancy present in the network where it can be utilized to protect against node failure etc. In this densely deployed wireless network, density control is another important approach to utilize redundant nodes to perform power conservation and management. The density of the nodes
indicates the degree of coverage of an area of interest by sensor nodes; this can be given by \( \mu \) and calculated as in [95]:

\[
\mu(R) = \frac{(N \pi R^2)}{A}
\] (3.1)

Where \( N \) are the scattered sensor nodes in region \( A \), and \( R \) is the radio transmissions range. Generally, \( \mu(R) \) gives the number of nodes within the transmission radius of each node in region \( A \). As well as the aforementioned advantages, density as measured for instance by the average number of neighbors that a single node has, there are also disadvantages. When we have a relatively crowded and large network as shown in Figure 3.1, many typical wireless networking problems arise as a result of the large number of neighbours: many nodes interfere with each other, there are many possible routes, nodes might needlessly consume much of their precious energy on transmission power to talk to distant nodes directly and as a consequence this would affect the reuse of the wireless bandwidth.

Using topology control techniques can solve some of the problems of a dense network. One technique is rather than using the maximum possible connectivity extent of a network, a deliberate choice is made to restrict the topology such that we can have the topology of the network determined by a subset of active nodes and the set of active links along which direct communication can occur. A topology-control algorithm takes a graph \( G = (V, E) \) representing the network, where \( V \) is the set of all nodes in the network and there is an edge \((v_1, v_2) \in E \subseteq V^2\) if and only if nodes \( v_1 \) and \( v_2 \) can directly communicate with each other, and transforms it to a graph \( T = (V_T, E_T) \) such that \( V_T \subseteq V \) and \( E_T \subseteq E \), [4].

![Figure 3.1: Large dense wireless sensor network (adapted and modified from [74])](image)
There are several different topologies that can be applied to configure a communication network as described below [74].

### 3.1.1 Single Hop Star

The simplest WSN topology is the single-hop star as shown in Figure 3.2. Every node in this topology communicates its reading data and measurement directly to the sink (base station). Although whenever feasible this approach shows simplicity in the design, it has a limitation. The limitation of this topology lies in its poor scalability and robustness properties when it comes to larger areas, since nodes that are distant from the sink might not be able to connect with weak wireless links.

![Figure 3.2: Star-connected single-hop topology (adapted from [74])](image)

### 3.1.2 Multi-hop Mesh and Grid

For large networks deployed in a large field of interest, the multi-hop routing becomes necessary. The sensor nodes could either be deployed randomly and uniformly or in a deterministic way where they are placed manually. Figure 3.3 (a) below shows an arbitrary mesh graph for the stochastic deployment while Figure 3.3 (b) shows a more structured communication graph such as the 2D grid structure in case of deterministic deployment.
3.1.3 Two-tier Hierarchical Cluster

In this topology hierarchical architecture is implemented where groups of nodes are selected to form clusters. In each cluster of nodes there is a cluster head that can be of different capabilities in terms of power storage, computation, transmission and memory storage. The advantage of the hierarchical cluster-based approach is that it naturally decomposes a large network into separate zones within which data processing and aggregation (compression) can be performed locally. Within each cluster sensor, readings and data could either be communicated single-hop or multi-hop to cluster heads and they would then be routed through the second tier network formed by a cluster-head to another cluster-head or a base station. The second tier network may utilize a higher bandwidth raid. In random deployment there may be no designated cluster-heads, as these must be determined by a process of self-election. Their hierarchical cluster is shown in Figure 3.4 below.
Figure 3.4: Two-tier hierarchical cluster topology (adapted from [74])

3.2 Connectivity in Network Topology

The two important properties of connectivity and coverage for random deployment network can be best analyzed using Random Graph Theory, [89]. Many studies in the literature have been carried out on different models of random graph, which is essentially a systematic description of some random experiment that can be used to generate graph instances [74]. These models have the tuning parameter that allows the average density of a constructed random graph to be varied.

A random graph model that more closely represents wireless multi-hop networks is the geometric random graph $G(n, R)$. In this type of graph, $n$ sensor nodes are randomly placed and uniformly distributed in a square area of unit size. There is an edge $(u, v)$ between any pair of nodes $u$ and $v$, if the Euclidean distance between them is less than $R$, where $R$ is the connectivity radius of a node.

Figure 3.5 shows $G(n, R)$ for $n = 40$ at two different $R$ values. When $R$ is small, each node can only connect to the node that is close by and which results in a sparse graph. On the other hand, when the value of $R$ is large it enables nodes to connect to other farther ones and that results in a dense network connectivity.
Figure 3.5: $G(n, R)$ geometric random graph: (a) Small $R$ value results in sparse graph and (b) Large $R$ value results in dense network connectivity (adapted from [74])
3.2.1 Network connectivity in $G(n,R)$

As can be seen in Figure 3.6, the probability of the network connectivity varies along with the varying of the radius parameter $R$ of a geometric random graph. The graph shows how the connectivity probability becomes significantly high when certain critical radius values are reached for a given number of nodes $n$. These transitions become sharper (shifting to lower radii) as the number of nodes increases.

Figure 3.7 shows the probability that the network is connected with respect to the total number of nodes for different values of fixed transmission range in a fixed area for all nodes. In this case a high probability of network connection is achieved when a certain number of nodes is reached, depending on the transmission range. This given analysis is relevant for random sensor nodes deployment as it provides some insight into the minimum density that may be needed to ensure network connectivity [74].

![Figure 3.6: Probability of connectivity for a geometric random graph with respect to transmission radius (adapted from [74])](image)
Figure 3.7: Probability of connectivity for a geometric random graph with respect to the number of nodes in a unit area (adapted from [74])
3.3 Dynamic Switching Sets Based Topology Structure

Density control can not only serve sufficient coverage and better power conservation in a wireless network but can also be designed towards network topology control as in our proposed research work, layering and grouping nodes in multiple sets. These multiple sensor node sets are then applied to execute multiple applications in a sequential coordinated manner; as well, the same network model can be applied in case of running one single application using the multiple-set approach which would in this case enhance the prolonging of network life time. The required number of nodes and hence quantity set for deployment is calculated based on the proposed new sensing range radius definition given in Chapter 4. It can also be seen that the overall network topology design is primarily related to the geometrical architecture that the network ends up with after identifying redundant nodes and grouping them with the other nodes, forming what we call dynamic switching sets (DSS) where different sets go active running one application while other sets stay in sleep mode [35]. The definition of topology control given in [89] is the reorganization and management of node parameters and modes of operation from time to time to modify the topology of the network with the goal of extending its lifetime. Of course this is provided maintaining important characteristics related to network sensing connectivity and coverage. Traditionally, the topology of a network would be controlled by changing the transmission power of the nodes, however in our proposed work we are utilizing the topology control in our work from a new concept of turning nodes on and off in a new design that is primarily tailored for running multiple applications. The design concept of the network topology towards our model transits through a number of phases.

3.3.1 Phase 1, Flat Structure and Initialization Phase

In our work we assume that sensor nodes are airborne and randomly deployed, however, the nodes deposition process is also assumed to have a uniform distribution over a given area. We consider that the initialization phase begins when sensor nodes take the flat network structure (without hierarchy) as shown in Figure 3.8 below. Each node has the same capabilities, being homogenous, and they start to find their neighbors, establish the corresponding links and self-configure as well as pass the data from node to node to reach the sink.
Figure 3.8: Phase 1 – Sensor node deployment and initialization

3.3.2 Phase 2, Topology Construction Phase – Formation of Dynamic Switching Sets

The second phase starts after the network initialization is completed. The sink (base station) begins to receive node coordinate locations in the network and save them in a look-up data base table that will be used in the partition algorithm. The application coordinator software in the sink then calculates the network area and determines the boundary nodes using the algorithm explained later in Chapter 4. Once the area of the network and boundary nodes is calculated, the application coordinator starts the partitioning process using the radial lines, as shown in Figure 3.9 below where the formation of the dynamic switching sets is established.
The application coordinator (planner) utilizes the nodes’ geographical coordinates in the field as input for the second algorithm to identify and allocate each node to its corresponding dynamic switching set. The number of partitions and hence dynamic switching sets are now known, as well as which node belongs to which set, thus taking the application coordinator to the third phase of switching certain sets to active mode to run the first applications and instructing other sets to transit to sleeping mode.

**Figure 3.9**: Phase 2 – Formation of network dynamic switching sets (DSS)
3.3.3 Phase 3, Clustering in a Dynamic Switching Set and Running Applications.

In this phase, the application coordinator (software) in the base station determines which set of DSS to remain alive and begin running the first application and which ones should go to sleep mode to wait for their sequential priority (later time slots) to run second and third applications as shown in Figure 3.10 and Figure 3.11 below.

Figure 3.10: Running first application on a DSS set
Once the active sets are identified for running an application, the route discovery process and packet routing begins. The next step is that the nodes inside each partition of the dynamic switching sets start the clustering process and the election of cluster heads (CHs) using Low Energy Adaptive Clustering Hierarchy (LEACH) protocol [6], or any other protocol for this purpose. Cluster heads then begin their main role of communication between the sink and their node members for receiving and executing instructions as shown in Figures 3.12 and Figure 3.13 respectively. Also cluster heads constantly monitor their energy level threshold as to when they need to hand over the role and rotate around the nodes. Their other roles are related to data aggregation and compression before delivering it to the sink (base station).

**Figure 3.11:** Running second application on a DSS set
Figure 3.12: Inner clustering structuring with cluster heads (CH) communicating with sink of the first DSS set
Figure 3.13: Inner clustering structuring with cluster heads (CH) communicating with sink of the second DSS set

3.4 Clustering in Wireless Sensor Networks

Wireless sensor networks by nature consist of a large number of tiny constrained-resources devices that would normally be spread over a large geographical area called the field of interest (FOI). As previously mentioned, they tend to serve many different applications. One common application is environmental monitoring where they are used to collect and process data such as temperature, humidity, light conditions, images of the environment, seismic conditions and so on. These data are collected to detect certain events and trigger activities, for example when these sensors are distributed over large areas of woodland, their main task is to monitor if a fire breaks out and trigger an alarm. Another example is when they are used in farmland to monitor the soil moisture and activate an irrigation system when a certain soil moisture threshold level is reached [96].

When these sensors are deployed, they have no prior knowledge of the network topology, need to communicate among each other via wireless channel and are constrained by memory storage, energy supply and limited bandwidth. These limitations make the desperate need to seek every possible means to reduce and conservatively use their available resources. With that mentioned, it
is necessary to implement effective procedures and mechanisms to deal with the sensor constraints. The hierarchical organization of sensor nodes, grouping them and assigning those specific tasks into the groups (as implemented by dynamic switching sets) before transferring the information to higher levels is one method and mechanism that deals with the sensor limitations, and is called clustering.

Clustering is particularly useful for applications that require scalability to hundreds or thousands of nodes and it implies the need for load balancing, efficient resource utilization and data aggregation. In many applications, data collection and processing need to be in situ for the hierarchical approach and hence clustering is an efficient method for organizing the sensor nodes in the network.

Every cluster of sensor nodes there is a cluster head (CH), which is a node that directly communicates aggregated data with the sink (base station). By assuming roles within a cluster hierarchy, the nodes in a WSN can control the activities they perform and hence reduce their energy consumption. Nodes in each cluster can have one of two roles, either simple data provider (saving energy) or acting as a gateway (cluster head) between the nodes and the base station. These roles are dedicated through an election. In an election, a decision is made on whether a node is elected for a cluster head is based on several factors such as power level signal, transmission schedules, nodes localization and networking function. Clustering brings a number of important benefits that help to solve some of the sensor’s constraints by reducing the cost of transmitting data to base stations, reducing device power consumption, maximizing the routing process execution, gathering sensed data and allowing scalability.

Graph theory concepts can be used in the techniques to create the clusters and represent the topology of the WSN. Graph theory not only helps to create clusters but also in identifying the cluster head, a sensor network can be represented by a graph $G = (V, E)$, where the vertices $V$ represent the sensors and the set of links $E$ represents the connection between vertices provided they are within the transmission of each other as shown in Figure 3.14 below.
Figure 3.14: Graph representation of a WSN (adapted from [96])

A stationary network comprising wireless sensor nodes is well represented by graph data structures and algorithms. The use of graph algorithms such as Depth First Search (DFS) or Breadth First Search (BFS), coupled with a well-established construction of basic graph structures such as trees, cliques or dominating sets, makes good cluster construction and specifically improves their node-to-node communication.

A WSN cluster is made up of three main elements:

1- Sensor Nodes (SN)
   These are sets of sensors that make up the network that collectively work on one dedicated application or multiple applications running on a network. They detect events, process and transmit data; however their prime constraint lies in their limited power available.

2- Base Station (BS)
   The base station is the gateway between the sensor nodes and the outside world, and is the processing point of all data collected from the sensors in the network. There is normally one base station in a network that is positioned at a fixed location, this base station also
manages and controls the applications running on the sensors as well as any software updates or reprogrammability needed.

3- Cluster Heads (CH)

Cluster Heads are the middle point between the base station (BS) and the sensor nodes (SN) in a network. Their main function is to aggregate the data collected from sensors in the cluster before forwarding them to the base station. It can be viewed that cluster heads act as sink to the nodes and the base station acts as sink to the cluster heads.

As previously mentioned the sensor nodes (SN) and the communication links between them can be represented by an undirected graph $G = (V, E)$, where each vertex $v \in V$ (the set of vertices in the graph) represents a sensor node with a unique ID. An edge $(u, v) \in E$ (the set of edges in the graph) represents a communication link only when the corresponding nodes $u$ and $v$ are within transmission range of each other. The graph is formed by defining the neighborhood of each node as well as the k-neighborhood of the nodes in the network and as follows [96]:

**Definition1. Node’s Neighborhood.** The neighborhood $N(v)$ is the set of nodes (neighbors) that resides within the circular transmission range of the node $v$, that is all the adjacent vertices to $v$. If $v$ is included into the neighborhood, it is called a closed neighbourhood of $v$ and it is represented by $N[v]$.

**Definition2. K-Neighborhood.** The $k$-neighborhood of $v$, $N_k(v)$, is the set of nodes with distance at most $k$ from $v$

$$N_k(v) = \{u | u \in V \land d(u, v) \leq k\} \quad (3.2)$$

After the graph is created, the next step is to define the adjacent nodes in the network along with the communication between them from which it would then be possible to determine any reachable node from a specific node as well as to calculate the corresponding hop distance (this is given in Definition 3) between any source and target nodes. Also it will help us determine the $k_{th}$ power graph of a node (as given by Definition 4), to limit the number of nodes that will be considered among the node’s transmission radius when creating the clusters.

**Definition3. Hop Distance.** The shortest path between two nodes $u$ and $v$ is the path with the minimum number of hops between them. The distance $d(u, v)$ is the number of hops in the shortest path between $u$ and $v$. 
**Definition 4.** $k_{th}$ Power Graph. The $k_{th}$ power graph of $G, G_k = (V, E_k)$, is the graph between the nodes in $V$ and an edge between every pair of nodes $u, v \in V$, such that $d(u, v) \leq k$ in $G$.

On determining the available links between the nodes, the next step required in communication is finding the available routes to send the information in WSN. A common mechanism used in a network for message delivery is based on the spanning tree structure which allows the use of the two algorithms Depth First Search (DFS) and Breadth First Search (BFS) to send messages in linear time in the graph created.

Let $T$ represent the spanning tree of $G_c$, rooted at $y$. The $i_{th}$ level of the tree is the set of nodes with hop distance equal to $i$ from $y$. The depth of the tree, depth $(T)$, is the index of the farthest level in the tree.

**Definition 5. Spanning Tree.** A spanning tree is a connected and undirected graph with no cycles. It has $n$ vertices and with every pair of vertices there is exactly one path connecting them, creating $n - 1$ edges in the tree.

Based on the structured spanning tree, the two algorithms of DFS and BFS can be applied. A simplified description for the BFS is to proceed by layers, marking all neighbor vertices that are one hop away from $y$, and then marking vertices that are one hop away from these neighbors, which are two hops away from $y$, and so on. The resulting tree rooted at $y$ is called the Breadth – First-Search (BFS) of graph $G_c$. BFS may be used to find the connected components of a network as well as the shortest distances in terms of the number of hops between the nodes of a network [97].

**Definition 6. Breadth First Search Tree.** A breadth first search tree $T$ of a graph $G_c$ is a spanning tree of $G_c$ such that for every node of $G_c$, the tree path is a minimum hop path to the root.

In order to find all of the vertices reachable from a source vertex $y$ in a graph, Depth-First-Search (DFS) is used. We have $y$ a starting point vertex, DFS visits all possible vertices as far as it can reach, and when all vertices are visited, it returns to the parent node. A DFS tree of graph $G_c$ can be defined as follows:

**Definition 7. Depth First Search Tree.** A frond edge is an edge that does not belong to a spanning tree. For a rooted spanning tree $T$ of graph $G_c$, let us denote by $S(u)$ all the nodes in the subtree of $u$, and $P(u)$ denote all the vertices that exist between $u$ and the root. A Depth-First-
Search tree of a graph $G_c$ is a spanning tree $T$ of $G_c$ such that for every frond edge $\{u, v\}, v \in S(u) \lor v \in P(u)$ [98].

Cluster Representation

The definition of cluster below is one of the many graph concepts used in the creation of clusters in WSN.

