

University of Windsor

## Scholarship at UWindor

---

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

---

2009

# Design and Implementation of a Cross-Platform Sensor Network for Transmission Line Monitoring

Patrick Casey  
*University of Windsor*

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

---

### Recommended Citation

Casey, Patrick, "Design and Implementation of a Cross-Platform Sensor Network for Transmission Line Monitoring" (2009). *Electronic Theses and Dissertations*. 116.  
<https://scholar.uwindsor.ca/etd/116>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email ([scholarship@uwindsor.ca](mailto:scholarship@uwindsor.ca)) or by telephone at 519-253-3000ext. 3208.

# Design and Implementation of a Cross-Platform Sensor Network for Transmission Line Monitoring

By

Patrick Casey

A Thesis

Submitted to the Faculty of Graduate Studies  
through the Department of Electrical and Computer Engineering  
in Partial Fulfillment of the Requirements for the  
Degree of Master of Applied Science at  
the University of Windsor

Windsor, Ontario, Canada  
2009

© 2009 Patrick Casey

All Rights Reserved. No Part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium by any means without prior written permission of the author.

Design and Implementation of a Cross-Platform Sensor Network for  
Transmission Line Monitoring

by

Patrick Casey

APPROVED BY:

---

Dr. Arunita Jaekel  
School of Computer Science

---

Dr. Narayan Kar  
Department of Electrical and Computer Engineering

---

Dr. Kemal Tepe, Advisor  
Department of Electrical and Computer Engineering

---

Dr. Rashid Rashidzadeh, Chair of Defense  
Department of Electrical and Computer Engineering

17 August, 2009

---

## **Author's Declaration of Originality**

---

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

---

## **Abstract**

---

The current era is calling for a greater electricity demand than ever seen before. There is much at stake with health care, manufacturing, communications, and safety systems relying on the assumption that a constant supply of power is always available. Slow information feedback and hesitation to relay the information to other power stations was a major contributor to North America's northeast blackout of 2003. Had fault information been properly passed on, the blackout area would have been much smaller. From experience, it can be seen that having the knowledge of knowing exactly when and what is happening on the power distribution grid is extremely valuable. This thesis develops a practical, self configuring, sampling and forwarding scheme for transmission line monitoring. A hierarchical communication topology is also proposed. The developed prototype operates successfully and its functionality is fully documented. In addition, a ZigBee testbed is developed and used to determine its ability to perform in harsh environments, such as the one that may be found in a transmission line or power station environment. This research found that ZigBee devices are able to perform suitably with harsh surroundings.

---

## **Acknowledgements**

---

I would like to give my deepest thanks to my supervisor Dr. Kemal Tepe for his unwavering support. I have truly enjoyed working with you and I consider myself lucky to have had the opportunity to have done so. I would also like to thank my committee members Dr. Narayan Kar and Dr. Arunita Jaekel for their valuable input and guidance throughout this research process. I would like to express my gratitude to my fellow lab members Brajendra Singh, Nicholas Doyle, and Anshuman Garg for their involvement and feedback. A special thanks to my loving family for always being there and believing in me and to Marisa Bedard for her constant encouragement and support.

---

## Table of Contents

---

<b>Author's Declaration of Originality</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>Acknowledgements</b>	<b>vi</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xi</b>
<b>Abbreviations</b>	<b>xii</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Thesis Contribution	2
1.4 Background	3
1.4.1 Types of Faults and Consequences	3
1.4.2 Smart Meters	4
1.4.3 Why WLAN and ZigBee	6
1.5 Thesis Organization	6
<b>2. Related Work</b>	<b>7</b>
2.1 Sensors	7
2.2 Sensor Management and Routing	8
2.3 Long Range Data Transmission	9
2.4 Security	11



2.5	Alternative Applications	11
2.6	Summary	12
<b>3.</b>	<b>WLAN Performance</b>	<b>13</b>
3.1	Multi-hop	13
3.2	Long Range Data Transmission	13
3.3	WLAN and ZigBee Co-existence Interference	14
3.4	Low-Power Access Points	15
3.5	Security	15
3.6	Summary	16
<b>4.</b>	<b>ZigBee Performance</b>	<b>17</b>
4.1	Introduction	17
4.2	Testbed Components and Structure	20
4.3	Experiment Procedure and Results	23
4.3.1	Indoor	23
4.3.2	Outdoor	25
4.3.3	Automotive Internal Monitoring	26
4.3.4	Machine Shop Floor	27
4.4	Summary	29
<b>5.</b>	<b>Transmission Line Monitoring System</b>	<b>31</b>
5.1	Overall Proposed System	31
5.2	Hardware and Software	32
5.3	System Requirements	34
5.4	Control Centre Node	35
5.4.1	Functionality	35
5.4.2	Graphical User Interface	35
5.5	Cloud Network Node Functionality	37
5.6	Gateway Node Functionality	38
5.6.1	WLAN Thread	39

	CONTENTS
5.6.2 ZigBee Thread	39
5.7 Sensor Node Functionality	48
5.7.1 Normal Operation	48
5.7.2 Received Commands	49
5.7.3 LED States	50
5.8 System Robustness	51
5.9 Performance Metrics	52
5.10 Extra System Information	54
5.10.1 Overall System	54
5.10.2 Gateway Node	56
5.10.3 Sensor Node	57
5.10.4 ZigBee Testbed	58
<b>6. Conclusion and Future Work</b>	<b>60</b>
6.1 Conclusion	60
6.2 Future Work	61
<b>References</b>	<b>63</b>
<b>Appendix</b>	<b>68</b>
A Programming the MicaZ Motes	68
B Configuring WLAN Settings	69
C ZigBee Testbed Results Chart	70
D Program Codes	72
D.1 Control Centre Node	72
D.2 Cloud Mesh Node	80
D.3 Gateway Node	84
D.4 Sensor Node	91
<b>Vita Auctoris</b>	<b>97</b>

---

## List of Figures

---

Figure:	Page:
1.1 An Example Smart Meter Residential Deployment	5
4.1 ZigBee Testbed Packet Structures for Control and Data Packets	21
4.2 ZigBee Testbed Structure	22
4.3 ZigBee Testbed Packet Transmission Sequence	22
4.4a Main Floor Layout	24
4.4b Basement Layout	24
4.5 Test Vehicle and Transmission Locations	26
4.6 Machine Shop Layout	28
5.1 Overall Transmission Line Monitoring System	31
5.2 Overall Transmission Line Monitoring Test System	32
5.3 Control Centre's GUI Screen Shot	36
5.4 Cloud Network's Node Screen Shot	38
5.5 Gateway's Address ID Table and Sequence Number Table Structure	42
5.6 DNGA Packet Communication Procedure	43
5.7 DNGA Problem 1 Depicting Node Addresses	44
5.8 DNGA Problem 2 Depicting Packet Sequence Numbers	46
5.9 Gateway Node's Screen Shot	47
5.10 Monitoring System's ZigBee Packet Structures	47
5.11 ZigBee Receiver Throughput	53

---

## List of Tables

---

Table:	Page:
5.1 Monitoring System's ZigBee Packet “Type” Field Values	48
5.2 Sensor Nodes' LED States and Description	50
5.3 IP Addresses Used for the WLAN Interfaces	55
5.4 Modified Files and Description for BS in ZigBee Testbed	59
5.5 Useful TinyOS Parameters	59
C.1 ZigBee Testbed Locations and Results	70

---

## Abbreviations

---

ADC	Analog to Digital Converter
AES	Advanced Encryption Standard
AM	Active Message
AMI	Advanced Metering Infrastructure
AODV	Ad-hoc On demand Distance Vector
BER	Bit Error Rate
BS	Base Station
CC	Control Centre
CCMP	Cipher block Chaining Message authentication code Protocol
CM	Cloud Mesh
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
DHCP	Dynamic Host Configuration Protocol
DNGA	Dynamic Node to Gateway Association
ESSID	Extended Service Set Identifier
ETT	Expected Transmission Time
GUI	Graphical User Interface
GW	Gateway
ID	Identification
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
JDK	Java Development Kit
LAN	Local Area Network
LED	Light Emitting Diode
LQI	Link Quality Indicator
MAC	Media Access Control

## ABBREVIATIONS

nesC	Network Embedded Systems C
OPA	Ontario Power Authority
PAN	Personal Area Network
PC	Personal Computer
PER	Packet Error Rate
PLR	Packet Loss Rate
RSSI	Received Signal Strength Indicator
SN	Sensor Node
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
UDP	User Datagram Protocol
USB	Universal Serial Bus
WAN	Wide Area Network
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network

## Chapter 1

### Introduction

---

#### 1.1 – Motivation

Most people in today's world take for granted that electricity is available at our finger tips whenever we fire up our computers or turn on a light. It is constantly being consumed by the hot water heaters, air conditioners, and furnaces that make our lives more comfortable. Electricity is essential to manufacture the automobiles we drive, process the foods we eat, and power the entertainment we all enjoy. In the not so distant past, these were luxuries beyond imagination. In order to properly maintain and expand this precious power distribution system, it is necessary to understand and monitor the system's behaviour. Coupled with fast and reliable communication networks, fault situations can be readily reported to control centres for processing so immediate action can be taken. This way costs incurred to manufacturer's for stagnant assembly lines or health risks to hospital patients caused by lengthy power outages are minimized.