**Definition 8. Cluster.** A cluster is any subset of nodes $C \subseteq V$, $y \in V$ is the cluster head and $G_c = (C, E_c)$ is the cluster graph.

$$E_c = \{(u, v) | u, v \in C \land (u, v) \in E\} \tag{3.3}$$

If $G_c$ is connected, then the cluster is connected, $d_c(u, v)$ is the shortest path inside the cluster and the cluster radius is the maximal distance between $y$ and any other node $v \in C$.

It is also possible to use additional graph concepts like node’s weight to add a parameter for the definition and functionality of the clusters.

**Definition 9. Graph Weight.** The nodes in the network graph can have a positive weight $w_v$. The total weight of a cluster is given by the following:

$$W_{\text{sum}}(C) = \sum_{v \in C} w_v \tag{3.4}$$

As previously mentioned, in WSN, clustering is one of the highly researched topics particularly on the effective techniques that can be implemented for networks where energy conservation is of utmost importance. One of the most widely implemented cluster-based techniques is the Low Energy Adaptive Clustering Hierarchy (LEACH) protocol [6]. LEACH includes the idea of clustering without the use of any powerful nodes for the cluster head role. Energy is evenly distributed among cluster node members by a mechanism that rotates the cluster head position to all members, so this is the first positive implementation of LEACH. Secondly, additional energy is saved as LEACH utilizes a Time Division Medium Access (TDMA) schedule based Medium Access Control (MAC) mechanism for intra-cluster communications. In a TDMA, packet collision is avoided as much as possible and the overhearing and idle listening problems are minimized by allowing nodes to turn on and off at scheduled times. Since the protocol relies on cluster heads directly transmitting aggregated data to the base station there is no inter-cluster communication. When cluster head communicate to the base station, LEACH utilizes Code Division Multiple
Access (CDMA), which allows cluster heads to transmit simultaneously without colliding with each other.

Basically, in LEACH, time is divided in rounds. In each round there is a set-up phase in which cluster heads are selected. An algorithm is executed in this phase and guarantees that each node will become a cluster head at some point during the network life time. By rotating the role of cluster heads on a round by-round basis, an evenly distributed node’s energy consumption is achieved that extends the network life time. The selection mechanism is implemented by a threshold-based random procedure that takes into account the probability of becoming a cluster head as given by the following [89].

\[
T(n) = \begin{cases} 
  p \frac{1}{1 - p \times (r \mod \frac{1}{p})} & \text{if } n \in G \\
  0 & \text{otherwise}
\end{cases}
\] (3.5)

Where \( T(n) \) is the threshold for node \( n \), \( p \) is a network parameter that represents the desired percentage of cluster heads, \( r \) is the current round, and \( G \) is the set of nodes that have not been cluster heads in the last \( \frac{1}{p} \) of rounds. When each node completes its threshold calculation, it draws a random number between 0 and 1 to make the final decision. If the number is less than the calculated \( T(n) \), the node becomes a cluster head. It can be seen that the probability of non-selected nodes increases with each round. Each node will become a cluster head during one of the \( \frac{1}{p} \) rounds as nodes already selected as cluster heads cannot become cluster heads again. The second phase is the steady state phase where normal operation takes place. The LEACH clustering technique is based on the received signal strength from the cluster heads. When a cluster head broadcast and advertisement message, nodes start to measure the signal strength at which they received the message and only then decide to join the cluster with the largest value.

Another important feature of LEACH is that cluster heads perform local data fusion (data aggregation). These mentioned strategies allow clustering with the LEACH protocol to save a considerable amount of energy by aggregating data from several sensors in one frame as well as eliminating the transmission of repeated information.

Although LEACH protocol achieves a number of important strategies like extending network life time and distributing energy consumption evenly among sensors, it has some drawbacks, the first drawback being the computationally difficult and energy consuming tasks that cluster heads must perform, such as managing the TDMA scheduling for nodes in the cluster, aggregating data from sensor nodes and sending them directly to the base station. The second drawback is the
synchronization needs for the TDMA scheme to function and finally the lack of multi-hop capabilities that makes it limited to small field spaces. The issue of scalability has been addressed in a similar protocol HEED [99].

3.5 Summary

Topology structure of a sensor network is an important key concept defining the overall network coverage and connectivity. When some sensor nodes leave the network due to their energy depletion or node failure in a harsh environment, the entire network topology is changed and connectivity must be maintained. The connectivity issue in this case is maintained only when we have a dense network where surplus nodes are available to take the place those left ones. Different available topologies have been compared from the simple single-hop star and the multi-hop mesh to the more advanced hierarchal topology architecture where groups of nodes are selected to form clusters. The hierarchical cluster-based approach topology is best suited for large dense networks, as in our case with the formation of dynamic switching sets. It decomposes a large network into separate zones within which data processing and aggregation (compression) can be performed locally.

Connectivity of the network topology is experienced with two varying different parameters, first with the communication range of the sensor node and secondly with the number of nodes and hence density of the network. We also presented a three-phase cycle for our network model topology, through which the formation of multiple sets called dynamic switching sets (DSS) takes place. The network topology in this case dynamically changes with the launch of each application in the network. Since each application runs on different sets of nodes, node selection for each set is critical and will determine the overall coverage and connectivity of the network. We conclude the chapter with clustering in wireless sensor networks and focus on the LEACH protocol [6] as being widely used in this respect. The cluster heads and clustering within each dynamic switching set add additional power saving to the network, as data aggregation mostly takes place in the cluster and communication to the base station is through cluster heads. The clustering and cluster head election process is also discussed. Although we mainly focus on LEACH protocol [6], we do not assume restriction for the use of any available clustering protocol to be implemented.
Chapter 4

4 Running Multi-sequence applications in WSN – Modeling and Algorithm

This chapter is dedicated to our proposed novel system algorithm and modelling of the wireless sensor network. We begin by following up on the graph theory introduced in previous chapters and how a distributed system such as WSN can be modeled as a graph $G(V, E)$, where $V$ represents vertices (nodes) and $E$ are the edges (communication links). Next, we implement the Directed Acyclic Graph (DAG) concept of the graph theory for our model for running multi sequence applications. We define each application as a collection of tasks that are executed based on predefined precedence constraints. Our network model with the detailed system algorithms is then presented followed by the overall system simulation flowchart.

4.1 Modeling of Wireless Sensor Networks

A WSN system can generally be divided into two types based on the network structure, a non-clustered WSN, which is sometimes called a flat network, clustered WSN which is also called a hierarchical WSN. When a large number of sensor nodes are deployed randomly and left unattended to perform pre-programmed functions to obtain data about the environment, these networks can sometimes be referred as flat networks. In these networks all sensor nodes report their sensor readings independently of each other to the sink. In the hierarchical type of network, a number of sensor nodes is grouped as one cluster led by a cluster head (CH). Sensor nodes in the same cluster communicate their gathered data to the cluster head and the cluster head then sends it to the base station. The cluster head’s main responsibilities are coordination among the sensor nodes in the cluster, aggregation (compression) of their collected data and finally communication of the aggregated data to the sink. The clustering concepts and ideas such as designing energy-efficient cluster head selection, or election of cluster heads, how to speed the clustering process etc. have been studied extensively in many research papers [100-104] and are completely beyond the scope of this thesis. The purpose of employing hierarchical WSN into our work is to use it in our proposed Dynamic Switching Sets (DSS) mechanism as each DSS represents a cluster that elects a cluster head to communicate with the sink.
A distributed system such as a wireless sensor network can be modeled as a graph $G(V, E)$ where $V$ is the set of vertices and $E$ is the set of edges of $G$. The computing nodes of the WSN are represented by the vertices of the graph, and an edge exists between the nodes if there is a communication link between them. Figure 4.1 below shows a general graph representing a distributing system consisting of seven nodes, numbered 1-7. It can be seen that the graph is connected, providing a communication path between any pair of nodes.

![Figure 4.1: A graph representing a distributed system](image)

A graph $G$ consists of two parts, a set $V=V(G)$ whose elements are called vertices and a collection $E=E(G)$ of unordered pairs of distinct vertices called edges and it is written $G(V, E)$. If the vertex set $V$ of a graph $G$ is finite then it is called a finite graph and this is what we consider in our model. For the graph $G=(V, E)$ and $v \in V$, the edge $e=\{v\}$ is called a self-loop. An edge is identified by two vertices, and the edge is said to be incident to the vertices, as an example, edge $e=\{v_1, v_2\}$, sometimes written as $e=v_1v_2$ or $e_{v_1v_2}$, is incident to the vertices $v_1$ and $v_2$.

The overall proposed network model is based on a particular type of graph, called a directed graph, on a two dimensional plane. A directed graph (digraph) $G(V, E)$ consists of a nonempty set of vertices $V$ and a set of directed edges $E$ where each $e \in E$ is associated with an ordered set of vertices $[97]$. An edge $e$ is associated with the ordered pair $(u, v)$ and is described as starting from $u$ and ending at $v$. Figure 4.2, below shows a digraph with $V=\{1,2,3,4\}$ and $E=\{\{1,2\}, \{2,4\}, \{3,2\}, \{3,4\}, \{4,3\}, \{4,1\}\}$.
In a directed graph, sometimes we call $u$ the tail of $e$, $v$ the head of $e$, and $u, v$ the end of $e$. If there is an edge with tail $u$ and head $v$, then we let $(u, v)$ denote such an edge and the edge becomes directed from $u$ to $v$. If an edge $e = (u, v)$ in a directed graph $G$ is such that $u = v$, then this is called a loop. Edges $e, f$ are parallel if they have the same tails and heads. An important relationship in a digraph is defined where a vertex $u$ is called the predecessor of vertex $v$ and the corresponding $v$ is called the successor of $u$, if only if edge $e_{uv} \in E, u, v \in V$. 

**Figure 4.2:** A directed graph
4.2 Modeling of Multi-Sequence Applications Running on WSN

The concept of the previously introduced graph theory and directed graph is used in the general implementation approach of the modeling of wireless sensor networks and specifically for our proposed system design of running multi-sequence applications. Implementing the vertex, edge concept of a direct graph on a WSN, it can be seen that a node mimics a vertex where both computation and sensing activities would take place; also the edge would mimic the communication link in this case. The time spent by a node (vertex) on a task computation/sensing as well as on communication (edge) is a measure for the cost which is incorporated in a graph model by assigning weights to its elements. The non-negative weight $s(v)$ representing the sensing/computation cost is associated with vertex $v \in V$ and the non-negative weight $c(e)$ representing the communication cost is associated with edge $e \in E$. These are the basic foundations for the application schedule running in which one application runs at a time on a single wireless network. The application sequential running mechanism is manifested by direct acyclic graph DAG and as follows: directed acyclic graphs are directed graphs that have no cycles. If an edge $e_{i,j} = (v_i, v_j)$ exits from node $v_i$ to node $v_j$, then $v_i$ is called the parent of $v_j$ and $v_j$ is called the child of $v_i$. If $v_i$ is the parent of $v_j$ and $v_j$ is the parent of $v_k$ then we say that $v_i$ is the ancestor of $v_k$ and $v_k$ is the descendant of $v_i$. DAGs may be used to model many different kinds of information. If we define each of the running applications as a collection of tasks and these groups of tasks are executed in a sequential manner, subject to constraints that certain applications must be performed earlier than others, then this may be represented by DAG. Algorithms for topological ordering may also be used to generate a valid sequence.

The overall design model transits through a sequence of phases as follows:

- The field of interest is divided into a number of sets (e.g. 1, 2, 3 sets) and each set is comprised of a number of DSS (group of nodes) called dynamic switching sets (DSS)
- If there were 30 DSS and three sets, then each set of 10 DSS is assigned to run on application when active.
- Once network DSS are established, the base station runs an algorithm to identify and allocate nodes to each one, based on the geographical coordinate where each node lies. A group of these nodes in a DSS are the ones that run a particular application.

- Node connectivity for the active running application is tested using one of the graph traversals methods in which to verify that all nodes are visited and hence connected. There are two search algorithms that can be implemented, depth first search (DFS) and breadth first search (BFS).

- DAGs would be implemented to schedule a group of tasks representing one application. Each group of nodes (tasks) running an application implemented by DAG to run in sequential order is encoded for precedence relations or dependencies such that application-3 runs only after application-2 is complete and application-2 runs only after application-1 is complete.

We consider a hierarchical WSN system for our proposed model and adopt the cluster-based network structure. Our suggested model consists of three main core functional components: an application coordinator, which is a software component deployed in the base station that organizes the entire dynamic switching sets mechanism as well as serving the gateway between the WSN and other external networks such as the internet, the clusters defined by the selected dynamic switching sets (DSS), and the actual sensor nodes being homogeneous and identical in resources which are also equipped with multi-sensing devices integrated on the sensing module component of the node.

The main responsibilities of the sink (base station) are: 1) maintaining a sequence list of the intended application to run during the network life time, 2) applying constraints and priorities for the sequence order of each application, 3) initiating the start and end of each running application based on the sensing field requirement and liaising these commands and instructions with the respective cluster heads in each DSS, and 4) gathering the aggregated data from cluster heads to communicate them to the external networks via the internet, for example. As previously mentioned, the application coordinator (planner) maintains a list of the application sequence scheduled to run during the entire network life time, and hence when the first application is due to run, an instance of DAG of the first application is executed. The application coordinator then further analyzes the application requirements and decomposes it into tasks ready to run on sensor nodes of the selected dynamic switching sets for that particular application. The dynamic switching set partitions are created by running the partitioning algorithms introduced later in this chapter. The application is run into completion for a specified predetermined time, scheduled by the
application coordinator in the sink. When the first application is complete, the sensor nodes in the DSS notify their cluster heads of their task completion status and the cluster head communicates this information to the base station. The base station then, via the application coordinator, instructs cluster heads to have their node members’ transit to sleep mode in preparation for the second application to run on a new set of DSS. With that achieved, the application coordinator initiates a new instance of DAG pertaining to application-2 and the selected DSS to run it. The nodes scheduled to run the second application are located in different set of DSS and have been in sleeping mode. The sink sends command packets to the CHs in these DSS instructing them to become active and begin executing the second new application on the network. During this time, sensor nodes pertaining to application-1 and application-3 are all in sleeping mode as per scheduled sequence. This is repeated until the last application is executed and complete.

The cluster heads, on the other hand, have the following responsibilities: 1) establish communication between their cluster members (sensor nodes) and the application coordinator in the base station, 2) maintain records of the current running application on their cluster nodes, the current status and geographical location of each of their sensor node members, 3) aggregate (compress) the data gathered from their sensor node members and send their readings to the base station (sink), 4) monitor the remaining energy of each node in the cluster as well as itself to keep an update status in case the predefined threshold is reached where a new election of CH is required, and 5) cluster heads cooperate with the sensor nodes and application coordinator to assign and complete the applications as per the sequence planned for the network.

The execution sequence of applications on a network mentioned above is based on the definition of scheduling in terms of processing a specific task that is generally defined in [105], where scheduling is the process of creating a schedule \( S \) for a task graph \( G \) on a target processing environment consisting of a set of processor \( P \) (sensor nodes) that are within a set of DSS running a particular application. The actual schedule responsibility is to coordinate the execution order (temporal) of each application running on the network as well as the location (spatial) of these dynamic switching sets (DSS) in the network. This is generally described by DAG where the execution sequence order of each application is decided by the precedent constraint of the DAG. The location of which DSS is active at any one time is mainly related to network coverage and connectivity when running a particular application. The scheduling system pertains to a number of important properties that are employed in our proposed system. The first one is a dedicated processing environment which basically is mapped to dedicate one application running at one time: no other tasks or applications are executed on the network while the tasks that constitute the
current running application are executed. This is also applied when running the second then third applications etc. The second property is a dedicated processor, which implies that when a sensor node is executing a task for the current running application, it can only execute that task at that particular time for that particular application and the execution is non-preemptive. The other two important properties are related to concurrent communication, assuming no communication contention for the data exchange in the system as well as assuming that the communication network is fully connected.

In a target environment system, each node can communicate with multiple other nodes via a dedicated identical communication link, which means that, for any given task graph there is more than one possible schedule to be obtained. As well, when we look into the traditional goal of schedule strategy in distributed systems, we find that it is mainly to complete a particular task or application as soon as possible and hence aiming for minimal execution time. However, in the case of our proposed model of multiple dynamic switching sets, the scheduling strategy is primarily concerned about which of the sets are to run first in time and come active for running a particular application. We are proposing running multiple applications on a network in a sequential order, more specifically, three applications. The scheduling strategy in this case maintains two lists; the first list contains the number of applications running on the network and the second list contains the number of nodes and their specific locations in DSS that need to be active for one of the three applications in list one. A list-scheduling heuristics class is a popular class that is used for scheduling strategies; it sorts the nodes of the task graph to be scheduled according to a priority scheme, at the same time respecting precedence constraints of the nodes, which results in a topological order node list.

The general structure of WSN, when deployed, can either be a cluster-base or non-cluster base structure that is sometimes referred to as a flat structure. Depending on the node capabilities used for a specific application deployment, the systems can be classified as either a homogenous WSN or heterogeneous WSN. In homogeneous sensor networks, all nodes have similar capabilities in terms of sensing, processing, memory storage, radio transmission and energy storage, as well as same node physical size. However, for a heterogeneous WSN, the nodes are equipped with different capabilities related to power storage, sensing and communication range, processing and memory storage.
In fact, the heterogeneity can be applied to each of the internal structure parts of the node for example, sensing heterogeneity where multiple sensing devices are integrated on the same sensor node. The type that we focus on in our thesis is homogenous in all aspects except the sensing, where it has different types of sensing devices integrated into the sensing board, and selection of sensing devices is to match the purpose of different applications running.

### 4.2.1 Network Model

We make the following assumptions about the network.