Now almost a full decade into the 21<sup>st</sup> century it is abundantly clear that alternative power sources to fossil fuels are crucial to meet the increasing power demands and decrease the effects of global warming. According to Statistics Canada [1], from 2004 to 2008 the Ontario population has grown by approximately 538,400 people, and Canada's population has grown by approximately 1,370,700 people. Additionally, North America is on the verge of mass production and distribution of electric vehicles. The introduction of millions of cars plugging in to recharge every day and the need to support an increasing population, will alone give rise to a massive electrical demand on Canada and Ontario's power transmission systems. The 2009 first-quarter report from the Ontario Power Authority (OPA) [2] illustrates that 6,778 MW of power is under development to meet this demand. This figure is in addition to the 3,801 MW that are currently in operation. Such an increase in power to be distributed would cripple a transmission

system that is not fit to receive it.

Ontario's first 110 kV transmission lines were installed and then operated in 1910, and the first 500 kV lines were under construction in 1960 [3, 4]. Since these times there have been many more kilometers of transmission lines put in place spanning across Ontario. The Ontario Energy Board [5] shows that in 2007 Hydro One had a total of 120,231 km of transmission lines. That is enough transmission lines in Ontario alone to circle the equatorial circumference of the earth 3 times! Of course, this massive distribution system does not always function flawlessly. Faults do occur and are a result of many different circumstances. Some faults are a consequence of bad weather such as tornadoes and ice storms, whereas others are caused by cranes, air planes, animals, and even fallen trees. Faults may also arise from contaminated and deteriorated insulators. Thus, conductors are exposed and short circuits may take place. Deterioration of insulators is mainly a product of age and overloading [6]. As previously stated, the first AC transmission lines are between 50 and 100 years old, and the amount of power Ontario is producing will almost double within 5 years. This could be a potentially hazardous situation without careful planning and monitoring of the power distribution grid.

## **1.2 – Problem Statement**

Essex Powerlines Corporation [7] is an electricity supplier in the area surrounding the City of Windsor. The University of Windsor collaborated with Essex Power to implement a transmission line monitoring system, and it is the task of this project to develop a prototype. The prototype is composed of two parts; the fault sensor, and the communication scheme to channel the data. The undertaking of this thesis is to design and create the communication scheme.

## **1.3 – Thesis Contribution**

In order to properly design an efficient monitoring system, it is necessary to



understand the limitations of the communication devices used in the system. Hence, a wireless testbed is created for the purpose of discovering the performance capabilities of IEEE 802.15.4. Performance trials are conducted in four different environments and the complete results are tabulated. Once a strong sense of the performance abilities are achieved, a self-configuring transmission line monitoring system is developed and a complete description of its functionality is given. Additionally, an IEEE 802.11 backbone network is developed to demonstrate the system feasibility. This backbone network includes a graphical user interface so an operator can monitor and interact with the sensing network.

## **1.4 – Background**

Since the establishment of the first three-phase alternating current (AC) power transmission system in 1893 by the Southern California Edison Company [6], this form of power transmission has gained much popularity. So much so, that three-phase AC is now used world wide as the preferred method for transmitting power from one place to another. In order to design an appropriate three-phase transmission line monitoring system, it is necessary to understand the difference between the transmission line's normal and abnormal operating parameters.

### **1.4.1 – Types of faults and Consequences**

For a three-phase transmission system there are four different types of faults that could occur. The first, and most popular, is the single line-to-ground fault. In this case one of the three conductors gets shorted to the ground. As a result, the shorted phase will experience a high current spike and the potential difference in the line will dramatically drop. The remaining two phases will both experience a sudden current decline (possibly to zero) and their voltage levels will change. Single line-to-ground faults make up 70%-80% of the faults that occur [6, 8]. Another type of fault is the double line-to-ground fault. The resulting current and voltage behaviour is similar to the single line-to-ground fault situation, except in this case two of the three phases are short circuited to the

ground.

The second most occurring fault type for a three-phase transmission system is a line-to-line fault. This occurs when one phase is short circuited with another phase. In such a case, these two phases experience equal current magnitude but it flows in opposite directions. Additionally, their voltage magnitudes are also equal, but they have different phase angles. This phenomenon causes the current to drop to zero in the remaining line that did not experience the short circuit.

The last type of fault is the least likely to take place, and does so only 5% of the time. It is known as the balanced three-phase fault. In this case, all three phases are shorted to the ground resulting in very high current flows and low voltage levels.

Determining the abnormal operating parameters is the responsibility of the power systems engineers. For the purpose of this thesis, it is enough to recognize that there will be both high and low thresholds for different circuit characteristics such as current and voltage, with normal operation occurring in between. The different combinations of exceeded thresholds for each phase will help establish which fault is present.

#### **1.4.2 – Smart Meters**

As Mankind pushes deeper into the Information Age, our goal for ubiquitous data access has not yet been attained. Nonetheless, one piece of technology that brings Man one step closer to that goal is smart meters. Smart meters are a digital meter to measure the amount of power being consumed at its final destination, such as commercial or residential buildings. They are an improvement upon the previous analog meters because of many additional features, but most importantly they are capable of wireless communication. Although several wireless communication technologies are possible, many smart meters implement IEEE 802.11.x WLAN as the preferred choice.

A neighbourhood of homes equipped with smart meters create a self organizing,

non-mobile, mesh network, as seen in Figure 1.1. Once every hour, each meter transmits the power consumption values it measured. This data gets forwarded through the network in a multi-hop fashion to a central collection node for the neighbourhood. At the end of the day, the collector sends all the information to the distribution company for bill processing. Under this scheme it is possible for the power company to have time-of-use pricing by charging different amounts for power consumption during different hours of the day. This is intended to motivate consumers to use less power at normally high peak times (lunch and dinner hours), and save high power demanding tasks for low peak times. An example would be to run dishwashers and clothes dryers during night time hours. As a result, the power generating and distribution companies will experience a more constant power demand during a 24h cycle instead of the fluctuating behaviour they experience now which introduces additional stresses to the system.

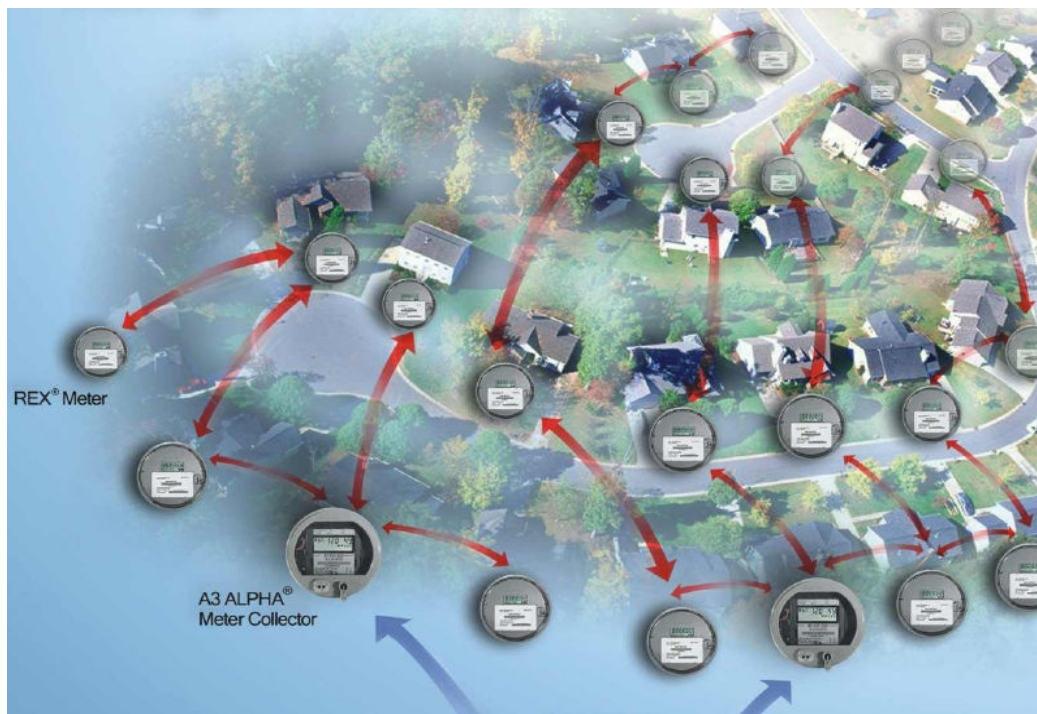


Figure 1.1: An example of a smart meter (REX Meter) residential deployment and its mesh network with the collector nodes [9].

The Ontario Energy Board put together a smart meter implementation plan in 2004 [10] that was submitted to the Ontario Ministry of Energy and Infrastructure. Details of this plan were then passed into legislation to make Ontario more energy efficient in the future. One of the details was to have smart meters installed on all

customers by the end of 2010 [10, 11]. This is significant to this project because it gives distribution companies a perfect network backbone to use for collecting transmission line fault information in populated areas. Since the WLAN mesh network and collecting device infrastructure would already be in place, the fault monitoring system would then only have to forward data by tapping into the existing smart meter network.

### **1.4.3 – Why WLAN and ZigBee**

The proposed transmission line monitoring system in this thesis uses the existing smart meter mesh network as an integral part of the communication scheme. Since the smart meters in the geographical area where the proposed system is intended to be implemented use WLAN technology, it is vital that the monitoring system be compatible with this technology. Nevertheless, many advantages of utilizing WLAN technology will be discussed in Chapter 2 and especially Chapter 3.

The IEEE 802.15.4 standard, otherwise known as ZigBee, is a Media Access Control (MAC) and physical layer standard specifically designed for short range wireless communication where low rate, low power and low bandwidth are required. This makes ZigBee an ideal choice when it comes to sensor networks for monitoring data collection and/or triggering process responses.

### **1.5 – Thesis Organization**

A description of the remaining parts of this thesis is as follows: Chapter 2 will discuss some related work in the field of sensor networks and transmission line monitoring, and Chapter 3 and Chapter 4 will analyze the performance capabilities of WLAN and ZigBee technologies, respectively. Chapter 4 will go in depth on the testbed design and results for determining ZigBee's performance in several different environments. Chapter 5 will thoroughly describe the designed system for a transmission monitoring network, followed by Chapter 6 with the project conclusion and ideas for future work.

## Chapter 2

### Related Work

---

For research to properly advance, it is important to understand what work has already been accomplished in that particular area. This chapter investigates research that is relevant to the design of different system components needed for a complete monitoring system. Section 2.1 focuses on the design of the sensors themselves and Section 2.2 discusses how to organize a team of sensors in a local area. Furthermore, Section 2.3 explores possible methods for long range transmission of collected data and Section 2.4 briefly touches on the security aspect of power monitoring systems. Lastly, alternative applications that utilize similar network characteristics as proposed in this thesis are discussed in Section 2.5.

#### 2.1 – Sensors

Focusing on the sensor design, Yang *et al.* in [12] addresses many design issues that are relevant to realizing a reliable and cost effective monitoring system. Some of these issues include types of wireless communication technologies, line segmentation and shunt impedance measuring, and integrating power, sensing, and communication functions. The wireless communication technologies suggested is ZigBee, for its low cost and low power consumption, and WLAN for its longer range, low cost, high throughput, high quality of service, and its security features. Measuring the shunt impedance of the line is important for detecting local disturbances, however this becomes difficult with interference from signal reflections in long transmission lines. Therefore a segmentation scheme is proposed. Additionally, it will be necessary for the sensor designers to make trade-offs with power management to operate sensing and communications while consuming minimal power.

Rodríguez and Tello in [13] show that current and voltage characteristics of a

transmission line are not the only pieces of data worth collecting. To further make use of a transmission line sensor network, temperature sensors and accelerometers may also be employed. These sensors help analyze the structural integrity of the lines and towers. The temperature sensors will pick up the high heat associated with a short circuit, and the accelerometers will indicate when there are high winds or a collapse.

## 2.2 – Sensor Management and Routing

One routing idea was to have a linear aligned network of sensors along a transmission line and they would forward their data to a collector node by multi-hopping the packets through the sensors. In [14] Gumbo and Muyingi studied this by comparing two situations for data collection. One situation was continuous querying where the sensors would constantly sample and transmit their data for the next hop to forward. The other situation was event-based querying where a sensor would not sample until it received a message to do so. The event-based querying scheme was more energy efficient, although the inevitable still occurred. Because of the nature of the sensor nodes' placement, each node forwards its data packets in addition to the ones it receives. As you can see, the nodes closest to the collector receive the highest amount of traffic and their batteries run out first. If they drop out of the network, then the link is broken and the collector cannot receive any more data from the remaining sensors. Although increasing the number of sensor nodes is important for increasing the utilization of a single collector, this multi-hop layout is hazardous to a power constrained network.

Maximizing utilization with many different kinds of sensors operating in proximity to a common collector node will require this node to execute some sort of management algorithm to function efficiently as a unit. Chen *et al.* in [15] use clustering to group certain nodes together and demonstrate that a two level communication model is more energy efficient than a single level communication model. In the two level model, one of the nodes in the cluster takes on the roll as the cluster head and any information in or out from the cluster passes through this node. Unfortunately this topology suffers from the same drawback as the routing scheme in [14] mentioned above where one

crucial link runs out of batteries first from processing all of the traffic. Two immediate solutions come to mind to alleviate this problem. The first is to have the sensor nodes exploit the advantages of solar cells (or other renewable energy sources) to recharge their batteries. The second solution could be to have each sensor node communicate directly with the collecting node destroying the two level communication model. In this thesis, the research takes advantage of this topology (single level communication model) to avoid short sensor node life spans, however a different communication scheme is used than was tested in this paper.

As with all kinds of sensing applications, there are many solutions to solve the same problem. Each solution has its own advantages and disadvantages. Transmission line monitoring sensing applications are no exception. In [16] Li *et al.* compare two different routing algorithms for organizing the ZigBee sensor nodes. One of the routing algorithms is a version of Ad-Hoc On demand Distance Vector (AODV) called AODVjr. AODVjr performs almost the same as AODV except all unnecessary aspects are removed and only the essential parts of the algorithm are used. It was found that AODVjr was more energy efficient, whereas the other algorithm, hierarchical routing algorithm, had shorter packet delivery latency. Which one is better to use in a sensor network depends on the power restrictions and the timing requirements for data delivery in a particular application.

### **2.3 – Long Range Data Transmission**

So far we have discussed some design issues associated with producing a feasible sensor, how to employ different types of sensors to increase the value of a collector, how to group these sensors into clusters for efficient management, and how to route data to a local collector from a network of sensors. A method for transporting the information in the collector to a distribution company's control centre is still needed.

There are many different communication topologies for monitoring transmission lines. Which is the most efficient and reliable way still has not yet been determined. It is

probable that different topologies will be more appropriate for certain situations. These situations may depend on whether it is an urban or rural location and the type of terrain and weather patterns. They may also depend on their proximity to other interfering technologies likely to be found in highly populated areas. Qing *et al.* in [17] propose that the collector act as a gateway between the sensor network and a cellular phone network for long range transmission to a control centre. This would probably be most useful for transmission lines in the county or on the outskirts of a city where too many nodes would be needed to multi-hop back to a control centre. The disadvantage would be that a cellular phone network radio subscription would need to be purchased for every sensor network gateway. This would become expensive, especially with the additional cost on monthly bills for bandwidth usage every time data is sent. Not to mention that cellular networks become overloaded with calls during destructive disasters when faults are likely to occur and may prevent fault data from being relayed to a control centre.

León *et al.* in [18] put forward a different idea for long range data transmission. This plan assumes that the gateway node for a cluster of sensor nodes is not power constrained. It involves each gateway transmitting its collected data to a gateway on the adjacent transmission line tower. Through multi-hop, data packets would eventually make their way to the transmission line's connecting control centre/substation. If a link is broken then the gateways forward their data packets in the opposite direction. This scheme might be more oriented for monitoring in populated areas or areas near a control centre. It would be unfeasible for monitoring a distant location where possibly hundreds of gateways would need to be installed just to channel the data to a substation. This thesis is also oriented toward transmission line monitoring in populated areas, by tapping into an already existing infrastructure.