1. The transmission range ($r_c$) for each sensor node is fixed.
2. All sensor nodes have identical energy.
3. Sensor nodes are densely deployed in the monitored region.
4. Accordingly, the total energy consumed by transmitting a data packet along a multi-hop path is proportional to the length of the path.
5. All sensors are randomly and uniformly deployed in a planar field.
6. All the sensors and the sink are static and aware of their locations via a localization technique [106].
7. All deployed sensors have a sufficient amount of program memory proportional to the number of applications to be run in a network life cycle.
8. One and only one application will run at a time in a logical predefined sequence that will be coordinated between the base station and cluster heads for every DSS in a network.

The proposed algorithm for running multi-sequence applications on wireless sensor networks primarily aims to achieve multi-application running sequentially but would also serve as an energy-efficient way of operation when the network is only needed to run a single application, hence prolonging the network life time for this specific purpose. There are three main issues that need to be considered.

The first issue is how to divide the entire sensing field (Field of Interest), shown in Figure 4.3, into equal subareas of equal sets of sensor nodes, which we will call Dynamic Switching Sets (DSS), this is shown in Figure 4.4. These DSS sets will run multiple applications in a predefined priority sequence such that a number of sets running one application, for instance, will be in active mode and all remaining sets that are assigned to different applications will be in sleep mode, waiting for
their priority sequence to activate. Several input parameters are needed by our proposed algorithm, not restricted to the following:

a) The total sensing area or field of interest is (A)

b) Density of nodes of a uniform distribution ($\rho_n$)

c) The number of applications to run on network during network lifetime is given by ($m$).

d) Nodes communication range ($r_c$) in Omni-direction

e) Node sensing range is a unit disk ($r_s$) such that: $r_c \geq 2r_s$

Figure 4.3: Sensing Field for wireless sensor nodes deployment
Since the deployed sensor nodes are randomly distributed on the field of interest region, our algorithm will need to identify and allocate a roughly equal number of sensors nodes to different sets for running different applications. And since there will be a number of sets running one application, and those sets will be active at one time while all other sets for other applications will be transiting to sleeping mode, it is imperative that coverage and connectivity are maintained. Our approach to coverage and connectivity is that we establish the necessary and sufficient conditions for coverage to imply connectivity [52]. From the above mentioned we conclude that the proposed algorithm will first need to determine the number of Dynamic Switching Sets (DSSs) for the intended network.
The next issue is to calculate and identify those sensor nodes that happened to be within each of the determined DSS. This means we need to identify which node is within certain $DSSI_i$ and assign an ID number to reflect where it belongs and also how many sensor nodes there are within each $DSSI_i$.

For $1 \leq i \leq k$, where $k$ represents the number of computed DSS sets for the intended field of interest.

The next step is to deal with the clustering issue in which each group of nodes in a DSS would run through a clustering algorithm and elect their cluster heads. Our algorithm is based on a homogenous sensor network, a network in which all the nodes have identical hardware capabilities [107]. The cluster heads are responsible for collecting data from other nodes in the cluster, performing data aggregation [107, 108], to the remotely located sink. In order to ensure load balancing and uniform energy drainage patterns across the entire network, it is proposed to establish a rotating role of cluster heads on the nodes.

Each cluster head (CH) in each DSS will store all the information related to the IDs of the nodes that reside in a particular $DSSI_i$. This information will be exchanged with the base station. It is now the role of the base station to liaise with all CHs to decide on which one stays active and starts running the first application and which ones go to sleep mode and wait for their turn to run a new scheduled application.

4.2.2 Network Security

When wireless sensor networks are deployed in sensitive applications such as health care, defense and early forest fire detection, it is important that data privacy and security of the network are considered. It is likely that after deployment of the network, sensor nodes are left unattended which causes various security concerns [122]. Along with security is the privacy issue in WSN which normally becomes vulnerable when data aggregation takes place at a hop of the network. Privacy has been widely addressed in literature, in [123] the authors address data privacy and integrity together and present a privacy preserving data aggregation algorithm which also preserves data integrity. In our proposed work of running multiple applications on wireless sensor networks, groups of sensors are activated sequentially to execute one application. When each group is active and running, the security and privacy of the data exchanged between sensors (source location) and sink (destination) over a wireless link are of primary importance. The sensitivity level of the security and privacy of data sent over the network largely depends on the type of applications for which network is being used. For example, applications such as military, homeland security as well as health care require high level of data protection from any outside adversary. Since the security and
privacy are outside the scope of our work we do not make any restrictions about the use of any available algorithm or technique on our deployed network for security implementation. However, the work presented by Zia et al. [124] shows a promising technique where experimental results proved minimum trade-off between security and performance. In their work, a comprehensive security solution against the known attacks in sensor network is suggested. The proposed work has a framework of four components that interact with each other to provide a holistic protection. These components are: a secure triple-key scheme (STKS), secure routing algorithms (SRAs), a secure localization technique (SLT), and a malicious node detection mechanism.

4.3 System Algorithm and Flowchart

1- Let $A$ be a convex area representing the sensing field of interest (FOI), as shown in Figure 4.3 and Figure 4.4 respectively.

2- Let $K$ be the number of subareas partition sets of $A$ as shown in Figure 4.4, where each partition is called Dynamic Switching Set (DSS) such that:

$$DSS_1, DSS_2, DSS_3, \ldots, DSS_K \ for \ a \ given \ K \geq 1$$ \hspace{1cm} (4.1)

3- Assume that the number of sensor nodes 'n' is uniformly distributed in $A$. The sensor nodes available in each DSS will be involved in the following roles:

a- Sensors in each DSS will represent a cluster itself

b- Sensors in each DSS will run one assigned application when active or wait in sleep mode to be called once its time slot arrives.

c- Sensors that are currently active in a particular DSS, by the algorithm ought to maintain coverage for the neighboring DSS area whose nodes are in sleep mode.

4- Since sensors nodes in each DSS form a cluster, $1 \leq i \leq k$ let $p_1, p_2, \ldots, p_k$ be the probabilities of sensor nodes located at $DSS_1, DSS_2, DSS_3, \ldots, DSS_K$ respectively [109].

The multinomial probability distributor can be used to represent the number of sensor nodes in each DSS as follows:

$$P(n_1, n_2, \ldots, n_k) = \frac{n!}{n_1!n_2!\ldots n_k!} p_1^{n_1}p_2^{n_2}\ldots p_k^{n_k}$$ \hspace{1cm} (4.2)

$$n = \sum_{i=1}^{K} n_i: \ is \ the \ number \ of \ sensors \ in \ the \ network$$
If $N_1, N_2, \ldots, N_K$ have a multinomial distribution with parameters $n_i$ and $p_1, p_2, \ldots, p_K$, then the expected number of sensor nodes within each cluster $E(N_i) = np_i$. To obtain an equal number of sensor nodes in each DSS (cluster), we have $p_1 = p_2 = \cdots p_k$ which cannot be achieved unless $DSS_1 = DSS_2 = \cdots DSS_k$ in terms of areas, assuming that the density function of sensor nodes in the sensing area is uniformly distributed.

5- Let each sensor node have a communication range ($r_c$) in Omni-direction

6- Let each sensor node have a sensing range a unit disk ($r_s$) such that:

$$r_c \geq 2r_s \quad (4.3)$$

For our algorithm, a new sensing range parameter $r_s'$ is introduced. This parameter is used for each sensor in our network and is calculated such that an active running sensor in a particular DSS, running a particular application would be able to cover the adjacent neighboring DSS partitions whose nodes at the time are in sleeping mode, as shown in Figure 4.4. This new range would depend on the number of applications designed to run on a network, however, for our simulation scenario, we have implemented up to three applications. The formula for the new sensing range of a node is given below:

$$r_s' = \frac{r_s}{m} \quad (4.4)$$

where: $1 \leq m \leq 3$

Where: $r_s'$ is the new sensing range for each node (see Figure 4.4), $m$ is the number of applications running on the network.

7- Let the perimeter of the sensing field be given by $P_s$ in meters.

Therefore the number of partitions DSS is $K$ and is given by:

$$K = \frac{P_s}{r_s'} \quad (4.5)$$

The value of $K$ returns the number of partitions and hence DSSs need to be formed in order to maintain and guarantee coverage when different applications are running by different DSS. That is when one set of DSS is up and running it will take care of the adjacent areas whose DSS are sleeping.
8 - The area of each DSS can now be found as follows:

\[
DSS_A = \frac{A}{K}
\]  

(4.6)

9 – The number of sensor nodes needed per DSS can be approximately calculated as follows:

\[
\rho_{DSS} = \frac{DSS_A}{\pi r_S^2}
\]  

(4.7)

10 – From equation (4.7) we can calculate approximately the total number of nodes needed to be deployed for our convex sensing field

\[
n_T = \rho_{DSS} \times K
\]  

(4.8)

11- Let the sensor nodes at the boundary of the sensing field be B.

The algorithm’s next step is to divide the sensing field area (FOI) into equal subareas which we call DSS or clusters, by radial lines from the center of the field area; this part of the algorithm requires the location of all sensor nodes at the boundary of the sensing field area as an input and is given by B.

Graham’s scanning algorithm [110] is now applied to find a set of the boundary sensing nodes B for the given convex polygon (P) of the sensing field. In this polygon, each sensor node is either on the boundary or inside the polygon of the sensing field. The area A of P can be calculated using the locations of boundary sensor nodes:

\[
(X_i, Y_i), 1 \leq i \leq P_n
\]  

(4.9)

Where: \( P_n \) is the number of boundary sensor nodes and \((X_j, Y_j)\) is the location of a boundary sensor node.

Assume the location of sensor node \((X_{pn+1}, Y_{pn+1})\) is \((X_1, Y_1)\). The area \( A_P \) (area A of convex polygon P) and the centroid location \((X_0, Y_0)\) of the polygon can be found as follows [111]:

\[
A_P = \frac{1}{2} \sum_{i=1}^{P_n} (X_iY_{i+1} - X_{i+1}Y_i)
\]  

(4.10)

\[
X_0 = \frac{1}{6A_P} \sum_{i=1}^{P_n} (X_i + X_{i+1})(X_iY_{i+1} - X_{i+1}Y_i)
\]  

(4.11)
\[ Y_0 = \frac{1}{6AP} \sum_{i=1}^{Pn} (Y_i + Y_{i+1})(X_iY_{i+1} - X_{i+1}Y_i) \]  

(4.12)

For a given number \( K \) and in our case 3, the sensing field area is divided into equal partitions of sets/ DSS/clusters and the area \( A_{CL} \) of each DSS is given by equation (4.6).

The partition/clustering procedure selects an arbitrary sensor node on the boundary B as the starting point \( P_1 \). It then selects sensor node \( P_2 \) on the boundary in anti-clockwise order. The area bounded by \( P_1, P_2, \) and the centroid \( P_0 \) is calculated using equation (4.10). If the area is greater than the \( A_{CL} \) (area of each DSS), this means the required area must be bounded by \( P_0, P_1 \) and intermediate point (a virtual sensor node) \( P_V \) that lies on the \( P_1P_2 \) line, as shown in Figure 4.5 (a). Making use of DSS and the locations of \( P_0, P_1 \) and \( P_2 \), the location of \( P_V \) (X and Y coordinates) is calculated as follows:

\[ Y_V = \frac{y_1-y_0}{x_1-x_0} X_0 + Y_0 - \frac{y_1-y_0}{x_1-x_0} X_0 \]  

(4.13)

\[ X_V = \frac{1}{y_0} (X_0 Y_V - 2DSS) \]  

(4.14)

If the calculated partition/cluster is less than previously calculated DSS as shown in Figure 4.5 (b), a new sensor node on the boundary of the sensing field next to \( P_2 \) needs to be added and the area recalculated. Ultimately, the sensor nodes of the boundary of the \( k \) partitions/clusters are stored in BCL. The partition procedure is shown below:

**Input**: \( k \): number of partitions/clusters, \( A_P \): area of Convex polygon, \( B \): all boundary nodes of the convex polygon, \( P_0 \): coordinate of centroid

**Output**: Set of boundary cluster sensor nodes BCL (i.e. sets of two boundary node and centroid for each cluster)

Select an arbitrary sensor node \( P_1 \) from \( B \); \( DSS_A = \frac{A_p}{K} \)

\( A = DSS_A \) store partition/cluster area

**For each** cluster \( i, 1 \leq i \leq k \) do

- Flag = False
- Put \( P_0 \) and \( P_1 \) as the boundary sensor nodes into \( BCL_i \)

**While** not flag do

- Select sensor node \( P_2 \) next to \( P_1 \) in anti-clockwise order
- If \( \text{area}(P_0,P_1,P_2) \geq DSS_A \) then
  - Calculate a virtual sensor node \( P_V \) using equs. 5 and 6
  - Add \( P_V \) as a boundary sensor node into \( BCL_i \)
  - \( P_1 = P_V \);
  - \( A = DSS_A \)
  - Flag = True

105
Else
   \[ A = \text{area} (P_0, P_1, P_2); \]
   Add sensor node P2 into BCL;
   \[ P_1 = P_2; \]
End
End
End

\textbf{Figure 4.5}: An example of clustering procedure/finding area of each dynamic switching set, P0 is the centroid location of sensing field, P1 and P2 are the locations of two boundary sensor nodes, PV belongs to P1P2, and ACL = DSS is the cluster area (adapted from [109])

The next stage is to identify and locate which actual nodes reside in each of the DSS partition. This is found with the following algorithm:

a) Each Dynamic Switching Set (DSS) is a sector that constitutes a certain angle \( \theta_{DSS} \). This angle is bounded by three points, the centroid and the two vertex points that make up the sector, e.g. \( \angle P_2P_0P_1 \) as shown in Figure 4.6
Figure 4.6: The angle $\theta$ shown for a Dynamic Switching Set partition

b) The algorithm first calculates the angle of each particular DSS sector and identifies its three points, for example in a counterclockwise $P_2 P_0 P_1$, as shown in Figure 4.6. It would then start to randomly pick a node within the neighborhood area adjacent to the sector. The selected node becomes the new vertex point ‘$P_z$’ which would repeat with every new node chosen in this area. Each time a new node is selected, an internal counter is incremented that keeps track of roughly the total number of nodes in each DSS, not to exceed the predetermined population of a DSS. The new $P_z$ vertex now makes a new angle with the other two fixed node points for this sector measuring in counterclockwise direction $\angle P_z P_0 P_1$ as shown in Figure 4.7.
c) If the new calculated angle from the new node is measured as less or equal to the original sector angle then this sensor node is either inside or on the boundary of the DSS sector and hence belongs to this sector. It is identified and given an ID that reflects its membership to this DSS.

d) If however the new measured angle is bigger than $\theta_{DSS_i}$ then it will be ignored as it resides outside that specific DSS and belongs to the adjacent sectors.

e) Then the next node is chosen and the same algorithm is applied. The node selection and angle comparison process is repeated continuously until the number of identified nodes
for a particular DSS reaches the estimated pre-calculated number (compared with internal counter) of nodes that need to be residing in each DSS.

f) The next DSS sector runs the same set of steps of the algorithm above until all the sectors in the sensing area identify their actual nodes in their jurisdictions.

g) Now we need to determine the candidates for cluster heads (CHs). We have \( r \) the transmission range for each sensor node; there is a strong relationship between the Euclidean distance \( d \) from the sender (a sensor node) to the receiver (its cluster head) and the number of hops within distance \( d \). Each route from a sensor node to its cluster head must meet:

\[
\text{Number of hops in a shortest path } \geq \frac{d}{r} \quad (4.15)
\]

Most routing protocols use hop counting as one of the route selection criteria. These protocols aim to minimize the number of transmissions required to send a packet along the selected path. In addition, if the network is a dense network, as in our case, then the minimum number of hops in the shortest path approaches \( d/r \).

To minimize the number of hops between a CH and its sensor nodes, the maximum distance between them needs to be minimized. Accordingly, it is assumed that the center of a cluster area is the best location of the cluster head, which balances energy consumption among the sensor nodes in the partition/cluster, as shown in Figure 4.8. Again, using equations (4.10), (4.11) and (4.12), the location of each CH in each partition is calculated. The procedure is as follows:

**Input:** \( k, DSS_A, BCL_A, P_0(X_0, Y_0) \)

**Output:** Set of Cluster Heads CH \((X_{CH}, Y_{CH})\)

**Foreach** cluster \( i, 1 \leq i \leq k \) **Do**
- Calculate partition/cluster centroid point \( CH_i(X_{CH}, Y_{CH}) \) using equations (4.10), (4.11) and (4.12) with \( DSS_A \) and \( BCL_A \) parameters.

**End**
h) This completes the infrastructure design of multiple sets of sensor nodes which we call Dynamic Switching Sets and the base station’s next step is to begin executing the required applications.

i) The base station then starts communication commands to cluster heads in each DSS to inform them which ones should ask their nodes to transit to sleep and which ones should start actively running the first application.

j) Once the first set of DSS begins executing the first application, all nodes in the remaining DSS transfer to sleeping mode on energy conservation mode [34] as shown in Figure 4.9.

k) From our previously stated algorithm and equations, those that are now up and running application-1 will also be covering the sensing areas of the sleeping ones, adjacent to them.

l) They will continue in the active state until the task/application is complete and they are instructed by the base station to go to sleep mode allowing for the next set to come active for the next application of the network cycle.

m) In that case those DSS that were on sleep and in their calculated position to run application-2 would kick off and again these will be able to cover the sensing areas of the others DSS that are in sleep mode, as shown in Figure 4.10.
n) This process would continue as in Figure 4.11 throughout the network life cycle, which provides the ability of the network to run multiple sequenced sets of nodes to run multiple different applications.