The number of ways that sensor data can be transmitted for long distances is limited only by our imaginations. Marihart in [19] describes numerous technologies for this application and summarizes their advantages and disadvantages. These technologies range from twisted pair cables, power line carries, and optic fibre for wired solutions, to microwave radios and satellites for wireless solutions. Cole in [20] illustrates how



satellites can be used by power companies to collect the usage information directly from smart meters for near-real-time data collection. This topology would be especially useful for dwellings in remote areas. The disadvantage though, would be the cost associated with purchasing and launching several satellites into orbit, or paying for processing time on existing orbiting satellites.

#### **2.4 – Security**

Now that a complete data transmission topology has been discussed, it would be beneficial for distribution companies to have some sort of security for their wireless sensor network. Zhao and Villaseca in [21] researched just that. It was found that implementing the Byzantine Fault Tolerance mechanisms in local area networks (LAN) and wide area networks (WAN) it is feasible to provide secure control of the system, while still maintaining strict timing requirements for real-time applications.

#### **2.5 – Alternative Applications**

Transmission line fault monitoring is the focus of this thesis and the related work so far, however there are other monitoring applications that require the same network topology. Still based on transmission lines, Huang *et al.* in [22] use a similar sensor network as discussed for detecting ice build-up and de-icing of transmission lines. Changing paces, Lin *et al.* in [23] also propose a familiar sensor network for monitoring a water distribution system.

Sensor networks are becoming extremely important in many different areas of science. Applications for sensor networks in the areas of home automation, agriculture, automotive, and manufacturing are discussed in Section 4.2 on ZigBee performance. If one is so inclined, the reader will also be able to find sensor network applications in the medical field such as Montón *et al.* in [24] where yet again the same wireless communication hierarchy is suggested for patient monitoring.

## 2.6 – Summary

An understanding of some of the system components needed for a transmission line monitoring system and their current progress in research has been established. Many diverse sensors and an efficient communication scheme are necessary to optimize the network. Additionally, an appropriate technology must be selected for long range data transmission depending on the location of the planned sensor network. The fact that many researchers are working with these types of network components in alternative applications demonstrates their usefulness. The reliability of these systems that have a sensor network with a gateway to another communication technology is also important. Therefore, performance characteristics for WLAN and ZigBee are examined in the following two chapters, respectfully.

## Chapter 3

### WLAN Performance

---

A background explanation of IEEE 802.11 (WLAN) will not be given in this thesis. Thorough documentation of the WLAN protocol and its operational details are readily available on-line. Nevertheless, the reader is referred to [25] for the WLAN official standard.

#### 3.1 – Multi-hop

Multi-hop capabilities are crucial to realizing a wireless transmission line monitoring system. Many of the proposed systems discussed utilize multi-hopping to forward data across a network. This is beneficial because additional hardware such as cabling or another type of wireless node for longer range transmission is not needed. Therefore it is mandatory that WLAN is able to perform this task efficiently. Kim *et al.* in [26] assess the performance of three different MAC layer protocols with three different transmission rate adaptation schemes when using the expected transmission time (ETT) routing metric. Using simulations, they determined that if an appropriate rate adaptation scheme is linked with a MAC layer protocol then these nodes perform well in a multi-hop environment.

#### 3.2 – Long Range Data Transmission

With every different wireless technology, one big question that comes to mind is how far can it reliably transmit? In the event that WLAN technology is used in a multi-hop fashion for transmission line monitoring, it is necessary to determine how far a single hop can be. It is well known that IEEE 802.11b,g have an approximate range of 150m with no additional equipment or power boosting. Conversely, it has been documented that successful IEEE 802.11b transmissions of 201km have been made [27]! Surprisingly

no power boosting was used in this experiment, however directional antennas and a direct line of sight were necessary. It may be unlikely that a transmission line gateway single hop will need to be this long, but it is good to know in case the situation should arise. For long hops substantially shorter than 201km, a smaller directional antenna could be used to reduce the cost of hardware. Additionally, since transmission lines are fairly linear in nature, the line of sight requirement for long hops would not pose a problem.

Ireland *et al.* in [28] conducted research to determine how directional antenna orientation and spacing affect the throughput of a long distance multi-hop WLAN network. It was found that throughput decreased when adjacent hops were transmitting in the same directions using the same channel. If different channels were used for transmissions and/or adjacent transmissions were sent in different directions, then throughput improved. This illustrates that frequency reuse techniques would be beneficial for a multi-hop backbone topology. In the event that antenna orientation cannot be changed and two adjacent links must use the same channel, it was also demonstrated that introducing a vertical separation of just 4ft will also increase throughput.

### **3.3 – WLAN and ZigBee Co-existence Interference**

The proposed system in this thesis requires both ZigBee and WLAN technologies to operate in the same environment. Since both of these technologies can transmit on the same frequency ranges, interference and collisions are likely to happen. Shuaib *et al.* in [29] created a physical testbed to determine if either technology affected the throughput of the other. In the worst case scenario with both technologies transmitting on the same frequency channel, ZigBee did not reduce the throughput of the WLAN down link and throughput of the up link was marginally reduced. ZigBee throughput was affected more by WLAN signals because the transmitting power of WLAN is higher. When these technologies transmitted on separate channels, their throughputs were not affected by the other.

Diving a little bit further into the behaviour of ZigBee and WLAN interference, Yang and Yu in [30] determine how the packet error rate and packet loss ratios vary as the frequency distance between interfering channels change. As one would expect, the larger the channel separation, the less interference occurs, and fewer packets are corrupted or lost. They also found that ZigBee experienced less interference by IEEE 802.11g than was experienced by IEEE 802.11b. This data is useful when determining which IEEE 802.11 standard should be used when deploying a wireless monitoring system such as in this thesis.

### **3.4 – Low Power Access Point**

In many papers, including this thesis, the assumption has been made that the gateway node to a wireless sensor network on a transmission line is not power constrained. That is, that the power source for these gateway devices would be the transmission lines themselves. Whether this is the case or not, it would still be ideal for the gateway nodes to consume minimal power. Zhang *et al.* in [31] provide three different frame layout schemes for power saving multi-hop access points. Different schemes performed better than others depending on the traffic loads and other network operations. In summary, low power access points are possible while still maintaining acceptable delays and system performance. Additionally in a related paper, Farbod and Todd [32] demonstrate that by implementing power saving procedures, a substantial reduction in access point resources can be realized.

### **3.5 – Security**

In order to prevent illegal tampering with transmission line data, the wireless data transmissions should be encrypted. On the other hand, this encryption should not degrade the system performance so that timely reporting of faults cannot be achieved. Siwamogsatham *et al.* in [33] compare four different levels of increasing encryption for a secure WLAN transmission and the effect that it has on the system throughput. The first is the control experiment where throughput is tested with no encryption techniques

implemented. Secondly, the test is repeated with wired equivalent privacy (WEP) encryption utilized. Next, temporal key integrity protocol (TKIP) and lastly cipher block chaining message authentication code protocol (CCMP) are also tested. CCMP is based on the advanced encryption standard (AES) algorithm which has been deemed suitable to protect classified information according to the US government. In all cases, a negligible throughput performance hit is experienced regardless of the MAC payload size, if user datagram protocol (UDP) or transmission control protocol (TCP) is used, and whether the system is operating in infrastructure or peer-to-peer mode.

#### **3.6 – Summary**

The above research demonstrates that WLAN technology possesses the necessary characteristics to be used for a dependable transmission line monitoring system. Its ability to perform multi-hopping on sort or very long distances, to be resistant to interference from ZigBee transmissions, to be used in low power access points, and to have very secure wireless transmissions are all attributes that reflect the robustness and flexibility of this technology. In order to have system wide reliability, then both wireless technologies need to be robust. In the next chapter the wireless performance of IEEE 802.15.4 (ZigBee) is discussed.

## Chapter 4

### ZigBee Performance

---

A background explanation of IEEE 802.15.4 (ZigBee) will not be given in this thesis. Thorough documentation of the ZigBee protocol and its operational details are readily available on-line. Nevertheless, the reader is referred to [34] for the ZigBee official standard.

#### 4.1 – Introduction

IEEE 802.15.4, commonly known as ZigBee, is a Media Access Control (MAC) and physical layer standard specifically designed for short range wireless communication where low rate, low power and low bandwidth are required. This makes ZigBee an ideal choice when it comes to sensor networks for monitoring data collection and/or triggering process responses. However, these very characteristics bring into question ZigBee's ability to perform reliably in harsh environments. This chapter thoroughly explains the experimental testbed setup and execution to demonstrate ZigBee's performance in several practical applications. This testbed is capable of measuring the minimum, maximum, and average received signal strength indicator (RSSI), bit error rate (BER), packet error rate (PER), packet loss rate (PLR), and the bit error locations.

As digital technology is rapidly advancing in the 21<sup>st</sup> century, much of this technology is oriented toward efficiently monitoring and reacting accordingly. Whether it is monitoring for building automation, assembly line manufacturing, or even National Security, sensor networks play a crucial role. There are several mediums in which to construct sensor networks with each having their own strengths for certain applications. IEEE 802.15.4 (ZigBee) is a leading technology for wireless short-range sensor networks. In order to discover the full potential of ZigBee devices, it is necessary to challenge them

in as many diverse applications as possible. In order to do this a reliable and efficient testbed is necessary. Such a testbed can be used to discover physical layer performance boundaries to increase utilization of ZigBee networks. The goal of this chapter is to thoroughly describe a testbed design and gather statistics describing ZigBee's physical layer reliability.

There are studies regarding ZigBee's performance based on theory and simulations such as [35, 36]. Hameed *et al.* in [35] put forward a scheduling scheme for guaranteed time slots for real-time applications, and in [36] Zeghdoud *et al.* obtained optimal throughput for different clear channel assessment modes in the presence of IEEE 802.11 interference. On the other hand, studies that examine transmission reliability for off the shelf ZigBee devices are scarce. Ilyas and Radha in [37] is one such study that investigated the error process in IEEE 802.15.4 devices for indoor and outdoor environments. Using transmission data, they collected and modelled the channel using the bit error rate (BER) probability density function and correlation coefficient. Industry is interested in the performance of ZigBee in different applications, such as in vehicles and in industrial settings like [38, 39] by General Motors, and General Electric and Sensicast Systems, respectively. These studies combined with this chapter's experimental results for several environments will give researchers an excellent foundation for ZigBee's ability to optimally perform in many real-time applications.

Home automation is gaining popularity with appliances like dishwashers, washing machines, fridges, furnaces, hot water heaters, and any other device that could be used to form a single home network. These appliances can be controlled to operate at ideal times of the day to minimize energy costs and maximize usage with smart meter technology. Reinisch *et al.* in [40] demonstrated that ZigBee is the most appropriate communication technology for home automation and Kim *et al.* in [41] put forward a scheduling scheme for frames and sub-frames in order to acquire optimal network parameters.

One possible outdoor applications of ZigBee is environmental monitoring, which would be beneficial to scientists and the agricultural industry. ZigBee would provide the



ability to network a wide range of sensors which detect soil and air moisture, the richness of the soil, temperature, solar radiation, wind speed and direction, and atmospheric pressure. This data can then be used to predict weather patterns, or determine optimal times to dispense water or other nutrients to plants. Siuli Roy and Bandyopadhyay in [42] provided a ZigBee network where soil properties are sensed for real-time monitoring. A testbed that determines outdoor channel measurements would help these applications flourish.

The idea of wireless communication within a vehicle is gaining interest for many reasons. Primarily, it results in much faster installation times, by cutting the need for wiring many components together from all corners of the vehicle, and greatly reducing the weight of the vehicle by eliminating the need to install up to several kilometers of cables. Ahmed *et al.* in [38] state issues as to why ZigBee technology is not yet ready for automotive applications that do not include transmission error reliability. Two main reasons are that ZigBee does not necessarily meet timing requirements depending on the sensor (shown in the popular NS-2 simulator), and these devices are still too expensive to be offset by the savings in cable costs. Although these issues are out of the scope of this chapter, the designed testbed is used to verify ZigBee's communication capabilities in vehicle environments.

Many industry solutions are now going wireless in an attempt to cut costs. It is not uncommon for data cables, which are connected to sensors on robotic arms or other mobile parts, to snap. The down time to repair and replace these cables creates an unnecessary cost to manufacturers. Additionally, the installation time for a wireless solution is much faster. A machine shop would be an accurate representation for a manufacturing facility. The University of Windsor's machine shop was used to represent this environment. Tang *et al.* in [43] is an example of a ZigBee wireless channel investigation for a similar environment. Their analysis is primarily based on RSSI and the link quality indicator (LQI). The packet error rate (PER) is calculated by assuming if a packet did not arrive then it was in error. This is not necessarily the case. It is possible that a packet could not arrive simply because the ZigBee radios are out of range, or there

is a physical obstruction blocking the transmission. Furthermore, many of the papers published in this area focus on the latency issues of wireless networks and do not address error rates.

#### 4.2 – Testbed Components and Structure

There are two types of nodes in the designed testbed. The first node is referred to as the base station (BS), which encompasses a ZigBee mote on a Crossbow MIB510 [44] programming board connected to a laptop. This connection is made via an RS-232 to USB cable. The second node is simply the transmitter, which is a stand alone ZigBee mote. The ZigBee motes that are used are the Crossbow MicaZ mote, which utilize the Chipcon CC2420 radio. The TinyOS-2.x environment [45] is used to program the MicaZ devices and they transmit data on channel 26 with a maximum transmission power of 0dBm (1mW). The BS laptop communicates with the serial port (and therefore, the ZigBee programming board and mote) using a Java program during the experiment execution. This program is referred to as *BaseStation.java*. Furthermore, the laptop has Java Development Kit (JDK) 6 installed, and this runs in an open source Mandriva Linux 2008 environment.

Once the BS and transmitter nodes are in place and turned on, the test begins by running *BaseStation.java* as a console command. Three of the input arguments include: the node ID of the transmitter node that is asked to send the data packets, the number of data packets the transmitter is to send in return, and the packet transmission rate (how many of these packets are to be sent within one second.) Passing these arguments to *BaseStation.java* increases the flexibility of the testbed and allows the parameters of each trial to be changed. It also allows consecutive trials to be conducted without the need to turn the transmitter node off and then back on. This feature is helpful when the transmitter is in a location that is difficult to reach (E.g. under the hood of a car while driving.) Once executed, *BaseStation.java* builds a proper TinyOS Active Message (AM) packet containing this information and sends it to the serial port connected to the programming board. Figure 4.1 illustrates both the control packet and data packet

structures. The ZigBee mote part of the BS simply transmits from the radio interface whatever is received on the serial interface. Figure 4.2 offers a graphical representation of the testbed structure. The computer domain contains the laptop hardware and software. Figure 4.3 shows the packet transmission sequence during the trials.