**Figure 4.9**: Running multi-sequence applications in WSN. Application-1 is active and running, application-2, and-3 are in sleeping mode.
Figure 4.10: Running multi-sequence applications in WSN. Application-2 is active and running, applications-1, and -3 are in sleeping mode
Figure 4.11: Running multi-sequence applications in WSN. Application-3 is active and running, applications-1, and -2 are in sleeping mode
4.3.1 System Simulation Flowchart

**Inputs:**
1) Pn = Number of boundary nodes of convex polygon
2) (X1, Y1), (X2, Y2), …(Xpn, Ypn) coordinates of boundary nodes
3) r = sensing range for sensor node
4) m = number of applications to run on one network (use 3 in our simulation)
5) Ps = perimeter of convex polygon
6) tsm = total simulation time (ex. 150sec)

**Calculate:**
1) \( A_p \) = polygon convex area – using equation (4.6)
2) \((X0, Y0)\) Base station centroid coordinate – using equations (4.7) & (4.8)
3) \( \bar{r} \) = new sensing range – using equation (4.12)
4) \( k \) = number of partitions/clusters – using equation (4.13)
5) \( DSS_A \) = partition/cluster area – using equation (4.14)
6) \( \rho_{DSS} \) = approximate number of sensor nodes needed in each cluster – using equation (4.15)
7) \( n \) = approximate total number of sensor nodes for the convex polygon

1) There are 3 sets covering the entire sensing field. Each set contains equal number of partitions/clusters, for example: \( K=36 \), then Set1 =12 clusters (\( DSS_A \)), Set2 =12 clusters, Set3 =12 clusters
2) Identify and assign sensor nodes to each cluster (\( DSS_A \)).
3) Identify and elect cluster head (CH) for each partition/cluster (\( DSS_A \)).

**Simulation Starts:**
1) BS (base station/sink) sends beacon (command) to CHs of Set1 initiating the start to actively running and executing data gathering application-1 on this set.
2) BS sends commands to CHs of Set2 & Set3 instructing them to go to sleep mode.
Active Nodes of Set1 sample the sensed data and send (ex. Temperature) to base station via cluster heads

1) BS sends command to CHs in Set1 to transit cluster nodes to sleep status
2) BS sends command to CHs in Set2 to turn on actively and begin executing data gathering application-2
3) Set3 sensor nodes remain and continue in sleep mode

BS tracks time elapsed if 50sec of the 150sec?

Yes

Active nodes of Set2 sample and send sensed data (ex. Humidity) to base station via cluster heads

4) BS sends command to CHs in Set2 to switch nodes to sleep status
5) BS sends command to CHs in Set3 to get them active and begin executing data gathering application-3
6) Set1 sensor nodes remain in sleep mode

BS tracks time elapsed if 50sec of the 150sec?

Yes

No

115
Figure 4.12: Flowchart for the system algorithm simulation
4.4 Summary

Distributed systems are systems consisting of a number of autonomous computing elements connected over a communication network that cooperate to achieve common goals. A wireless sensor network is considered to be a distributed computing platform that meets the basic requirements of a distributed system. These requirements can be summarized as follows: nodes should be autonomous so that they can work independently, the network should be connected, that is any node should have a communication link directly or indirectly to any other node, and there should be a coordination mechanism for the nodes to operate to achieve common goals. An important advantage of a distributed system is resource sharing, where as in a central resource access a communication bottleneck is encountered as well as being a single point of failure. Distributing the resources such as the data base and peripherals over a network overcomes these problems [97].

We present a novel algorithm to be applied on large dense networks deployments to meet two objectives: running multiple applications on a single network, and prolonging network life time when only one application is needed to run over the entire network life time. The algorithm first partitions the sensing field into an equal number of virtual dynamic switching sets (DSS) using radial lines controlled by the base station. A collection of a number of DSSs constitute one major set that runs one application and in our case we define three sets for three different applications. When each DSS is formed, another algorithm is used to allocate sensor nodes to that DSS taking into account node geographical proximity within the DSS. The number of nodes in each DSS needs to be sufficient such that coverage and connectivity are maintained. Once node allocation is complete, clustering and cluster head election takes place within each DSS. Since we defined three sets for our network, this means that three different applications can be executed. Application execution is managed by the software coordinator in the base station based on a predefined sequence of precedence execution. When the first application is up and running on the first set of DSS, the two other sets of DSS are transitioned to energy-saving sleep mode. The process continues until the three applications have completed running on the network. In the event only a single application is required, this means each of the three sets will have the same instance of the application task and hence will dynamically run the single application three multiple times that will balance out average node energy consumption and hence prolong network life time as we will see by the experimental results presented in the next chapter.
Chapter 5

5 Simulation - Experimental Results and Analysis

The information contained in this chapter is devoted to final experimental results obtained from simulating the proposed system model. The overall simulation results are divided into two main parts, one pertaining to the execution of multiple applications on a single network that shows the switching of different dynamic switching sets with the active one executing packet exchange with the base station while the other sets are in energy-saving mode. The second part of the simulation is on the execution of a single application, implementing the proposed algorithm to prolong network life time. The experimental results obtained show significant energy savings with the proposed algorithm. We selected the Castalia simulator based on an OMNeT++ platform as it meets the following criteria. First, it should support the simulation of WSNs: second, it should be open-source: and third, it should be actively supported by its developers. Of the many available, two were considered and investigated, Castalia and TOSSIM (TinyOS Simulator), however the latter did not have direct support for energy consumption. Other important rationales that are Castalia is highly parametric and has a realistic path loss model for wireless channels, none of which the other available simulators have. Also with Castalia, open source software provides code customization flexibility for algorithm implementation using the C++ programming language. We used C++ to write the code program for the application module for our algorithms and to implement different simulation scenarios. The chapter continues on to discuss both Wireless Channel and Energy models which are implemented in our simulation. After setting the general and top level network simulation parameters, we ran a number of simulation scenarios with different parameter configuration and the experimental results obtained were recorded in tables and graph plotted.

5.1 Research work Simulator - OMNeT++ and Castalia

OMNeT++ is an Object-oriented Modular Network Test bed written in C++ and, as a discrete event network simulation frame work, it is designed to simulate distributed systems and can be extended for modeling other systems [112]. OMNeT++ is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations. The OMNeT++ model is a collection of hierarchically nested modules, as shown in Figure 5.1. The top level module is also called the system module or network. This module contains one or more modules, each of which may contain other sub-modules. Modules are distinguished as being either simple or compound. A simple module is associated with a C++ file that supplies the desired behaviors that encapsulate
algorithms. Simple modules form the lowest level of the module hierarchy. Compound modules are aggregates of simple modules and are not directly associated with the C++ that supplies behaviors. Modules communicate with each other via message passing and message exchange. Messages represent frames or packets in a computer network. The local simulation time advances when a module receives messages from another module or from itself. Self-messages are used by a module to schedule events at a later time. The structure and interface of the modules are specified using a network description language (NED). They implement the underlying behaviors of simple modules [7].

Figure 5.1: OMNeT++ Model, simple and compound modules (adapted from [112])

The main features of OMNeT++ are given below:

- OMNeT++ allows the design of modular simulation models, which can be combined and reused.
- The object oriented approach allows the flexible extension of the base classes provided in the simulation kernel (implemented in the Castalia simulator).
- Model components are compiled and linked with a simulation library, and one of the user interface libraries to form an executable program. One user interface library is optimized for command-line and batch-oriented execution, while the other employs a graphical user interface (GUI) that can be used to trace and debug the simulation.
OMNeT++ offers an extensive simulation library that includes support for input/output, statistics, data collection, graphical presentation of simulation data, random number generators, and data structures.

OMNeT++ simulation kernel uses C++, which makes it possible to embed it in large applications.

OMNeT++ models are built with Network Description Language (NED) and omnetpp.ini and do not use scripts, which makes it easier for various simulations to be configured.

The Castalia is a simulator for Wireless Sensor Networks (WSN), Body Area Networks (BAN) and generally networks of low-power embedded devices. It is based on the OMNeT++ platform which is used by researchers and developers to test distributed algorithms and protocols in realistic wireless channel and radio models. It also gives a realistic node behavior when accessing the radio model [113]. Our selection of simulator was based on the following criteria. First, it should support the simulation of WSNs: second should be open-source, and third, it should be actively supported by its developers. Of the many available two were investigated, Castalia and TOSSIM (TinyOS Simulator), however the latter did not have direct support for energy consumption. Castalia is highly parametric and has the provision to be customized using the C++ programming language to implement different simulation scenarios with some code modification or using available templates to create new modules (ex. Application module). Since the Castalia Simulator is based and built on the OMNeT++ framework platform, it follows the same communication structure of OMNeT messages as the means of communication between modules. Packets are also a form of messages created by OMNeT with some extra fields and methods to model packets in communication networks.

In Castalia, both messages and packets are used for different purposes, everything that gets transmitted/received from the radio is a packet. The control messages serve the purpose of non-network communication information exchange. For example, a control message is used in our simulation when a command is sent to the radio to change its state to SLEEP or back to active (TX State). Another example, a command to a radio to change its transmitting power, or its modulation scheme is modeled as a control message or asking a sensing device for a sensor sample and sensing device replying back, so information exchange is happening within a node. In real sensor nodes these control messages are implemented as interrupts, method calls, asserting pins of a chip etc. and have nothing to do with network packets.
The main features of the Castalia simulator are:

- **Advanced Channel Model based on empirically measured data**
  - Model defines a map of path loss and complex model for temporal variation of path loss.
  - Interference is handled as received signal strength

- **Advanced Radio Model based on real radios for low-power communication.**
  - Probability of reception based on signal interference to node ratio (SINR), packet size and modulation type.
  - Multiple Transmit (TX) power levels with individual node variations allowed.
  - Realistic modeling of Received Signal Strength Interference (RSSI) and carrier sensing.
  - States with different power consumption and delays switching between them.

- **Extended Sensing Modeling provisions**
  - Highly flexible physical process model
  - Sensing device noise, bias, and power consumption

- **Availability of MAC and Routing protocols as well as designed for adaptation and expansion.** Figure 5.2 below shows the internal structure of a node module in Castalia.

![Figure 5.2: Node internal structure in Castalia Simulator (adapted from [113])](image)
5.2 Technical Preliminaries

In this section, we introduce the main models used to represent the hierarchical WSN system, the precedence of the three applications running on the network, the wireless channel for the communication between two nodes and the energy models considered in our work. These are as follows.

5.2.1 Network System Model Design

In this work we make the following assumptions, starting by having a stochastic deployment type with a homogeneous (or heterogeneous when cluster heads are used with more resources, regarding processing and energy) hierarchical WSN. In this case, all sensors have identical energy and software and hardware specifications except the base station (BS) with superior power and capabilities. All sensor nodes are static and aware of their locations. The sequence of application execution is managed and coordinated by the base station in a precedence collaborative approach as per the network tasks requirement (e.g. app-1, app-2 and then app-3).

The system is deployed into a two-dimensional region and mainly composed of three types of devices, sensor nodes, cluster heads (CH) and the sink which is known as the base station (BS). Once sensor nodes are identified and allocated for each dynamic switching set (DSS), we do not make any restriction about the algorithm for cluster formation and cluster head election inside each of the DSSs, for example the one presented by Heinzelman et al. in [6] can be used. These cluster heads are to exchange messages and commands between base station and sensors and have knowledge of the geographical position, residual energy and device sensor types used of their sensors in the cluster. These cluster heads’ main task is communication and no computation tasks are performed. Each cluster is formed by a set of static and homogenous wireless sensor nodes with a single omnidirectional antenna forming a multi-hop network. The radio module (transceiver) used is assumed to be half-duplex that is transmission takes place in one direction at a time. Each dynamic switching set represents a cluster and is modeled as undirected graph as mentioned in Chapter 4, where \( G = (V, E) \) and \( V = (v_1, v_2, \ldots, v_n) \) that represent a set of sensor nodes and \( E = (e_1, e_2, e_3, \ldots, e_m) \) represents the communication links for the sensors. All sensor nodes have identical energy, processing time and communication range \( r^c_i \leq l \), (\( l \) : is the maximum communication range a sensor can reach) that is calculated in our algorithm to take into account the adjacent sleeping DSS when a particular application is running.
5.2.2 Application Model for Dynamic Switching Sets

The prime goal of the overall network system in our proposed work is the task of executing three applications over the entire network life time. Hence, when the three applications collaboratively run based on a precedence set of criteria, this will ultimately achieve the goal of running multiple applications. These three applications can be modeled as three tasks as a Directed Acyclic Graph (DAG), as explained in Chapter 4. A task in this case runs on one set of DSS modeled as a DAG $T = (V, E)$ that is comprised of vertices $V$, representing the non-preemptive sensor nodes running task 1 (application-1), and weighted edges $E$, representing the task dependence and the communication load. This leads to the predefined sequence controlled by the base station in which sensor nodes for application-1 execute as precedence to application-2 executed by the second set of DSS and finally application-3 will only begin to execute on the third set of DSS once its precedence criteria in application-2 are completed.

5.2.3 The Simulated Wireless Channel Model

The wireless channel is a difficult medium to model, especially when taking into account mobile nodes, a changing environment and broadband communications [113]. We used the Castalia Simulator [114] whose strength is the channel/radio model that is built on the work done on modelling of the radio and wireless channel based on empirical measured data [115,116]. Castalia is considered the most realistic simulator for WSN, as it makes the necessary provisions to capture various important features of the wireless channel.

One important aspect of the wireless channel modeling is to estimate the average path loss between two nodes. This is one of the simulation scenarios we implemented to get some realistic experimental results taking into account the path loss. Nodes in real deployment can be a couple of meters to hundreds of meters of separation and therefore the lognormal shadowing model [115] has been shown to give accurate estimates for the average path loss. The equation below gives the path loss in (dB) as a function of distance and other parameters.

$$PL(d) = PL(d_0) + 10 \eta \log \left( \frac{d}{d_0} \right) + X_\sigma$$  \hspace{1cm} (5.1)

$PL(d)$ : is the path loss at distance $d$, $PL(d_0)$ is the known path loss at a reference distance $d_0$, $\eta$ is the path loss exponent, and $X_\sigma$ is a Gaussian zero-mean random variable with standard deviation $\sigma$. Those four parameters are defined in Castalia as given below, we used them as part
of the realistic simulation scenario for which the experimental results obtained were compared with the Naïve model simulation results findings.

double PathLossExponent = default (2.4);
double PLd0 = default (55);
double d0 = default (1.0);
double sigma = default (4.0);

The sigma parameter represents the standard deviation of the path loss variation. If it is set to zero, a case which we used in one of our simulation scenarios in the Naïve Model, then it basically allows all nodes at a certain distance from a transmitter to get the exact same signal strength. However, setting sigma to different positive values, as in another case we used in one of our scenarios for performance evaluation, more or less affects the path loss making it a more random/jittery environment replicating a realistic node environment. Some pairs of nodes get a boost in the signal going from one transmitter to another received. In other words, we can see more connectivity in one case and none in other cases. The process of varying a sigma is also known as the randomness of shadowing.

The path loss exponent values are for the medium in which the signal travels, typically for the free air the values are 2.0 and 2.4. The PLd0 and d0 parameters represent the average path loss in dB and the reference distance in meters respectively. These two values of 55dB and 1m are reference default values used in the Castalia Simulator, taken from the work presented in [115]. In our simulation we have calculated used path loss values, the default 55dB value and a new calculated new value of 62 dB for the new transmission communication range. The experimental results given in the next sections were the effect of path loss variations having an impact on packet reception along with the collision of packets. Wireless channel parameters settings can produce two models, the Naïve Model and the Realistic Model.

The Naïve model is defined as a simple model of a disk model and fulfills the need to test and validate the switching mechanism of our dynamic switching sets and the hypothesis of the proposed work in running multiple applications. This model is only related to make the wireless channel simpler, however, keeping all other simulation parameters unchanged. The wireless channel parameters in this model are set as follows:

SN.wirelessChannel.Sigma = 0
SN.wirelessChannel.bidirectionalSigma = 0

This way all nodes at a certain distance from the transmitter would get the exact same signal strength and links are perfectly bidirectional in terms of quality. Also to provide sharp thresholds
for radio reception, which means either perfect reception of a packet or no reception at all, we need to set the radio mode to ideal.

SN.node[*].Communication.Radio.mode = “IDEAL”

Hence, by setting the two sigmas of the wireless channel to 0 and the radio mode to ‘IDEAL’ we can emulate this model which is still prevalent in unit disk communication mode (i.e. transmissions within a certain range from a transmitter are perfectly received, and outside this range not received at all).