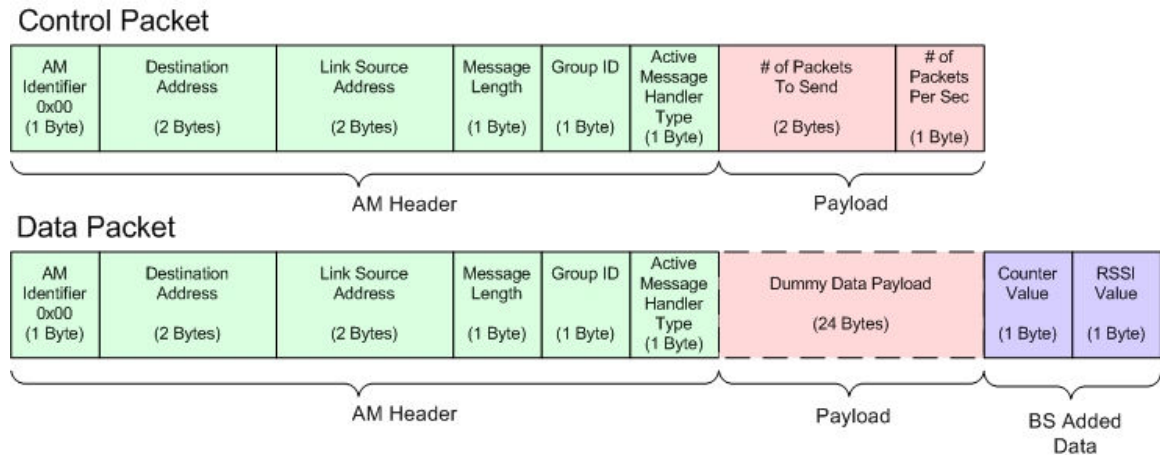


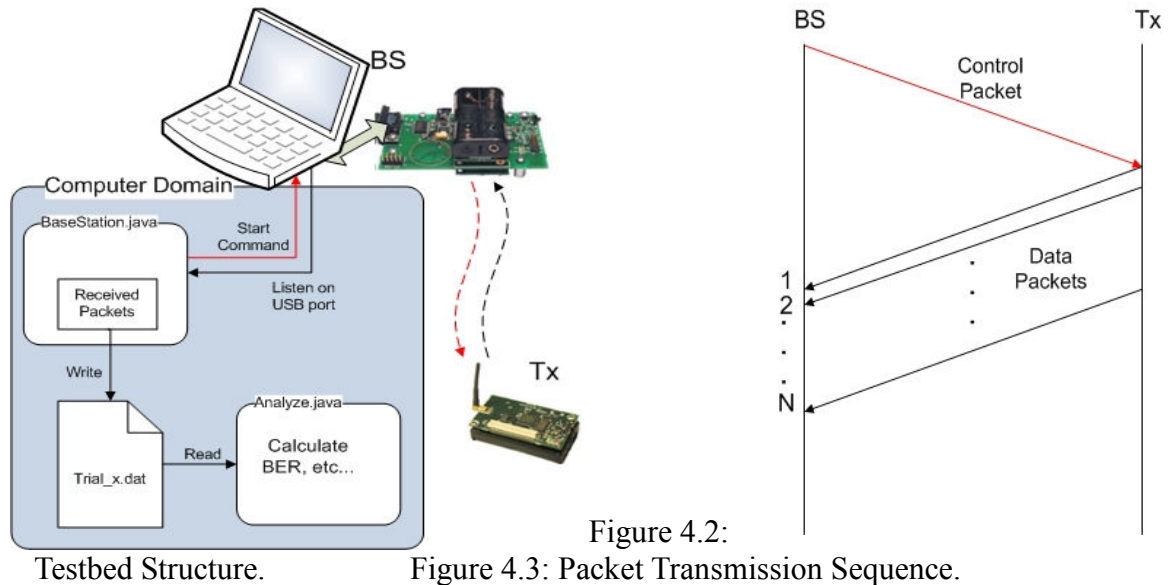
Figure 4.1: Packet Structures for Control and Data Packets.

In order for the BS to calculate the number of errors caused by the wireless channel, the transmitter node always transmits the same data packet. The first 8 octets are AM header information and the dummy data payload is decimal number 85 which was chosen simply because it is alternating 0's and 1's in binary. The total packet length is 32 octets.

Upon receiving these data packets, the BS mote adds two additional octets of information on to the end of the packet before it forwards them through the serial port to the laptop. The first octet is a counter value, which will be discussed later, and the second octet is the radio's calculation of the RSSI for that particular packet. All of these packets are picked up by *BaseStation.java* listening on the USB port and it saves each consecutive packet in a file for future analysis. A new file is created for each trial.

A second Java program (*Analyze.java*) was developed to analyze the saved files containing the received packets. Since the transmitted data is known, this program can easily calculate the BER and PER for all received packets in each trial. In addition, it

determines the bit error locations, the number of packets received, and the maximum, minimum, and average RSSI values for every trial. A conversion chart for CC2420's RSSI values to dBm is given in [46] at page 49.



Testbed Structure.

Figure 4.3: Packet Transmission Sequence.

By default, the ZigBee mote radio chips conduct a cyclic redundancy check (CRC) on each packet. Packets that do not pass the CRC are immediately dropped by the CC2420 radio and would not be available for analysis. This poses a problem when there is a need to calculate BER and even PER, and creates ambiguity because it is not known whether the packet had an error or was lost. Also, the BER would be impossible to determine when erroneous packets are dropped after CRC. In order to circumvent this, some modifications to the TinyOS driver files for the radio chip were made in order to allow not only the correct packets through, but also the packets that have errors. Table 5.4 describes the modifications made to the specific driver files.

The BS ZigBee mote appends a counter value to the end of the incoming packets. Since this mote has been modified to allow error packets through, occasionally only partial packets will be received during poor channel conditions. Sizes of these partial packets vary, which would make for an unnecessarily complicated *Analyze.java* program to deal with them correctly. Instead, simply the occurrences of partial packets are counted and such packets are discarded. Consequently, accepted packets do not have

errors in the length field of the packet header. Partial packets are included in the PER calculation.

### **4.3 – Experiment Procedure and Results**

In this section, the experimental procedures and results will be provided, along with a clear representation of the test sites and transmitter locations. The indoor, outdoor, vehicle, and machine shop test sites were chosen because they cover the majority of the application environments. The numerical results for all trials are shown in Table C.1. This includes the approximate distance between the nodes, the BER, PER, PLR, the maximum, minimum, and average RSSI, and the number of partial packets received.

#### **4.3.1 – Indoor**

The house in which the indoor trials were conducted was a 12.5m x 8.7m two-story home with a basement. Figures 4.4a) and 4.4b) are the layouts of the first floor and basement, respectively. Trials were done with the BS located in three different areas. First, the BS was located in the kitchen (main floor) while the transmitter was positioned in several key locations around the house. Second, the BS was placed in front of the fuse box (basement) while the same key locations were tested. In the final trial the BS was positioned on the front control unit of the furnace while the transmitter was placed one floor above on the thermostat. Additionally, there was no movement of residents in the house during test execution and each location/trial called for 1000 packets to be transmitted at a rate of 5 packets per second.

#### 4. ZIGBEE PERFORMANCE



Figure 4.4a: Main Floor Layout.

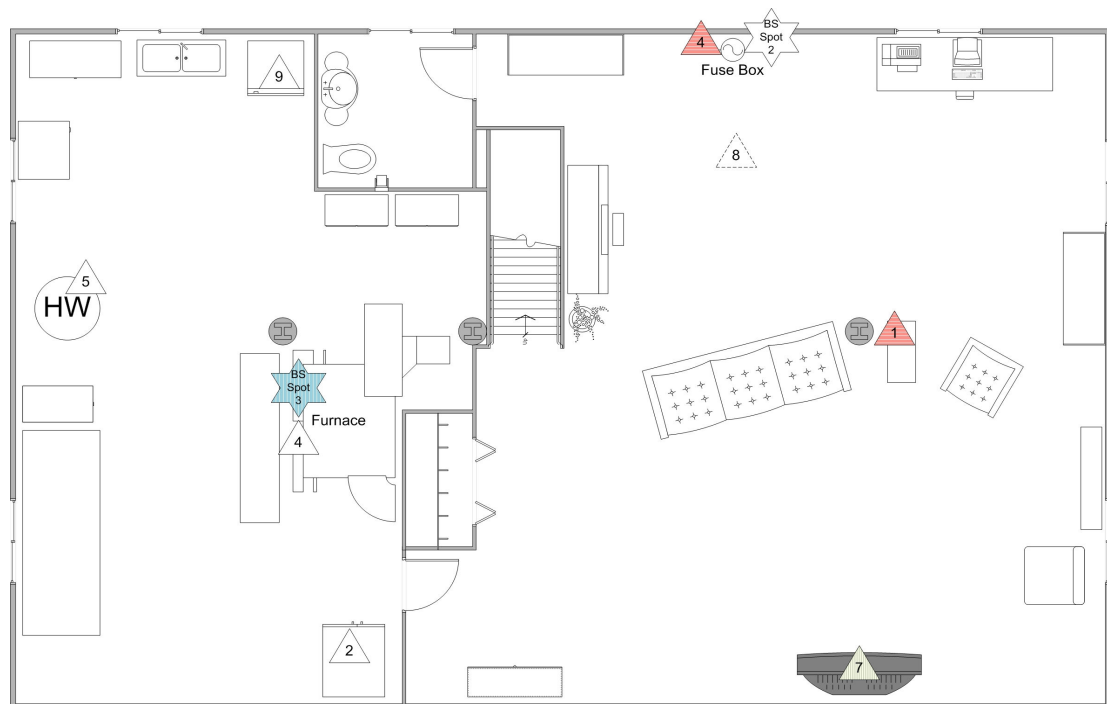


Figure 4.4b: Basement Layout. The triangles represent the transmitter test locations. The horizontal stripe is for trials conducted with the BS in the kitchen. White is for trials with the BS at the fuse box. The vertical stripe is for the trial with the BS at the furnace.

The results were very good when the BS was located in the kitchen while the transmitter was placed at either the various kitchen appliances, the electricity (hydro) meter directly outside of the kitchen wall, or one floor below. However, reception was either extremely poor or non-existent for certain key areas such as the hot water heater and furnace in the basement. In these cases the direct transmission path was impeded by several walls and appliances such as a refrigerator, stove, or furnace.

When the BS was located in the basement at the fuse box, the results were much better. There was reception from all key locations and this proved to be a much more ideal location for a BS. For the final indoor test the BS and transmitter were not located very far apart, even though they were on separate floors. The transmission was reliable with less than a 1% PER and 0.09% BER demonstrating that a wireless connection between furnace and thermostat is viable.