We have used two disk range values in our simulations calculated as follows:

a) 28m range based on PLD0 = 55 dB

Re-arranging equation (5.1) we get:

\[
\text{PLD}_0 = (\text{TxPowerUsed}_{dBm} - \max (\text{receiverSensitivity}, \text{noiseFloor+5dBm})) - 10*\text{pathLossExponent} * \log (\text{range})
\]

In our simulation we used: TxPowerUsed_{dB} = -5dB, receiverSensitivity = -95dB (as used by Castalia based on Telos nodes data sheet), noiseFloor = -100 dB (based on Telos motes data sheet used in Castalia), PLD0 = 55 (default value used in Castalia)

\[
55dB = (-5dB - \max (-95dB, (-100dB+5dB))) - 10*2.4*\log(X)
\]
\[
55dB = (90dB - 24 * \log(X))
\]
\[
-90dB + 55dB = -24*\log(X)
\]
\[
-35/-24 = \log (X)
\]
\[
X = \text{range} = 28.73 \text{m}
\]

b) 15m range based on PLD0 = 62 dB

Re-arranging equation (5.1) we get:

\[
\text{PLD}_0 = (\text{TxPowerUsed}_{dBm} - \max (\text{receiverSensitivity}, \text{noiseFloor+5dBm})) - 10*\text{pathLossExponent} * \log (\text{range})
\]

In our simulation we used: TxPowerUsed_{dB} = -5dB, receiverSensitivity = -95dB (as used by Castalia based on Telos nodes data sheet), noiseFloor = -100 dB (based on Telos motes data sheet used in Castalia), PLD0 = 62 as a new value for the path loss that would give shorter range

\[
62dB = (-5dB - \max (-95dB, (-100dB+5dB))) - 10*2.4*\log(X)
\]
\[
62dB = (90dB - 24 * \log(X))
\]
\[
-90dB + 62dB = -24*\log(X)
\]
\[
-28/-24 = \log (X)
\]
\[
X = \text{range} = 14.7m \sim 15m \text{ range}
\]
We also have the bidirectional sigma parameter given by:

\[ \text{SN.wirelessChannel.bidirectionalSigma} = \text{default (1.0)} \]

In [113], the empirical results obtained show that to capture the correlation between two directions of the link could not be accurately achieved using the lognormal shadowing model. If the two directions are treated as independent links, the variance obtained is much larger than the one experienced in reality. This is why the model used was to return an average path loss for both directions of a link by adding and subtracting a separate Gaussian zero-mean random variable with small standard deviation as defaulted (1.0).

The other important aspect related to the wireless channel is the reception and interference of signals and their calculation. The radio module operates on signals provided from the wireless channel. If a signal is received by the radio module without any other signals received at the same time, then its reception depends on the value of the noise floor (a parameter for the RX mode of the radio operation) and value of the received signal strength which are both used to calculate the Signal to Noise Ratio (SNR). The number of bit errors that the packet/signal experiences through propagation in the wireless channel not only depends on the SNR but also on the modulation scheme, the data rate (parameter of the RX mode of radio operation) and the length of the packet/signal. With the bit error calculated and knowing the encoding scheme, it is decided if a packet is correctly received. If interfering signals are added then Signal to Interference Noise Ratio is added to the above mentioned calculations by the Castalia simulator for modeling the interference. The parameter below is used to include the interference/collision model as part of the realistic model simulation for our scenario which is compared to the Naïve model provided also by Castalia.

\[ \text{SN.node[*].Communication.radio.collisionModel} = [0-2] \]

If set to 0 then there are no collisions happening, which we also used in our first scenario for the Naïve model as will be seen later. If set to 1, there is a simplistic model for collisions where if two nodes are concurrently transmitting and the receiver can receive both of their signals, even minimally then there is always a collision at the receiver. The third value is setting it to 2, which we also used in one of our simulations to mimic the real environment and conditions of realistic deployment. In this case, an additive interference model is used where transmissions from other nodes are calculated as interference by linearly adding their effect at the receiver. There are two possibilities for the additive interference a collision occurs or the receiver receives the stronger of the two transmissions.
So in general, the collisionModel parameter defines the collision model used by the radio to compute the impact of various incoming signals on each other’s reception, a process called interference.

### 5.2.4 The Simulated Energy Model

The energy model that Castalia simulator uses is based on three different power consumption values in a sensor node power consumed when the node is in transmission mode, power consumed when in receiving/idle mode, and the base line power that sums up power consumed by microcontroller and associated electronics circuits. In general we have

\[ \text{Power (W)} = \frac{\text{Energy (Joule)}}{\text{Time (second)}} \]

Based on the radio type used which is chipcon CC2420, the three main power consumption values are:

- **BaseLine Node Power:** 6mW
- **RX Mode:** 62mW = 0.062W (Joules/Sec)
- **TX Mode:** There are different transmission levels in dB from which each corresponds to how much energy the radio is spending when transmitting in a power level.

<table>
<thead>
<tr>
<th>Tx_dBm</th>
<th>0</th>
<th>-1</th>
<th>-3</th>
<th>-5</th>
<th>-7</th>
<th>-10</th>
<th>-15</th>
<th>-25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tx_mW</td>
<td>57.42</td>
<td>55.18</td>
<td>50.69</td>
<td>46.2</td>
<td>42.24</td>
<td>36.3</td>
<td>32.67</td>
<td>29.04</td>
</tr>
</tbody>
</table>

### 5.3 Simulation Results

In this section, the switching mechanism of the proposed algorithm of dynamic switching sets (DSS) running multiple applications in a predefined sequence is evaluated in three different simulated environments. In each scenario, different parameter settings and configurations were implemented introducing wireless channel variations, packet collision models, and different packet rates to evaluate their impact on energy savings of the network. In the first simulation, section 5.3.2, detailed analyses were carried out on the experimental results for DSS running three different applications, as well as a detailed analysis of the evaluation performance of energy conservation for a single application when run with and without the proposed DSS model. The experimental results obtained were then compared to the theoretical calculated values, which revealed a close correlation with our experimental results. The parameter configurations used in the first simulation scenario were: three applications running on dynamic switching sets, 100 nodes, simulation time 150 seconds, initial energy of 100J, average path loss of 55dB that gives a disk range of 28m. In the second part of the first simulation, energy performance of the network was evaluated based on the following parameters: one single application ran on three DSS sets, one application ran on
a single network without DSS implementation, three different packet rates for node packet exchange with the base station were experimented with 1pkt/sec, 5pkts/sec, and 10pkts/sec, 100 nodes, 150 seconds of simulation time, path loss of 55dB (disk range 28m). In section 5.3.3, the second scenario was simulated introducing sigma variation on the wireless channel, a different parameter value for the bidirectional link quality between nodes, a different collision model, initial energy was set to 18720J (representing two AA batteries) and the rest of the parameters remained the same, such as three dynamic switching set groups, 100 nodes, 150 seconds of simulation time, path loss of 55dB (disk range 28m). Finally, in section 5.3.4 same parameters of 5.3.3 section were used except the path loss variation of the wireless channel was changed to 62 dB (PLD0 =62dB) that gives a new node disk communication range of 15m from the previous default set value of 28m. As previously mentioned, a discrete-event simulator, the Castalia simulator based on the OMNeT++ frame work platform [112], was used with the code written in object oriented C++ programming language. All the simulations were run on Intel core Duo CPU of 2.0GHz machine and Linux operating system running Ubuntu 12.04. In the aforementioned simulations, we assessed the system performance according to two metrics: schedule length of each of the three applications running on the network as they switch from one application to another and energy consumption when running a single application on the proposed algorithm.

5.3.1 Simulation Set Up

Before introducing our experimental results, we present the main parameters used in our Castalia simulator. These parameters are all part of the configuration file named “omnetpp.ini” from which they can be set to different configuration values as given below:

Simulation Configuration Parameters: (omnetpp.ini)
[Config General]
1. Sim-time-limit = 150s // Total simulation time
2. SN.num Nodes = 100 // This is 99 nodes + base station
4. SN.node[*].Communication.Radio.TxOutputPower =”-5dBm”
5. SN.wirelessChannel.sigma = 0 // For Naïve Model, different values for realistic model
6. SN.wirelessChannel.bidirectionalSigma = 0 // Naïve model
7. SN.node[*].Communication.Radio.mode = “IDEAL” // Naïve model
8. SN.node[*].Communication.Radio.CollisionModel = 0 //Naïve model
9. SN.wirelessChannel.onlyStaticNodes = True
10. SN.node[*].Application.packetRate = 1 (bytes/Sec)
11. SN.node[*].Application.constantDataPayload = 2000 (in bytes)
12. SN.node[0].Application.isSink = True
13. SN.node[*].Application.reportDestination = “SINK”
14. SN.node[*].Communication.RoutingProtocolName = “MultipathRingRouting”
15. `SN.field_X = 30, SN.field_Y = 30`
16. `SN.deployment = "[0]→center; [1—99]→uniform"
17. `SN.node[*].ApplicationName = "ScheduleApp"
18. `SN.node[*].Application.groupCount = 3 //Number of applications running
19. `SN.node[*].Application.timeSlot = 50`
20. `SN.node[0].Application.collectTraceInfo = True
21. `SN.node[*].Application.collectTraceInfo = True
22. `SN.node[*].Communication.MACProtocolName = "TunableMAC"
23. `SN.node[*].ResourceManager.initialEnergy = 100 // another value used of 18720J
24. #New configuration is given below for the simulation of energy performance when running single
   application on the same network without the switching mechanism
   [Config withoutGroup]
25. `SN.node[*].Application.groupCount = 1
26. `SN.node[*].Application.timeSlot = 150

The Castalia simulation program developed for our model/algorithm to run multi-sequence
applications on a WSN is comprised of the following main parts:

1- The .cc file, this is the actual code for the application module developed in C++ and named
   “scheduleApp” for our multi-sequence application program.
2- The .h file is the header file where we declare a new C++ class that implements our module.
   The name of the class has to match the name of our module which will inherit from the
   “VirtualApplication” class which is a base class for any Castalia application. This class
defines the methods to operate within the Castalia framework as well as defining a set of
methods that the specific application can use to interact with the rest of the modules such
as communication (Radio, MAC and Routing), Mobility manager, Resource manager,
Sensor manager, Physical process and wireless channel modules.
3- The .ned file contains a list of mandatory system parameters as well as user-defined
   parameters specific for our new developed module. These parameters are passed to the
module at runtime from the simulation configuration. In this file, it is possible to provide
default values for parameters or else the parameters have to be assigned a value in the
configuration input file (omnetpp.ini). Also, in this file the interfacing gates between
modules are established such that output and input to different modules are connected.
These gates allow the sensor node module to connect to the wireless channel in order to
communicate with other nodes and to connect to physical processes so it can sample them.
4- The omnetpp.ini file is the configuration input file where we can assign new values to
   simulation parameters and for all modules different to the default values set in the .ned
   file. In this file we can create different configurations to test different scenarios. Some
   parameters are given below.
Table 5.1 below shows a list of Castalia output configuration files with the date created. The titles with “General” in the table are those configurations for running three applications for the multi-application execution over a WSN network. The titles ‘withoutGroup’ are configuration files for running a single application without Dynamic Switching Set implementation and are used for evaluating energy consumption and performance when compared with the DSS implementation.

Table 5.1: Castalia output files for all simulations, with and without DSS implementation

<table>
<thead>
<tr>
<th></th>
<th>Configuration</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>131101-061257.txt</td>
<td>withoutGroup (1)</td>
<td>2013-11-01 06:12</td>
</tr>
<tr>
<td>131101-054611.txt</td>
<td>General (1)</td>
<td>2013-11-01 05:46</td>
</tr>
<tr>
<td>131104-214312.txt</td>
<td>[General,withoutGroup] (20)</td>
<td>2013-11-04 21:43</td>
</tr>
<tr>
<td>131104-213448.txt</td>
<td>General (1)</td>
<td>2013-11-04 21:34</td>
</tr>
<tr>
<td>131104-213404.txt</td>
<td>General (1)</td>
<td>2013-11-04 21:34</td>
</tr>
<tr>
<td>131104-194647.txt</td>
<td>General (1)</td>
<td>2013-11-04 19:46</td>
</tr>
<tr>
<td>131104-193803.txt</td>
<td>General (1)</td>
<td>2013-11-04 19:38</td>
</tr>
<tr>
<td>131104-193734.txt</td>
<td>General (1)</td>
<td>2013-11-04 19:37</td>
</tr>
<tr>
<td>140125-184759.txt</td>
<td>General (1)</td>
<td>2014-01-25 18:47</td>
</tr>
<tr>
<td>140217-232939.txt</td>
<td>General (1)</td>
<td>2014-02-17 23:29</td>
</tr>
<tr>
<td>140217-232505.txt</td>
<td>General (1)</td>
<td>2014-02-17 23:25</td>
</tr>
<tr>
<td>140310-234521.txt</td>
<td>General (1)</td>
<td>2014-03-10 23:45</td>
</tr>
<tr>
<td>140310-224610.txt</td>
<td>General (1)</td>
<td>2014-03-10 22:46</td>
</tr>
<tr>
<td>140311-002429.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-11 00:24</td>
</tr>
<tr>
<td>140311-001128.txt</td>
<td>General (1)</td>
<td>2014-03-11 00:11</td>
</tr>
<tr>
<td>140313-005500.txt</td>
<td>General (1)</td>
<td>2014-03-13 00:55</td>
</tr>
<tr>
<td>140316-235755.txt</td>
<td>General (1)</td>
<td>2014-03-16 23:57</td>
</tr>
<tr>
<td>140316-235330.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-16 23:53</td>
</tr>
<tr>
<td>140316-234048.txt</td>
<td>General (1)</td>
<td>2014-03-16 23:40</td>
</tr>
<tr>
<td>140316-233311.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-16 23:33</td>
</tr>
<tr>
<td>140316-231746.txt</td>
<td>General (1)</td>
<td>2014-03-16 23:17</td>
</tr>
<tr>
<td>140316-230848.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-16 23:08</td>
</tr>
<tr>
<td>140316-215928.txt</td>
<td>General (1)</td>
<td>2014-03-16 21:59</td>
</tr>
<tr>
<td>140317-000638.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-17 00:06</td>
</tr>
<tr>
<td>140331-233153.txt</td>
<td>withoutGroup (1)</td>
<td>2014-03-31 23:31</td>
</tr>
<tr>
<td>140331-232751.txt</td>
<td>General (1)</td>
<td>2014-03-31 23:27</td>
</tr>
<tr>
<td>140402-114824.txt</td>
<td>withoutGroup (1)</td>
<td>2014-04-02 11:48</td>
</tr>
<tr>
<td>140402-114447.txt</td>
<td>General (1)</td>
<td>2014-04-02 11:44</td>
</tr>
<tr>
<td>140402-113513.txt</td>
<td>withoutGroup (1)</td>
<td>2014-04-02 11:35</td>
</tr>
<tr>
<td>140402-112832.txt</td>
<td>General (1)</td>
<td>2014-04-02 11:28</td>
</tr>
<tr>
<td>140402-105137.txt</td>
<td>General (1)</td>
<td>2014-04-02 10:51</td>
</tr>
<tr>
<td>140402-104131.txt</td>
<td>General (1)</td>
<td>2014-04-02 10:41</td>
</tr>
<tr>
<td>140402-101949.txt</td>
<td>[General,withoutGroup] (20)</td>
<td>2014-04-02 10:19</td>
</tr>
<tr>
<td>140402-101248.txt</td>
<td>withoutGroup (1)</td>
<td>2014-04-02 10:12</td>
</tr>
<tr>
<td>140402-093821.txt</td>
<td>General (1)</td>
<td>2014-04-02 09:38</td>
</tr>
</tbody>
</table>
A Castalia output file listed in Table 5.1 is used as an input file for the CastaliaResults script that parses the file and produces list of outputs recorded by different modules. These are shown in Table 5.2 below.

**Table 5.2: List of outputs produced by modules comprising the node during simulation**

<table>
<thead>
<tr>
<th>Module</th>
<th>Output</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application level latency, in ms</td>
<td>1x1(11)</td>
</tr>
<tr>
<td></td>
<td>Packets received per group</td>
<td>1x1(3)</td>
</tr>
<tr>
<td></td>
<td>Packets received per node</td>
<td>1x99</td>
</tr>
<tr>
<td></td>
<td>Remaining energy</td>
<td>100x1</td>
</tr>
<tr>
<td>Communication.MAC</td>
<td>TunableMAC packet breakdown</td>
<td>100x1(4)</td>
</tr>
<tr>
<td>Communication Radio</td>
<td>RX pkt breakdown</td>
<td>100x1(3)</td>
</tr>
<tr>
<td>ResourceManager</td>
<td>TXed pkts</td>
<td>99x1</td>
</tr>
<tr>
<td></td>
<td>Consumed Energy</td>
<td>100x1</td>
</tr>
</tbody>
</table>

The dimensions column shown in the table above is NXM, where N is the number of modules that produced the corresponding output and the M refers to different indices as each index number is related to each sender of packets. For example, packets received per node in the output column show the sink that generated this output and that the 99 nodes of the networks are those who sent packets.
5.3.2 First Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s SimulationTime | NodeInitialEnergy 100J | PLD0 55dB

5.3.2.1 Experimental results for running three applications on WSN

In this model, the variability of the wireless channel sigma, communication links, and packet collisions have been removed and set to ideal for the purpose of validating our proposed algorithm and the associated mechanism of switching different sets for application running. Two simulation scenarios were executed in this simulation. One was for testing multiple (three) applications running on a single network and the second was to test energy saving and performance when a single application ran on the network with and without the implementation of the DSS mechanism and to achieve the goal of prolonging network life time. After the sensor allocation process to each DSS was complete, each set ran one application in sequence and began communicating and sending packets to the base station. The experimental results shown in Table 5.3 are collected from three applications executed over 150 seconds of network life time. It can be seen that the total packets sent by each DSS group 1, 2 and 3 respectively are for the 33 sensor nodes that make up each of the three sets. Hence, the average number of data packets communicated by each node in each group is approximately 50 packets which verifies the actual fact of packet rate of 1 packet per second used in this simulation.

Table 5.3: Average number of packets received by the base station from each DSS group and from each sensor node (packet rate of 1 packet/sec)

<table>
<thead>
<tr>
<th>Application: Packets received per group</th>
</tr>
</thead>
<tbody>
<tr>
<td>+----------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+----------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+----------------------------------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Packets received per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>+----------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+----------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+----------------------------------------</td>
</tr>
</tbody>
</table>
Table 5.4: First Simulation, trace sample of time stamp of packets sent to the base station

<table>
<thead>
<tr>
<th>Trace #</th>
<th>Simulation Time</th>
<th>Module Produced Trace</th>
<th>Trace Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.007968396428</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 99</td>
</tr>
<tr>
<td>2</td>
<td>1.014313467657</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 87</td>
</tr>
<tr>
<td>3</td>
<td>1.019315647838</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 96</td>
</tr>
<tr>
<td>4</td>
<td>1.024393333196</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 12</td>
</tr>
<tr>
<td>5</td>
<td>1.028638141700</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 69</td>
</tr>
<tr>
<td>6</td>
<td>1.033676935209</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 30</td>
</tr>
<tr>
<td>7</td>
<td>1.037984559534</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 81</td>
</tr>
<tr>
<td>8</td>
<td>1.042468460669</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 33</td>
</tr>
<tr>
<td>9</td>
<td>1.047026713873</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 18</td>
</tr>
<tr>
<td>10</td>
<td>1.051444472692</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 24</td>
</tr>
<tr>
<td>11</td>
<td>1.055738166722</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 6</td>
</tr>
<tr>
<td>12</td>
<td>1.06017613949</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 21</td>
</tr>
<tr>
<td>13</td>
<td>1.064823126949</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 57</td>
</tr>
<tr>
<td>14</td>
<td>1.069263190413</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 45</td>
</tr>
<tr>
<td>15</td>
<td>1.073742162268</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 90</td>
</tr>
<tr>
<td>16</td>
<td>1.079524802641</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 54</td>
</tr>
<tr>
<td>17</td>
<td>1.084280352023</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 3</td>
</tr>
<tr>
<td>18</td>
<td>1.089168491860</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 66</td>
</tr>
<tr>
<td>19</td>
<td>1.093810894009</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 78</td>
</tr>
<tr>
<td>20</td>
<td>1.098622553503</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 15</td>
</tr>
<tr>
<td>21</td>
<td>1.103465844753</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 9</td>
</tr>
<tr>
<td>22</td>
<td>1.10774625202</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 63</td>
</tr>
<tr>
<td>23</td>
<td>1.112922288172</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 39</td>
</tr>
<tr>
<td>24</td>
<td>1.117249232561</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 75</td>
</tr>
<tr>
<td>25</td>
<td>1.122457663665</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 36</td>
</tr>
<tr>
<td>26</td>
<td>1.127083681369</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 93</td>
</tr>
<tr>
<td>27</td>
<td>1.132985156939</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 42</td>
</tr>
<tr>
<td>28</td>
<td>1.137843162697</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 72</td>
</tr>
<tr>
<td>29</td>
<td>1.143023578494</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 60</td>
</tr>
<tr>
<td>Trace #</td>
<td>Simulation Time</td>
<td>Module Produced Trace</td>
<td>Trace Message</td>
</tr>
<tr>
<td>-------</td>
<td>----------------</td>
<td>-----------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>30</td>
<td>1.147470926823</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 84</td>
</tr>
<tr>
<td>31</td>
<td>1.152173371994</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 48</td>
</tr>
<tr>
<td>32</td>
<td>1.160361720717</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 27</td>
</tr>
<tr>
<td>33</td>
<td>1.166682266784</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 51</td>
</tr>
<tr>
<td>34</td>
<td>2.00796787173</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 99</td>
</tr>
<tr>
<td>35</td>
<td>2.013666861624</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 12</td>
</tr>
<tr>
<td>36</td>
<td>2.01791521384</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 45</td>
</tr>
<tr>
<td>37</td>
<td>2.017968841767</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 96</td>
</tr>
<tr>
<td>38</td>
<td>2.022655117534</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 87</td>
</tr>
<tr>
<td>39</td>
<td>2.028472265434</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 36</td>
</tr>
<tr>
<td>40</td>
<td>2.029664136465</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 6</td>
</tr>
<tr>
<td>41</td>
<td>2.034041551129</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 81</td>
</tr>
<tr>
<td>42</td>
<td>2.038821680154</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 42</td>
</tr>
<tr>
<td>43</td>
<td>2.043759097488</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 15</td>
</tr>
<tr>
<td>44</td>
<td>2.04826492478</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 33</td>
</tr>
<tr>
<td>45</td>
<td>2.052858391252</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 60</td>
</tr>
<tr>
<td>46</td>
<td>2.052904080071</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 75</td>
</tr>
<tr>
<td>47</td>
<td>2.057163957716</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 66</td>
</tr>
<tr>
<td>48</td>
<td>2.062071800515</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 39</td>
</tr>
<tr>
<td>49</td>
<td>2.066662901487</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 93</td>
</tr>
<tr>
<td>50</td>
<td>2.07113297167</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 90</td>
</tr>
<tr>
<td>51</td>
<td>2.075529272459</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 18</td>
</tr>
<tr>
<td>52</td>
<td>2.080066538951</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 72</td>
</tr>
<tr>
<td>53</td>
<td>2.085374008763</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 9</td>
</tr>
<tr>
<td>54</td>
<td>2.09003488418</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 24</td>
</tr>
<tr>
<td>55</td>
<td>2.094893532302</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 27</td>
</tr>
<tr>
<td>56</td>
<td>2.099489369125</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 3</td>
</tr>
<tr>
<td>57</td>
<td>2.103952673388</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 21</td>
</tr>
<tr>
<td>58</td>
<td>2.108948857029</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 51</td>
</tr>
<tr>
<td>59</td>
<td>2.108958865288</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 69</td>
</tr>
<tr>
<td>60</td>
<td>2.114885951043</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 57</td>
</tr>
<tr>
<td>61</td>
<td>2.121292273765</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 54</td>
</tr>
<tr>
<td>62</td>
<td>2.125598885661</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 30</td>
</tr>
</tbody>
</table>
In Table 5.4 we present a sample of data generated by the application module output of the base station. The time trace shows time instants during the 150 seconds simulation interval at which a certain packet number was sent by a node and delivered to the base station. Node IDs are the once active sending packets to the base station as while executing applications 1, 2, and 3. In the first 50 sec simulation interval, 33 of the 99 nodes are actively running application-1 (TX/RX mode) while the remaining 66 nodes in sleep mode with radios in idle/listening state, then the second set of DSS nodes comes active for the next 50 sec time interval of the total simulation time running second application. Meanwhile nodes subscribed to the first and third DSS groups are transitted to sleep mode. Third group of 33 nodes becomes active for the last 50 sec interval of simulation time running application-3. All time instants of these active and sleeping nodes are recorded in Castalia-Trace.txt file, of which 71 time instants sample is shown in Table 5.4.
Table 5.5: Running three applications on wireless sensor network with Dynamic Switching Sets packets exchange

<table>
<thead>
<tr>
<th>SN ID</th>
<th>DSS G01 (#Pkts)</th>
<th>DSS G02 (#Pkts)</th>
<th>DSS G03 (#Pkts)</th>
<th>SN ID</th>
<th>DSS G01 (#Pkts)</th>
<th>DSS G02 (#Pkts)</th>
<th>DSS G03 (#Pkts)</th>
<th>SN ID</th>
<th>DSS G01 (#Pkts)</th>
<th>DSS G02 (#Pkts)</th>
<th>DSS G03 (#Pkts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>67</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>35</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>68</td>
<td>0</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>36</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>69</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>49</td>
<td>0</td>
<td>37</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>70</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>71</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>39</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>72</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>73</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>74</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>75</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>43</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>76</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>44</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>77</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>45</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>78</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>49</td>
<td>0</td>
<td>46</td>
<td>0</td>
<td>49</td>
<td>0</td>
<td>79</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>48</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>81</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>49</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>82</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>83</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>51</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>84</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>85</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>53</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>54</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td>87</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>55</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>88</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>57</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>58</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>91</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>27</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>93</td>
<td>51</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>61</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>94</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>49</td>
<td>95</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>63</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>96</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>64</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>97</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>65</td>
<td>0</td>
<td>49</td>
<td>98</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>66</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>99</td>
<td>50</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Running Multi-Sequence Applications on WSN – Three Applications

Total Network Time Simulation = 150 Seconds

DSS Group1 = (0-50Sec) | DSS Group2 = (51-100Sec) | DSS Group3 = (101 – 150Sec)
The experimental results obtained for running multi-sequence applications from the trace file (Castalia-Trace.txt), of which samples were recorded in Table 5.4, have been compiled and represented in Table 5.5; we ran the simulation on a total of 100 sensor nodes (Sink + 99 nodes) and running three different applications over a 150 second network life time. There are three different groups of dynamic switching sets, named DSS-Group-1; DSS-Group-2 and DSS-Group-3 as shown in Table 5.5, and in each group different node IDs are subscribed and assigned to run one application. In DSS-Group-1 for example, node IDs (3, 6, 9, 12, 15, 18 ...etc.) have been allocated by the base station and tagged for the first application that was executed for the first 50 seconds of the simulation time. The average number of packets exchanged by each node and sent to the base station was 50 packets. During this time interval (based on the packet rate parameter that was set to 1 packet/second), as shown in Figure 5.3, at the same time node IDs (1, 2, 4, 5, 7, 8 etc) belonging to the other two DSS sets have zero packets exchanged since they were transitted to sleep mode. In Figure 5.4, the second set of DSS nodes is active running application-2 (nodes 1, 4, 7, 10, 13, etc.), while nodes of DSS-Groups 1 and 3 are switched to energy saving mode, sleep mode. Finally, Figure 5.5 shows active nodes in DSS of set 3 running the last application while node group DSS 2 and 3 are in sleeping mode.

At the start, all nodes report their location coordinates to the main base station (sink), which is located at the center. The sink begins the sorting process of the deployed nodes on the field into three major layers of dynamic switching sets, each containing 33 nodes, then it begins sending control messages via communication packets to all sensors in the network (99 nodes). Once these commands are received by the radio module in each node, they are passed up to the application module via MAC and routing layers. The application layer then reads the command and compares its destination address. If the destination address matches the current self-node, it sends an internal message to the radio module to put it to sleep, otherwise it stays active and ready to run the intended application and at the same time forwards the packet received to the next available hop node. The process continues until all nodes that belong to the dynamic switching sets of 2 and 3 are entered in sleep mode. The rest of the nodes remain active and constitute the first set of DSS and collaboratively begin running the first application. The base station's next step is to send a configuration packet to the active set selecting the sensing devices needed to be used for the first application.
It should be noted that each sensor node is equipped with an embedded sensing board of built-in sensors such as temperature, humidity, light, acoustics, rain, smoke, etc. from which any can be selected to match the requirement of a particular application. These active nodes then begin to sample their sensors and send readings via packets to the sink at a constant rate. The packet rate configured in the omnetpp.ini file of our simulation was 1 packet/sec and hence we see an average of 50 packets sent by each sensor node to the base station in the course of 50 seconds. The first application was set by the base station to run for 50 seconds during which other nodes in different DSS have been in sleeping mode.

Once the first set of DSS (33 nodes) completes the application-1 task, the base station sends new sets of commands to transit them to sleep mode, and retrieves the table of coordinates of nodes next in line to come active and run the second application. In the meantime nodes belonging to dynamic switching set 3 continue to be in sleeping mode until the second set completes its task and receives commands from the base station to begin the execution of the third application. As previously stated, different sensing devices are selected for measuring different physical quantities as required by the running application and hence the type of data collected depends on the purpose of the executed application.
Figure 5.3: Active Nodes in first DSS running application one
Figure 5.4: Active Nodes in second DSS running application two
Figure 5.5: Active Nodes in third DSS running application three
Figure 5.6 shows the complete execution of the three applications running on WSN with total and individual group packets sent to the base. As previously mentioned, a total of 99 nodes was distributed on three DSS groups, each of which executed one application in a predefined sequence.

**Figure 5.6**: Total network packets and individual group packets exchanged during the execution of three applications on WSN
5.3.2.2 Experimental results for energy consumption and performance evaluation in our proposed work for WSN

The Castalia simulator takes into account several values of consumed power when simulating a WSN. They are mainly related to power consumed by the radio module being in different transmitting, receiving or listening modes, with a baseline power of 6mW that sums up the power consumed by the microcontroller and associated electronic circuits, and transit power when a node transits from sleep to RX or TX or vice versa. If we look at the radio module implemented in Castalia which is a chipcon C2420, the type that is most likely to be used for low power radio in wireless sensor network platforms, it features the following [113]:

- Multiple States, transmit, receive/listen, multiple (configurable) sleep states
- Transition delays from one state to another
- Multiple (configurable) transmission power levels
- Different power consumption for the different states and Tx levels used
- Multiple modes of operation (defined by modulation, data rate, bandwidth, noise floor and other parameters) that can dynamically change.
- Multiple modulation schemes (FSK, PSK etc.)
- Continuous calculation of RSSI (Received Signal Strength Indicator)
- CCA (Channel Clear Assessment) capability; interrupt to the MAC module

The RX MODES for the CC2420 module contains the following format:

Name, dataRate (kbps), modulationType, bitsPerSymbol, bandwidth (MHz), noiseBandwidth (MHz), noiseFloor (dBm), sensitivity (dBm), powerConsumption (mW)

This is mapped to the following row of quantities:

Normal, 250, PSK, 4, 20, 194, -100, -95, 62

It is important when it comes to power consumption calculations that the last quantity of 62mW represents radio power consumption when in receiving or listening/idle mode. Now we move to the Transmission section

The section TX_LEVELS implemented is values taken from the data sheet of the CC2420 and shown in two lines below:

Tx_dBm: 0  -1  -3  -5  -7  -10  -15  -25
Tx_mW: 57.42  55.18  50.69  46.2  42.24  36.3  32.67  29.04
The first line lists the output power of different transmission levels in dBm (decibels) at the antenna, whereas the second line represents the corresponding energy spent when transmitting in that particular power in line one. In the next two sections, we show the delay (in msec) of transmitting and switching between the three main radio states: RX (receive), TX (transmit) and SLEEP, as well as the associated consumed power (in mW):

<table>
<thead>
<tr>
<th></th>
<th>RX</th>
<th>TX</th>
<th>SLEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>-</td>
<td>0.01mSec</td>
<td>0.194mSec</td>
</tr>
<tr>
<td>TX</td>
<td>0.01mSec</td>
<td>-</td>
<td>0.194mSec</td>
</tr>
<tr>
<td>SLEEP</td>
<td>0.05mSec</td>
<td>0.05mSec</td>
<td>-</td>
</tr>
</tbody>
</table>

Delay Transition Matrix

<table>
<thead>
<tr>
<th></th>
<th>RX</th>
<th>TX</th>
<th>SLEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>RX</td>
<td>-</td>
<td>62mW</td>
<td>62mW</td>
</tr>
<tr>
<td>TX</td>
<td>62mW</td>
<td>-</td>
<td>62mW</td>
</tr>
<tr>
<td>SLEEP</td>
<td>1.4mW</td>
<td>1.4mW</td>
<td>-</td>
</tr>
</tbody>
</table>

Power Transition Matrix

The medium access control protocol (MAC) is an important part of the node’s behavior and for our simulation we used TunableMAC which is a duty-cycled MAC that employs a CSMA (Carrier Sense Medium Access) mechanism for transmission. It is a contention-based MAC that can be tuned to its persistence and backing-off policies. However, its major function is to duty-cycle the radio and to transmit an appropriate train of beacons before each data transmission to wake up potential receivers [113]

Duty-cycling the radio has a major impact on the power consumption, so tuning the MAC, which in turn controls when the radio is listening or sleeping, provides controllability of the node’s power consumption. The duty cycle is the time (in mSec) during which the node is in listening mode over the total cycle time (in mSec), so it is the fraction of time that the node stays on listening to the channel. However, this parameter needs to be used carefully, since switching from/to different states also consumes power and time delay transition, as we have seen in the two table matrices above.
5.3.2.2.1 **Effect of varying node data exchange packet rate on power consumption and performance.**

There are many other parameters affecting power consumption in the nodes and ultimately network power consumption during network life time. One important parameter is the packet rate (packets/second) at which the node exchanges data; our experimental results show that the choice of packet rate is an important parameter to carefully consider for saving power and directly related to the node’s active interval.

The experimental results shown in Tables 5.6, 5.7 and 5.8 are for three different values of energy consumed and energy remaining (in Joules) in a single node for packet rates, 1, 5 and 10 respectively. These results were for a network of 100 nodes (99 + sink), simulation time of 150 seconds and initial node energy of 100 Joules.

**Table 5.6: Energy consumption in Joules for node transmission with packet rate 1 pkt. /sec**

<table>
<thead>
<tr>
<th>Application: Remaining energy (in a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Remaining energy per group (DSS = 33 nodes each)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-------------------------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-------------------------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-------------------------------+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceManager: Consumed Energy (by a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
</tbody>
</table>
Table 5.7: Energy consumption in Joules for node transmission with packet rate 5 packets/sec

<table>
<thead>
<tr>
<th>Application: Remaining energy (in a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td>95.776</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Remaining energy per group (DSS = 33 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-------------------------------------------------------</td>
</tr>
<tr>
<td>3162.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceManager: Consumed Energy (by a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td>4.224</td>
</tr>
</tbody>
</table>

Table 5.8: Energy consumption in Joules for node transmission with packet rate 10 packets/sec

<table>
<thead>
<tr>
<th>Application: Remaining energy (in a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td>94.292</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Remaining energy per group (DSS = 33 nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-------------------------------------------------------</td>
</tr>
<tr>
<td>3054.87</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceManager: Consumed Energy (by a single node)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-----------------+</td>
</tr>
<tr>
<td>5.708</td>
</tr>
</tbody>
</table>
There is a marginal improvement in energy consumption when the packet rate increased from one to five per second, as the radio was in TX mode during most of the active interval of the TunableMAC and minimum time was spent in RX (once the transmission is complete, the mode of the node automatically switches to listening/receiving mode where energy consumption is slightly higher) before the node goes into sleeping mode.