Since no two indoor environments are the same, it is difficult to formulate precise conclusions. Nevertheless, these results give a good indication on how ZigBee may perform in a home automation system. Although performance greatly depends on the indoor layout and node locations it may be a good idea to have a BS for every floor in a home, or one for every 7m radius.

#### **4.3.2 – Outdoor**

The outdoor tests were conducted in two different, large, open fields. The results for both were very similar. The transmitter node was placed on a tripod so that its antenna was 1.5m above the ground. The BS's receiving antenna was 1.15m above the ground and the laptop was positioned behind and below it to minimize interfere. For each trial 1000 packets were transmitted at 5 packets per second. It was mostly sunny and there was no precipitation for these tests.

As can be seen, the error rates start to significantly climb once the node separation

is beyond a 70m distance. Strangely, the devices experienced a poor communication region between 20m and 30m. However, when the height of the transmitter was changed, reception greatly improved. This may have been caused by multipaths destructively interfering given the original height of the antennas and the distance between them. Furthermore, the reception at 90m was more reliable than at 80m and 85m. This could be attributed to small scale fading as described in the 20m to 30m fading region. Although extremely poor, there was still reception at 95m, but no reception at 100m. This test showed that a distance of 70m appears to be a reliable range if the transmitter is at least 1m above the ground with no obstructions.

### 4.3.3 – Automotive Internal Monitoring

For this test a 1994 Toyota Corolla was used. The BS mote receiver was placed in the closed glove box closest to the centre of the car. In the event that ZigBee technology is used in this environment, we hypothesized that the master node would be located somewhere in the front dashboard. Also, only one person was present in the car during the trials and was sitting in the driver's seat. The transmitter node was placed at twelve key locations around the car including under the hood, in the trunk, and in the passenger cabin as shown in Figure 4.5. All twelve trials were conducted both when the engine was on and off, but always in park. Each trial had 1000 packets transmitted at 5 packets per second. Although the state of the engine had little effect on the performance, errors only occurred when the engine was running. Generally, all packets were received error free in almost all trials. The biggest interferer seemed to be the human body if it was located in between the transmitter and receiver nodes.

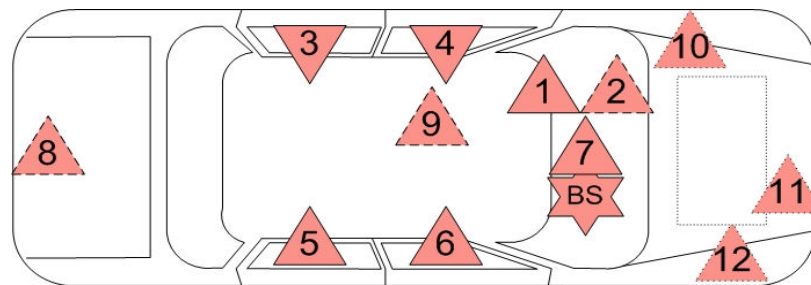


Figure 4.5: Test Vehicle and Transmission Locations.



#### 4. ZIGBEE PERFORMANCE

In addition to the above automotive tests, trials were conducted with the transmitter under the hood (at position 11) while driving on the expressway and through the city. The city driving trials were typical 15 minute drives (4500 packets at 5 packets per second) while zigzagging across the city. There were plenty of stops, turns, and straight runs. The speed of the car ranged from 0km/h to 65km/h. The wireless transmissions performed the best when the car was either stopped or moving at an approximate constant velocity. The **majority** of the bit errors were observed to occur during acceleration. This is **not** to say that they occurred during **all** accelerations, **nor** did they only occur during acceleration.

The expressway test was interesting in that on some trials almost all packets were transmitted perfectly, and on others trials, packet transmission was not so impressive. It is possible that the wind from the high speeds altered the antenna position when driving in one direction and not the other. Also, a portion of the expressway where the trials were conducted is adjacent to an airport, so radio frequency interference could be a possibility at times. The speed of the car during the expressway trials ranged from 100km/h to 120km/h and trials had either 1500 or 1200 packets transmitted at 5 packets per second. The communication performance in the car was much better than expected particularly for the expressway and city driving tests with the worst case scenario only having a PER of 4.8%.

#### 4.3.4 – Machine Shop Floor

Trials were conducted in a similar fashion as in the indoor test. Two separate locations were used for the BS while several key spots were chosen to place the transmitter. Figure 4.6 shows the layout of the machine shop. Shop workers were present and were free to move around during testing. Shop machines were on and off as the workers proceeded with their normal daily work schedule.



Figure 4.6: Machine Shop Layout.

When the BS was at the first location there are two noteworthy points to make. The first is that when the transmitter was on the CNC lathe machine (position 3), the test was conducted twice, once with the machine off and then once with it running. The running lathe machine had virtually no effect on the results from the first trial. The second noteworthy point is the drastically improved reception at position 2 when the transmitter was placed high on the ceiling lights compared to at shoulder height.

The second BS position in the middle of the shop floor had much better reception results overall, since it was closer to most of the transmitter locations. There was not any reception with the transmitter at position 6, since it was behind a thick concrete wall. Also, there was poor reception at position 7, which may be attributed to the fact that a worker was standing directly in the transmission path while operating one of the machines. The machine shop illustrated that a BS for every 10m radius without major obstructions would be appropriate.

#### 4.4 – Summary

A flexible testbed that is capable of determining many performance measurements has been developed, and a detailed description of its structure and operation is given. This testbed was used to test ZigBee's natural communication capabilities in four different practical environments without any additional techniques to improve reception. From these tests, some notable observations can be made. Firstly, none of the environments tested extremely hindered the communication of the MicaZ motes. Based on these results, it is reasonable to believe that these devices are capable of operating in similar conditions, and that they will be even more reliable as technology advances. Secondly, the human body appears to reduce the RSSI value for packets more than walls or machines. Bodies can cause the RSSI to drop by 20 to 30 units. Consequently, performance of the wireless connections greatly depends on the transmitter and receiver locations. As discovered in the machine shop, higher installation locations are better in order to avoid obstructions from machines and workers. It was also noticed in the outdoor test that the closer the transmitter was to the ground, the shorter the communication range available.

Unfortunately a ZigBee performance test was not able to be done with a transmitter node attached to a transmission line, however some of the results from these tests will give some clues as to how they would perform. Since the transmitter nodes would be up on the transmission lines, they should have an increased broadcast range with no humans or machines as obstructions. To help uncover the answer as to if the electromagnetic field from the transmission line would disrupt the wireless signals from the transmitter nodes, a few key results are highlighted. When the transmitter was placed in the extremely noisy environment next to the car engine, the PER never surpassed 4.8%. This is true even for the expressway driving trials. Electromagnetic waves would be generated from the engine spark plugs and possibly from other electrical systems under the hood. Additionally, the PER was always below 7% in the machine shop within a 10m radius. The machine shop was a harsh environment because of the numerous obstacles and running machines. These environments are thought to be more unforgiving

than a transmission line deployment.

Now that a strong understanding of the communication capabilities of both WLAN and ZigBee technology has been established, a transmission line monitoring system utilizing these technologies can be implemented. The following chapter thoroughly explains the designed system carried out for this project.

## Chapter 5

### Transmission Line Monitoring System

#### 5.1 – Overall Proposed Solution

Smart meters are currently being deployed in many places across Ontario and some are already being utilized in the United States, Europe, Canada, and other countries. These meters are the first major step in the fight against global warming by intelligently monitoring power consumption for individual subscribers. Once fully deployed, smart meters will be placed on all residential buildings, offices, businesses, manufacturing plants, and any location where electricity is consumed. The system which measures, transmits, collects, and processes power consumption data is known as the advanced metering infrastructure (AMI). There are already many companies which specialize in designing, manufacturing, programming, deploying, and/or operating devices which are used in the AMI. AMI technology is a proven technology for reliability, security, and scalability. In addition, within large cities there could be hundreds of kilometers of transmission lines which are adjacent to smart meter equipped buildings. This makes it ideal to be used for a transmission line monitoring system backbone for populated areas.