We can also see from the results obtained in Table 5.8 that when the packet rate is increased to 10 per second, node energy consumption is increased. The fact that the radio consumes more power when in listening/receiving mode (62mW) compared to transmitting (47mW) explains why increasing packet the rate, for example in our case to 10 resulted in more power consumed than when we ran the simulation for one packet per second. The radio spends more time in RX mode (as opposed to TX mode which is less power hungry). In the TunableMAC protocol we used, at 10 packets per second, the transmitted packets took less time than the basic active period, which ended up staying longer in RX mode (more power hungry) and hence more power consumption. However, as we previously mentioned, careful parameter selection and time synchronization are key in energy conservation, for example our experimental results show better energy consumption when we had 5 packets per second but that only applied for the type of MAC protocol we selected.

Different MACs behave differently, for example in some MACs such as TMAC if the transmission of the packet in the slow rate extends further than the basic active period of 15ms, it means we are adding one more 15ms active period. Hence, if we are just over the 15ms period, then with a faster rate we could bring back transmission to less than 15ms and spend about half the energy we would spend in the slower rate.

In our case with TunableMAC, the extension is more gradual and the active period is extended only as long as we need to transmit (TX). If the active period is 10ms (listen interval) in TunableMAC and we need 11ms to transmit with the low rate, then the node will be in TX mode for 11ms and then go to sleep mode. Now had we doubled the rate, that would mean the transmission takes 5.5ms and then we are spending 10ms-5.5ms = 4.5ms more in RX mode to complete the active interval and hence more power hungry that would result in more power consumption.

The value of node sampling frequency is another important factor affecting the energy efficiency of the deployed sensor node, and this value will depend on what physical phenomenon is being monitored. If the sampling rate is too low, the probability of missing important events is viable;
on the other hand if it is too high, this affects the depletion of node power as the consumed power increases. A balanced approach is to alter this value dynamically to meet optimum performance.

As we also have seen, the frequency of transmitting a data packet is another important factor in the lifespan of a node, as this requires wireless transmission by the radio model which consumes a significant portion of the available resources. If real-time data collection is needed then one transmission can occur per sensed value, however in all other cases a group of multiple readings can be sent in one data packet or message. A node can intelligently decide whether or not it needs to transmit data based on comparing the current sensed data with a previous one and if not much change has occurred then there is no need for the transmission to take place.

5.3.2.2.2 Effect of applying DSS model to a network running a single application on power consumption and performance in WSN

Next, we discuss the experimental results obtained that came in line with our claim of prolonging network lifetime when the proposed DSS setup of three switching sets is applied to run a single application on WSN. We simulated two environment set ups as follows: 1- Running Single Application on wireless sensor network with the implementation of DSS mechanism 2-Running Single Application on wireless sensor network without the use of the proposed system design of DSS. The experimental results obtained are discussed in the following sections.

1 - Energy efficiency evaluation when applying DSS mechanism over single application running on WSN

In this scenario, we had a total of 99 nodes and one base station that ran for a total simulation time of 150 seconds and initial energy of 100 joules with packet rate of one packet per seconds. We applied the dynamic switching sets of three groups on a single application ran on WSN. This simulation setting is given by a ‘General’ configuration file, where one application was executed on multiple (three) sets of 33 nodes each. The experimental results of the average energy consumed by a node over the network life time as well as the remained energy are given in Table 5.9.
Table 5.9: Energy results for running a single application with DSS implementation.

Application: Remaining energy (in a single node)

| +----------------++----------------++----------------++----------------++----------------++----------------++----------------++----------------++
|                      |                      |                      |                      |                      |                      |                      |                      |
|                       | 95.763               |                       |                      |                      |                      |                      |                      |

Application: Remaining energy per group (DSS= 33nodes)

| | 1 | 2 | 3 |
| +----------------++----------------++----------------++----------------++----------------++----------------++----------------++
| 3161.73         | 3161.73         | 3165.49         |

ResourceManager: Consumed Energy (by a single node)

| +----------------++----------------++----------------++----------------++----------------++----------------++----------------++
|                      |                      |                      |                      |                      |                      |                      |                      |
|                       | 4.237               |                       |                      |                      |                      |                      |                      |

The experimental value obtained, 4.237 Joules, represents the average amount of energy consumed by each of the 99 nodes that ran in the network over 150 seconds. If we compare this value with the theoretical value calculated from below, we get:

Average Energy Node Consumption (based on DSS) = (energy consumed by TX/RX + Baseline power energy drawn) * Node Active Time On + 2 (base line energy consumption * Node Sleep Time) + (Energy Consumed due to transit from Sleep to R/TX states)

Average Energy Node Consumption = (0.062J/Sec+0.006J/Sec)*50sec + 2(0.006J/Sec*50sec) +0.0014J // All the substituted values are from the CC2420 Chipcon data sheet spec.

**Average Energy Node Consumption = 4.014 Joules**

The theoretical value calculated 4.014 Joules is well correlated to the experimental value of 4.237 obtained, which proves our results for the average node consumption on a network employing the dynamic switching set (DSS) model. Next, we ran the single application on the same network with the same parameter settings but without the DSS model applied.
2- Energy efficiency evaluation without DSS implementation over a single application running on WSN

In this scenario, one application was executed on all network nodes (99 nodes) for a total simulation time of 150 seconds with a packet rate of one packet per second, initial node energy 100J. The configuration file used in Castalia to simulate this scenario is named ‘withoutGroup’ configuration file. The average energy consumption/remain for one node result obtained is given in Table 5.10. Both simulation scenarios were run for one seed and will later see the results when more than one seed (20 repetitions) were also computed.

Table 5.10: Energy results of average node consumption when running a single application without DSS implementation

| Application: Remaining energy (in one node) | +-----------------+ |
|                                             |                  |
|                                             | 89.811           |

| Application: Remaining energy per group      | +-----------------+ |
|                                             |                  |
|                                             | 8896.83          |

| ResourceManager: Consumed Energy (by one node)| +-----------------+ |
|                                             |                  |
|                                             | 10.189           |

Similarly, the experimental value obtained, 10.189 Joules, represents the average amount of energy consumed by each of the 99 nodes that ran in the network over 150 seconds, but now without the DSS implementation. If we compare this value with the theoretical value calculated from the formula below, we get:
Average Energy Node Consumption (No DSS) = (energy consumed by TX/RX + Baseline power energy drawn) * Node Active Time On

Average Energy Node Consumption = (0.062J/Sec+0.006J/Sec)*150sec // All the substituted values are from the CC2420 Chipcon data sheet spec.

**Average Energy Node Consumption = 10.2 Joules**

It can be seen that the experimental value of the average energy consumed, 10.189J, is equal to the value obtained from theoretical calculations based on the radio module data sheet spec values.

Analyzing the two values of average energy consumption with and without applying the dynamic switching sets algorithm, it is clear that a significant improvement in node energy consumption is achieved with the implementation of DSS, and a substantial energy saving yielding more than 50% in each node and hence the overall network. Prolonging network life time is an important major achievement in our proposed work when using the network to run single application.

Figure 5.7 (a) shows the remaining energy in joules for the two simulated environments of WSN, first with the implementation of DSS mechanism and second without DSS. It is clear that network energy performance is improved significantly and energy savings are achieved with the use of our proposed model of DSS. We see the remaining energy after 150 seconds is 95.763J using DSS versus 89.811J without DSS. Figure 5.7 (b) also shows the total amount of energy in Joules consumed by a node when applying DSS and without DSS of 4.237J and 10.189J respectively. The improvement in energy consumption of each node is about 58% with the Dynamic Switching Set Model of the total energy consumed per node.
Figure 5.7: (a) Remaining energy in joules of a node, (b) Average energy consumption in Joules of a node with and without DSS mechanism.
5.3.2.2.3 Effect of running simulations with more than one seed (repetition)

We also decided to run our simulations with more than one set of random seeds (i.e. more than one set of produced random number seeds). In Castalia this is implemented by using the –r switch followed by the number of repetitions we want to run (each repetition is run with a different set of random seeds). We used 20 repetitions and the results obtained are shown in Table 5.11. The remaining energy values obtained represent the average of all nodes when running with DSS model implementation and without it. This is very useful to show how much energy is saved when compared to non-DSS configuration.

Table 5.11: Energy performance comparison with 20 seeds (repetitions) simulation with and without DSS implementation for running single application on a WSN

<table>
<thead>
<tr>
<th>Application: Remaining energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>+-------------------------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| Application: Remaining energy per group |
| +----------------------------------------+|
| | General | 1 | 2 | 3 | +---------+|
| | 3161.727 | 3161.726 | 3165.555 | +---------+|
| | withoutGroup | 8896.867 | 0 | 0 | +---------+|

| ResourceManager: Consumed Energy |
| +-------------------------------+|
| | General | withoutGroup | +---------+|
| | 4.237  | 10.189       | +---------+|

The experimental results obtained and discussed above support our claim of improving and prolonging network lifetime with the use of the proposed design model of organizing sensor network into multiple switching sets, each running the same application in a sequential defined logic. All nodes on the distributed field of interest are equipped with the same sensing devices and uploaded with the same application. However, the density control of these nodes has been exploited by splitting them into three dynamic switching sets (DSS), whose mode of operation is controlled by the base station (BS). At the start of the simulation, control commands are sent as packets of data from BS to nodes whose coordinates/addresses are calculated and saved in a look-up table. This first group of nodes comprises the first set of DSS that remain active for the defined amount of time to execute and run the task1 application, while nodes of groups 2 and 3 transit to
sleeping mode. Once the first application running on DSS1 nodes is complete, the BS then instructs this group to go to sleep mode and initiates commands for the DSS2 group to wake up and run actively the application. This is repeated for the third set of nodes in DSS3, which gets activated to continue running the application in the network while both DSS1 and DSS2 are in sleeping mode. Ideally, the network life time is three times the traditional network lifetime (if zero energy is consumed when nodes are in sleeping mode). However, in real applications and simulations, sensor nodes continue to consume energy even when in sleep mode. In fact, as previously mentioned, when the radio completes transmitting packets and the active period is still on, the node transit to RX/listening mode before going to sleep and that adds to energy consumption. Even in sleeping mode, nodes tend to turn on their radio from time to time as they need to check if they are called for duty.

By the time the third set of DSS is up and running, the first DSS set would have waited in a sleeping mode twice the time of the second DSS and hence the life time of nodes each round is less than the preceding round as energy is still being consumed, due to the fact that each radio module needs from time to time to transit from idle/sleep mode to active mode for a short time slot to check if any commands have arrived. In conclusion, the results have proved that the model has significantly improved the energy saving of around 58.4% when compared to a network running without DSS implementation. When comparing this energy saving accomplishment to other work conducted on energy saving our results outperform what has been achieved by other research work related to dense networks. The work presented in [125] suggests a DSF (Dynamic Switching-Based Data Forwarding) method that optimizes, expected data delivery ratio, expected communication delay and expected energy consumption for low-duty-cycle WSN. The concept presented in this work mainly applies to networks under unreliable communication links. The energy saving results depends on the duty-cycle variation of the node (active/dormant), however there was no mention on the energy consumption impacted by different latency timings resulted from different duty cycle settings.
5.3.3 Second Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s SimulationTime | NodeInitialEnergy 18720J | PLD0 55dB

The three dynamic switching sets algorithm for running three applications on a WSN is now implemented with different parameter values related to some different deployment environments with regard to packet collisions in networks and the effect of their reception at the base station. It is only of our interest to see the impact of introducing sigma and its variation on the average path loss of the wireless channel as well as introducing the collision model to a value where signal strength and floor noise are taken into calculations when receiving a packet. The node disk range is kept on its default value of 28m, however in the next simulation it is varied to show the connectivity effect that results from decreasing the range. In this scenario we have the same three groups deployed, each with 33 nodes and a total of 100 nodes including the base station. The simulation time is 150 seconds and the initial node energy used is 18720J (equivalent to two AA batteries with which the sensor node is deployed). The experimental results given in Table 5.12 show the total packets received per DSS group exchanged during its active time when running an application, also the average number of packets received by each individual node. The experimental results obtained in Table 5.12 reflect the effect of the wireless channel parameters’ variations on the exchanged packets in the network. There is a significant decrease in the number of packets arriving at the base station and hence received by the base station. The average number dropped to 30 packets received from each node compared to the average of 49 packets in the first simulation. The collision model introduced has also affected packets arrival at the base station due to interference with nodes each other’s signals. The average energy consumption by each node, as shown in Table 5.13 remained unchanged to 4.237J as each node sent 50 packets during the interval of its DSS time (50sec), however, because of the wireless channel variations and interface only 30 packets were received at the base station.

The wireless channel and radio module parameter values for this simulation are given below, they are all explained in section 5.2.3.

- SN.wirelessChannel.Sigma = 4.0
- SN.wirelessChannel.bidirectionalSigma = 1.0
- SN.node[*].Communication.Radio.mode = “Normal”
- SN.node[*]. Communication.radio.collisionModel = 2
Table 5.12: Average packets received by the base station per node with wireless channel parameter variation, sigma = 4.0, collision model = 2.0, path loss = 55dB

Application: Packets received per group

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>884</td>
<td>1011</td>
<td>958</td>
</tr>
</tbody>
</table>

Application: Packets received per node

| 30.032 |

Table 5.13: Average energy consumed by a node and remaining energy in Joules with wireless channel parameter variations, sigma = 4.0, collision model=2.0, path loss =55dB

Application: Remaining energy

| 18715.773 |

Application: Remaining energy per group

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>617622</td>
<td>617622</td>
<td>617626</td>
</tr>
</tbody>
</table>

Resource Manager: Consumed Energy

| 4.237 |

In Table 5.14, we also show a sample of the data generated by the output of the application module of the base station for this simulation. The time trace shows time instants during the 150s simulation interval at which a certain packet number was sent by a node and delivered to the base station. Node IDs are the active nodes in their dynamic switching sets sending packets to the base station when executing their assigned applications 1, 2 and 3. In the first 50sec simulation interval, only 33 of the 99 nodes are active running application-1 staying in TX/RX mode while the remaining 66 nodes are at energy saving sleep mode (radios in idle/listening state).
Once application-1 is complete, the second set of DSS nodes wake-up active for the second 50sec time interval of the total simulation time running the second application. Meanwhile nodes subscribed to the first and third DSS set groups transit to sleep mode. Finally the third group of 33 nodes becomes active for the last 50sec interval of simulation time running application-3 while the other two groups (1 and 2) go to sleep mode. All the time instants of these active and sleeping nodes are recorded in the Castalia-Trace.txt file. Only a sample of fifty time instants is shown in Table 5.14.

Table 5.14: Second Simulation, trace sample of time stamp of packets sent to the base station under wireless channel parameter variations, sigma = 4, collision model = 2.0, path loss = 55dB

<table>
<thead>
<tr>
<th>Trace #</th>
<th>Simulation Time</th>
<th>Module Produced Trace</th>
<th>Trace Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.015257440912</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 54</td>
</tr>
<tr>
<td>2</td>
<td>1.022671555917</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 45</td>
</tr>
<tr>
<td>3</td>
<td>1.027120351497</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 78</td>
</tr>
<tr>
<td>4</td>
<td>1.03141236742</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 15</td>
</tr>
<tr>
<td>5</td>
<td>1.049307761072</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 69</td>
</tr>
<tr>
<td>6</td>
<td>1.060199227686</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 21</td>
</tr>
<tr>
<td>7</td>
<td>1.07010511714</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 93</td>
</tr>
<tr>
<td>8</td>
<td>1.074521741294</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 39</td>
</tr>
<tr>
<td>9</td>
<td>1.0788387925</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 9</td>
</tr>
<tr>
<td>10</td>
<td>1.083450157053</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 27</td>
</tr>
<tr>
<td>11</td>
<td>1.088153364139</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 42</td>
</tr>
<tr>
<td>12</td>
<td>1.109279840664</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 84</td>
</tr>
<tr>
<td>13</td>
<td>1.104663648931</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 33</td>
</tr>
<tr>
<td>14</td>
<td>1.10904407723</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 81</td>
</tr>
<tr>
<td>15</td>
<td>1.127797857607</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 57</td>
</tr>
<tr>
<td>16</td>
<td>2.015133783161</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 87</td>
</tr>
<tr>
<td>17</td>
<td>2.019609790967</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 21</td>
</tr>
<tr>
<td>18</td>
<td>2.031468746956</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 48</td>
</tr>
<tr>
<td>19</td>
<td>2.036548994547</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 24</td>
</tr>
<tr>
<td>20</td>
<td>2.040994343742</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 78</td>
</tr>
<tr>
<td>21</td>
<td>2.059559673783</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 39</td>
</tr>
<tr>
<td>22</td>
<td>2.071885454191</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 30</td>
</tr>
<tr>
<td>Trace #</td>
<td>Simulation Time</td>
<td>Module Produced Trace</td>
<td>Trace Message</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>23</td>
<td>2.085255820357</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 84</td>
</tr>
<tr>
<td>24</td>
<td>2.089563870029</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 57</td>
</tr>
<tr>
<td>25</td>
<td>2.108463219785</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 42</td>
</tr>
<tr>
<td>26</td>
<td>2.113369142677</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 9</td>
</tr>
<tr>
<td>27</td>
<td>2.12511786495</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 33</td>
</tr>
<tr>
<td>28</td>
<td>2.139943420617</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 69</td>
</tr>
<tr>
<td>29</td>
<td>2.147875248461</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 81</td>
</tr>
<tr>
<td>30</td>
<td>3.02486967707</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 78</td>
</tr>
<tr>
<td>31</td>
<td>3.038814072244</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 42</td>
</tr>
<tr>
<td>32</td>
<td>3.069455720176</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 9</td>
</tr>
<tr>
<td>33</td>
<td>3.076157424001</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 69</td>
</tr>
<tr>
<td>34</td>
<td>3.080657326247</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 18</td>
</tr>
<tr>
<td>35</td>
<td>3.092594810628</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 81</td>
</tr>
<tr>
<td>36</td>
<td>3.06979905496</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 84</td>
</tr>
<tr>
<td>37</td>
<td>3.102972994905</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 27</td>
</tr>
<tr>
<td>38</td>
<td>3.115326869369</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 93</td>
</tr>
<tr>
<td>39</td>
<td>3.121145738323</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 39</td>
</tr>
<tr>
<td>40</td>
<td>3.126033721413</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 24</td>
</tr>
<tr>
<td>41</td>
<td>3.132178782089</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 3</td>
</tr>
<tr>
<td>42</td>
<td>3.138016255879</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 33</td>
</tr>
<tr>
<td>43</td>
<td>3.14429766082</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 57</td>
</tr>
<tr>
<td>44</td>
<td>4.015196081015</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 54</td>
</tr>
<tr>
<td>45</td>
<td>4.025241986744</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 69</td>
</tr>
<tr>
<td>46</td>
<td>4.034587682319</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 36</td>
</tr>
<tr>
<td>47</td>
<td>4.051320276622</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 30</td>
</tr>
<tr>
<td>48</td>
<td>4.055668628949</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 63</td>
</tr>
<tr>
<td>49</td>
<td>4.059980079373</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 21</td>
</tr>
<tr>
<td>50</td>
<td>4.072867288061</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 9</td>
</tr>
</tbody>
</table>
5.3.4 Third Simulated Scenario – Three sets of DSS groups running multiple applications.100 Nodes|150s 
SimulationTime | NodeInitialEnergy 18720J | PLD0 62dB

In this last scenario, we opted to investigate the effect of variation of the path loss parameter of the wireless channel, which is directly related to the disk range of a node. The value of the path loss parameter (PLD0) in this scenario was changed to 62dB from its default value set in the Castalia Simulator of 55dB, this in turn reduced the communication disk range of a node from 28m to 15m (calculations are given in section 5.2.3). The simulation results of the average packets received are shown in Table 5.15 and we can conclude how the number of packets received by the base station has been significantly reduced as a result of increasing the path loss value to 62dB and its impact on the communication range of the node. We also present in Table 5.17 a sample of 50 instants of the data generated by the output of the application module of the base station for this simulation scenario. The recorded node IDs and the packets delivered are only those that reached the base station. When comparing the data in this trace table to the other traces we see in this case and during a certain time interval fewer instants recorded for the same time interval in the previous time stamp traces recorded by the base station. Wireless channel parameters, collision models and node disk range must be carefully designed and selected alongside the size of the field of interest and the number of nodes deployed. In this scenario, the three applications ran as per the algorithm design and the dynamic switching sets for each application successfully switched to saving mode once their task was completed. Also recorded in Table 5.16 is the average energy consumed by each node. It can be seen from the experimental results in Table 5.16 that the average energy consumed in each node reflects the number of packets each node should have exchanged, which is approximately 50 packets, however due to the aforementioned parameter settings pertaining to the node disk range and other wireless channel variations only fewer than half the number made it and reached the base station, i.e. basically those nodes that were within communication range and packets with higher signal strength were received.
**Table 5.15**: Average packets received by the base station from each DSS set with variation of wireless channel parameters and setting disk range of the node to 15m.

<table>
<thead>
<tr>
<th>Application: Packets received per group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>271</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Packets received per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.519</td>
</tr>
</tbody>
</table>

**Table 5.16**: Average node energy consumed and remaining energy (in Joules) after 150s simulation time with path loss value 62dB

<table>
<thead>
<tr>
<th>Application: Remaining energy per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>18715.773</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application: Remaining energy per group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>617622</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResourceManager: Consumed Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.237</td>
</tr>
</tbody>
</table>

160
Table 5.17: Third Simulation, trace sample of time stamp of packets sent to the base station under wireless channel parameter variations, sigma = 4, collision model = 2.0, path loss = 62dB

<table>
<thead>
<tr>
<th>Trace #</th>
<th>Simulation Time</th>
<th>Module Produced Trace</th>
<th>Trace Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.026052157495</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 9</td>
</tr>
<tr>
<td>2</td>
<td>1.036393417211</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 84</td>
</tr>
<tr>
<td>3</td>
<td>1.041878310454</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 27</td>
</tr>
<tr>
<td>4</td>
<td>1.07800572622</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 42</td>
</tr>
<tr>
<td>5</td>
<td>1.093894688843</td>
<td>SN.node[0].Application</td>
<td>Received packet#0 from node 93</td>
</tr>
<tr>
<td>6</td>
<td>1.10140917648</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 69</td>
</tr>
<tr>
<td>7</td>
<td>2.025328533618</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 39</td>
</tr>
<tr>
<td>8</td>
<td>2.041100294046</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 84</td>
</tr>
<tr>
<td>9</td>
<td>2.08458189334</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 42</td>
</tr>
<tr>
<td>10</td>
<td>2.100508577341</td>
<td>SN.node[0].Application</td>
<td>Received packet#1 from node 81</td>
</tr>
<tr>
<td>11</td>
<td>2.107574600135</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 33</td>
</tr>
<tr>
<td>12</td>
<td>2.121454736514</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 48</td>
</tr>
<tr>
<td>13</td>
<td>3.026090278481</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 9</td>
</tr>
<tr>
<td>14</td>
<td>3.031343166968</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 42</td>
</tr>
<tr>
<td>15</td>
<td>3.036116289359</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 9</td>
</tr>
<tr>
<td>16</td>
<td>3.053462079165</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 42</td>
</tr>
<tr>
<td>17</td>
<td>3.074830480655</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 9</td>
</tr>
<tr>
<td>18</td>
<td>3.101025257641</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 42</td>
</tr>
<tr>
<td>19</td>
<td>3.112118984118</td>
<td>SN.node[0].Application</td>
<td>Received packet#2 from node 9</td>
</tr>
<tr>
<td>20</td>
<td>4.035569076299</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 84</td>
</tr>
<tr>
<td>21</td>
<td>4.070048262788</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 39</td>
</tr>
<tr>
<td>22</td>
<td>4.078709591405</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 42</td>
</tr>
<tr>
<td>23</td>
<td>4.0911106071911</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 48</td>
</tr>
<tr>
<td>24</td>
<td>4.102744440349</td>
<td>SN.node[0].Application</td>
<td>Received packet#3 from node 81</td>
</tr>
<tr>
<td>25</td>
<td>5.057903654145</td>
<td>SN.node[0].Application</td>
<td>Received packet#4 from node 9</td>
</tr>
<tr>
<td>26</td>
<td>5.079102263834</td>
<td>SN.node[0].Application</td>
<td>Received packet#4 from node 42</td>
</tr>
<tr>
<td>27</td>
<td>5.091798144775</td>
<td>SN.node[0].Application</td>
<td>Received packet#4 from node 93</td>
</tr>
<tr>
<td>28</td>
<td>5.099691119722</td>
<td>SN.node[0].Application</td>
<td>Received packet#4 from node 33</td>
</tr>
<tr>
<td>Trace #</td>
<td>Simulation Time</td>
<td>Module Produced Trace</td>
<td>Trace Message</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
<td>----------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>29</td>
<td>6.02515543987</td>
<td>SN.node[0].Application</td>
<td>Received packet#5 from node 69</td>
</tr>
<tr>
<td>30</td>
<td>6.063276637379</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 27</td>
</tr>
<tr>
<td>31</td>
<td>6.069889968944</td>
<td>SN.node[0].Application</td>
<td>Received packet#5 from node 42</td>
</tr>
<tr>
<td>32</td>
<td>6.08278069706</td>
<td>SN.node[0].Application</td>
<td>Received packet#5 from node 9</td>
</tr>
<tr>
<td>33</td>
<td>6.094143744663</td>
<td>SN.node[0].Application</td>
<td>Received packet#5 from node 84</td>
</tr>
<tr>
<td>34</td>
<td>6.108360502052</td>
<td>SN.node[0].Application</td>
<td>Received packet#5 from node 93</td>
</tr>
<tr>
<td>35</td>
<td>7.024242647806</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 87</td>
</tr>
<tr>
<td>36</td>
<td>7.04245375339</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 33</td>
</tr>
<tr>
<td>37</td>
<td>7.055505045428</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 39</td>
</tr>
<tr>
<td>38</td>
<td>7.080868077431</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 27</td>
</tr>
<tr>
<td>39</td>
<td>7.093903768889</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 81</td>
</tr>
<tr>
<td>40</td>
<td>7.117429497521</td>
<td>SN.node[0].Application</td>
<td>Received packet#6 from node 69</td>
</tr>
<tr>
<td>41</td>
<td>8.025068892996</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 69</td>
</tr>
<tr>
<td>42</td>
<td>8.049394369468</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 39</td>
</tr>
<tr>
<td>43</td>
<td>8.086226926634</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 33</td>
</tr>
<tr>
<td>44</td>
<td>8.090986909573</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 42</td>
</tr>
<tr>
<td>45</td>
<td>8.098217826577</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 9</td>
</tr>
<tr>
<td>46</td>
<td>8.102703442916</td>
<td>SN.node[0].Application</td>
<td>Received packet#7 from node 87</td>
</tr>
<tr>
<td>47</td>
<td>9.026204641439</td>
<td>SN.node[0].Application</td>
<td>Received packet#8 from node 9</td>
</tr>
<tr>
<td>48</td>
<td>9.04407453937</td>
<td>SN.node[0].Application</td>
<td>Received packet#8 from node 39</td>
</tr>
<tr>
<td>49</td>
<td>9.064793362427</td>
<td>SN.node[0].Application</td>
<td>Received packet#8 from node 81</td>
</tr>
<tr>
<td>50</td>
<td>9.07519955759</td>
<td>SN.node[0].Application</td>
<td>Received packet#8 from node 84</td>
</tr>
</tbody>
</table>

## 5.4 Summary

The proposed research system model algorithm has been tested and implemented for a number of different simulation scenarios for the objectives of running multiple applications on a WSN and prolonging network life time when only a single application is executed. The simulation software used was Castalia simulator based on an OMNeT++ framework platform, as it meets a number of important criteria such as open source software, realistic wireless channel and energy models, customizable using C++ and actively supported by its developers and the research community.
The experimental results presented in this chapter are for the following simulated scenarios:

1- First Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s SimulationTime | NodeInitialEnergy 100J | PLD0 55dB
   a. Experimental results for running three different applications on a WSN
   b. Experimental results for energy consumption and performance evaluation for the proposed model on WSN
      i. Effect of varying node data exchange packet rate on power consumption and performance (1, 5 and 10 packets/sec)
      ii. Effect of applying DSS model on energy consumption and performance on a network running single application (with and without DSS mechanism)
      iii. Effect of running simulation with more than one seed (repetitions)

2- Second Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s SimulationTime | NodeInitialEnergy 18720J | PLD0 55dB

3- Third Simulated Scenario – Three sets of DSS groups running multiple applications. 100 Nodes | 150s SimulationTime | NodeInitialEnergy 18720J | PLD0 62dB

The evaluation results of the proposed algorithm was primarily based on theoretical analysis and extensive simulations conducted on a network of 100 sensor nodes using OMNeT++ platform. The physical testbed hardware implementation was impractical as high cost is associated and because the algorithm is designed for dense, large scale WSN. However, at an early work stage we had implemented two hardware setups of sensor nodes for the DSS model using the available motes. The first set up was creating a network of physical testbed of two motes and a base station, a starter kit from crossbow. That kit contained two IRIS sensor nodes, each with MTS 400 environmental sensor board mounted on the node and a base station/gateway MIB 520 (node ID0). The interfacing software used was MoteView that collected incoming data from the base station via USB and displaying on the screen with time plots of sensor values and topology of the network. The second hardware setup was conducted using physical testbed of four Kmote nodes with built-in outer antenna, similarly sensed data was sent to the base station for processing and analysis.
Chapter 6

6 Conclusions

This chapter discusses the conclusion and direction for future work. The conclusion summarizes the analysis results of testing running three applications on a network as well as the significant improvement in energy saving when running single application under the implementation of DSS mechanism. In the second part we discuss the future directions for our work to incorporate a mobile base station that runs on a predefined calculated projectile path for better energy node balance in the network. As well we discuss the idea of applying the vulnerable state in a sensor node [117] as a third additional state particularly for nodes concentrated at the center area of the field. This would further enhance energy saving as well as reduce packet traffic congestion in the network.

This thesis has addressed several major issues related to the limitations of running a single application on a dense WSN when deployed in the field of interest. One of the main issues in a dense network running a single application is the waste of resources when it comes to consider that extra nodes deployed are to serve a redundancy purpose only, and the issue of total cost pertaining to a return of investment that is directly related to the ‘usefulness’ of the network deployed. Addressing these issues, coupled with the high market demand to have a single network able to perform multiple task functions, have led us to work on the proposed research. The required number of nodes to be deployed is an important element to be considered when deploying a WSN, and plays a key role in achieving our proposed goals of running multiple applications on a network and achieving energy enhancement when applying our proposed model to run only a single application on the network and hence prolonging network life time. Achieving the two goals establishes a wireless sensor network that features modular and flexible infrastructure.

6.1 Summary of Contributions

The thesis began by first identifying the limitations of deploying a WSN to only run a single application and the need to have multiple applications running on a network for meeting the demands of more sophisticated applications as well as for better returns on investment. Our main findings focused on the simulation-based evaluations for the following two objectives:
- **Infrastructure system model to run multiple applications on a WSN**

Our proposed system’s algorithm first divides the field of interest (FOI) into a number of equal partitions called Dynamic Switching Sets (DSS). The area size of each individual DSS is calculated based on our new calculated node’s communication range (omnidirectional – disk range) that will ultimately determine the number of nodes needed in each DSS. The quantity of nodes and hence the density within one DSS is another criterion to follow, since an average number of nodes needs to be equally allocated to each DSS to collectively achieve the required network connectivity and coverage. Those newly created DSS are then tagged to three main sets corresponding with their geographical locations on the field. Once each of the three main sets identifies its DSS, the next step is to identify and allocate sensor nodes for each DSS partition. The term ‘allocation’, refers here to the selection of sensor nodes out of a set of available nodes which are used to execute a request for an application. Allocation of nodes in our proposed algorithm follows specific criteria, one of which being that a node needs to be within the same geographic area of the partition bounded by the virtual radial lines of the segment that is called dynamic switching set. Once the hierarchal structure of the network is completed (three main sets, each set consisting of a number of equal number of DSS, each DSS consists of a number of sensor nodes and a cluster head, the base station begins the scheduling of the applications switching mechanism. Scheduling, deals with the order in which the application requests are executed. This is one of the planner coordinator’s responsibilities, a software that lies in the base station. The execution sequence for our proposed research is determined and hardcoded prior to deployment, based on the tasks mission needed to be completed for the sensing field. At the start of the network life cycle, the base station instructs two of the three main sets 2 and 3 to have all their DSS transit into energy saving/sleeping mode and keep only DSS of set 1 to run the first application. Once the time interval of the first application is complete, the second set of DSS come active and nodes begin executing the second application while the DSS of sets 1 and 3 are in sleeping mode. Finally, the third set of DSS wakes-up to run the third application on its nodes while moving all nodes in the sets 1 and 3 into energy saving mode. The simulation experimental results have successfully showed the switching mechanism of the three sets and their DSS during the life time of 150 seconds.

- **Prolonging network life time when only a single application needs to run**

The dynamic switching set mechanism was then applied on a network where only single application was needed to run over the entire network lifetime. The experimental results obtained showed that the average energy consumption of each node in the network with the DSS model implemented
by the algorithm was 4.237 Joules during the simulation time of 150 seconds. Then the same single application was executed on the network without the DSS model and the average energy consumption of a node came to 10.189 Joules during the 150 seconds period. This shows a remarkable energy improvement of 58% with the algorithm implementation of our DSS model and hence prolongation of network life time. The energy saving was also experimented with using different parameters, most importantly the node’s packet rate transmission with three different values of 1, 5 and 10 packets per second.

6.2 Future Work

There are several promising future directions for this work as there is still much work to be done both in areas of running multiple applications on a single network and enhancing energy efficiency design mechanisms and protocols.

A future enhancement would be adding extra scheduling criteria to the nodes, particularly those located near the base station area to reduce the congested packet traffic and to better exploit some redundant nodes in the proximity of the base station. The work proposed in [117] can be utilized in this respect so that each of these sensors can have one additional state which is called vulnerable state. A sensor is always in one of three states: active, idle, or vulnerable. In the vulnerable state, if the sensor discovers that its sensing area cannot be fully covered by any of its active or vulnerable members, it immediately turns itself into the active state. Otherwise, it enters the idle state if its sensing area can be fully covered by any of its active or vulnerable members with a higher energy level. Therefore, it is possible for multiple neighbouring sensors to enter the idle state simultaneously so as to prolong system lifetime.

Another future work idea is to implement a mobile base station for the deployed network, such that it would travel on a predefined trajectory axis within the internal sensing field. This would also contribute to reduce the non-uniform energy consumption among the sensor node and prolong network life time.
Bibliography