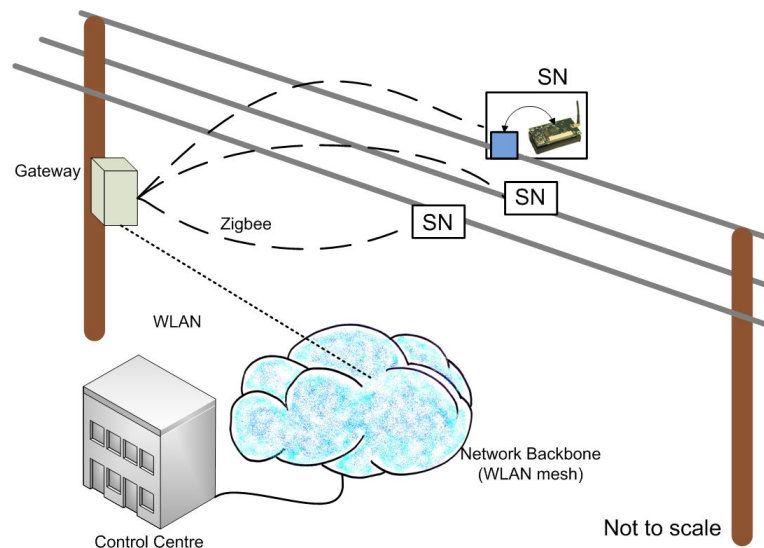


Figure 5.1: Overall System

Since smart meters utilize IEEE 802.11 cards as their primary communication radio, it will be necessary to develop a gateway device that can collect transmission line data and forward it into the smart meter mesh network for data aggregation. This project covers these lower two tiers of the larger system for transmission line monitoring. That is, sending data from the sensor nodes (SN) to the gateway (GW), and then from the gateway to the smart meter mesh network. Figure 5.1 gives a graphical representation of the overall system. The cloud depicted in this figure is representative of any neighbourhood WLAN smart meter mesh network (as was shown in Figure 1.1). Since actual smart meters are not available to be used for testing purposes, this network is replaced with a single PC, WLAN hop. Additionally, the solid square in the SN is representative of the actual sensor to detect current flow in case of a fault situation. Since this sensor has not yet been developed, a stand-in sensor board is used. See Figure 5.2 for the resulting illustration of the overall system.

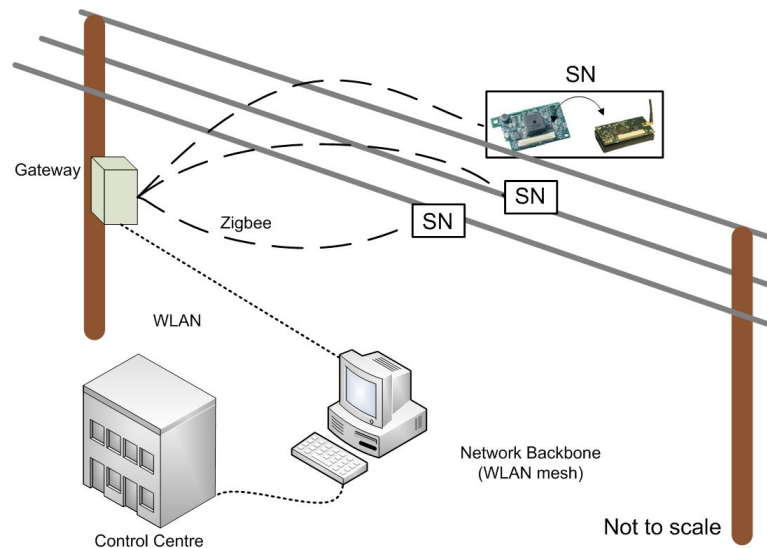


Figure 5.2: Overall Test System

## 5.2 – Hardware and Software

Much of the tools used are as described in the first paragraph of the ZigBee performance, Section 4.3 Testbed Components and Structure. Nevertheless, it will be reiterated here to take into account the additional nodes in the system.

There are four types of nodes in the implemented system. The first node is referred to as the GW, which encompasses a ZigBee mote on a Crossbow MIB510 [44] programming board connected to a laptop. This connection is made via an RS-232 to USB cable. The second node is the SN, which is a stand alone ZigBee mote with a sensor board. The ZigBee motes that are used are the Crossbow MicaZ mote, which utilize the Chipcon CC2420 radio. The Crossbow MTS300CA is the sensor board that is used, which contains a microphone, buzzer, light sensor, and temperature sensor. The TinyOS-2.x environment [45] is used to program the MicaZ devices and they transmit data on channel 26 with a maximum transmission power of 0dBm (1mW). The GW laptop communicates with the serial port (and therefore, the ZigBee programming board and mote) using a Java program during execution. This program is referred to as *Gateway.java*. Furthermore, the laptop has Java Development Kit (JDK) 6 installed, and this runs in an open source Ubuntu Linux 8.10 environment.

As for the other two types of nodes, they each consist of a fit-PC 1.0 which runs their respective programs written in Java. They also have JDK 6 installed and also have Ubuntu Linux 8.10 as the operating system. One of the nodes is the one representing the cloud network which runs the program *CloudMesh.java* (CM), and the other is the control centre (CC) end node which runs *ControlTerminal.java*. A graphical user interface (GUI) was created for the control centre node so the user could easily operate the controls without needing to memorize command-line functions. The GUI was created using NetBeans IDE 6.1.

For the three nodes that utilize the WLAN interface (GW, CM, and CC) the Linksys WUSB54GC was used for the WLAN USB. It was not necessary to download and install a driver for these USB's because Ubuntu already has the support to operate them.

### 5.3 – System Requirements

For this prototype to be a successful deployable system, it must be capable of certain tasks. Work crews should not be responsible for configuring and programming each sensor node before installation. This would require additional training, and inevitably, network misconfigurations due to human error would still occur. Therefore, the sensor nodes should be self configuring so all that is required is to put them in place and turn them on.

It would also be beneficial for users in the control centre to be able to remotely manipulate the SNs. Three basic commands that would be useful are enabling the SN to go into sleep mode, wake up from sleep mode, and being able to probe it for its sensor data at any time. These three commands would greatly increase the functionality of the system.

The sleep command is valuable for when maintenance is already being conducted on the transmission line. During the maintenance procedure it is possible that power will cut in and out, and it is not needed to have the SN reporting this since a work crew is already present. Therefore, prior to the maintenance procedure the CC can put those SNs into sleep mode so they can conserve their power. Also, in the event that repairs are ahead of schedule or the SN was put to sleep for too long, the CC should be able to wake them up at any time.

During the regular sensing cycle for a SN, it may be programmed to sample its sensors every 10 minutes. Once a fault is detected however, it would be advantageous for this cycle to repeat every 5 seconds so more data about the situation can be logged. For this to be accomplished, the SNs will have to be intelligent enough to know when a fault is detected and be able to dynamically change its cycle period accordingly.

Lastly, it is also beneficial for the GW to not instantly forward data packets as soon as they arrive with fault data. Elaboration of this requirement will be discussed in



Section 5.7 describing the GW's functionality.

The explanation to follow will describe the functionality of each node individually starting with the control centre and then moving down into the network.

## **5.4 – Control Centre (CC) Node**

### **5.4.1 – Functionality**

*ControlTerminal.java* is a double threaded program. One thread is responsible for listening for incoming packets, where the duty of the other is to construct and send command packets when desired. Breaking up responsibilities into two separate threads allows for longer listening periods than if one thread has to do all the processing. The listening thread simply opens a socket during initialization and waits for data packets to arrive. Upon receiving a data packet, its information is displayed on the screen and the thread returns to the listening state. A separate function (named *affix*) had to be written in order for the text to be displayed because the GUI does not refresh until it has completed the function it was in. If printing to the screen was done in the running thread function, nothing would be displayed since the thread constantly loops around waiting for new packets. Once started, this thread continues to run until the program is closed.

When the sending thread is activated, the program captures the command type and the integers from the text fields. A datagram packet is then constructed using this data as its payload and the packet is then forward into the network to the next hop toward the ZigBee GW. The sending socket is then closed and this thread terminates until it is needed again to send a new command.

### **5.4.2 – Graphical User Interface (GUI)**

The GUI program for operating the CC node is a single windowed application as depicted in Figure 5.3. The button in the top right corner of the window labelled “Start

## 5. TRANSMISSION LINE MONITORING SYSTEM

Receiving” will start the thread that opens the WLAN socket for listening. This button only needs to be triggered one time when *ControlTerminal.java* is started. In fact, once it is pressed, the button is disabled so it cannot be pressed again and another thread cannot be triggered. The top portion of the window is the display area where received data packets are printed. If the “Start Receiving” button has not been activated, then received data packets will not appear on the screen. In the event that data is received and the values indicate a fault has occurred, these particular values will be displayed in red text, whereas everything else is displayed in black text. Notice the red fault values that are below the low operating threshold (700) and above the high operating threshold (990). In addition, if a data value indicates that a SN is running low on battery power, this value will also be displayed in red text. Therefore the user should know far in advance if a SN will need a battery change.

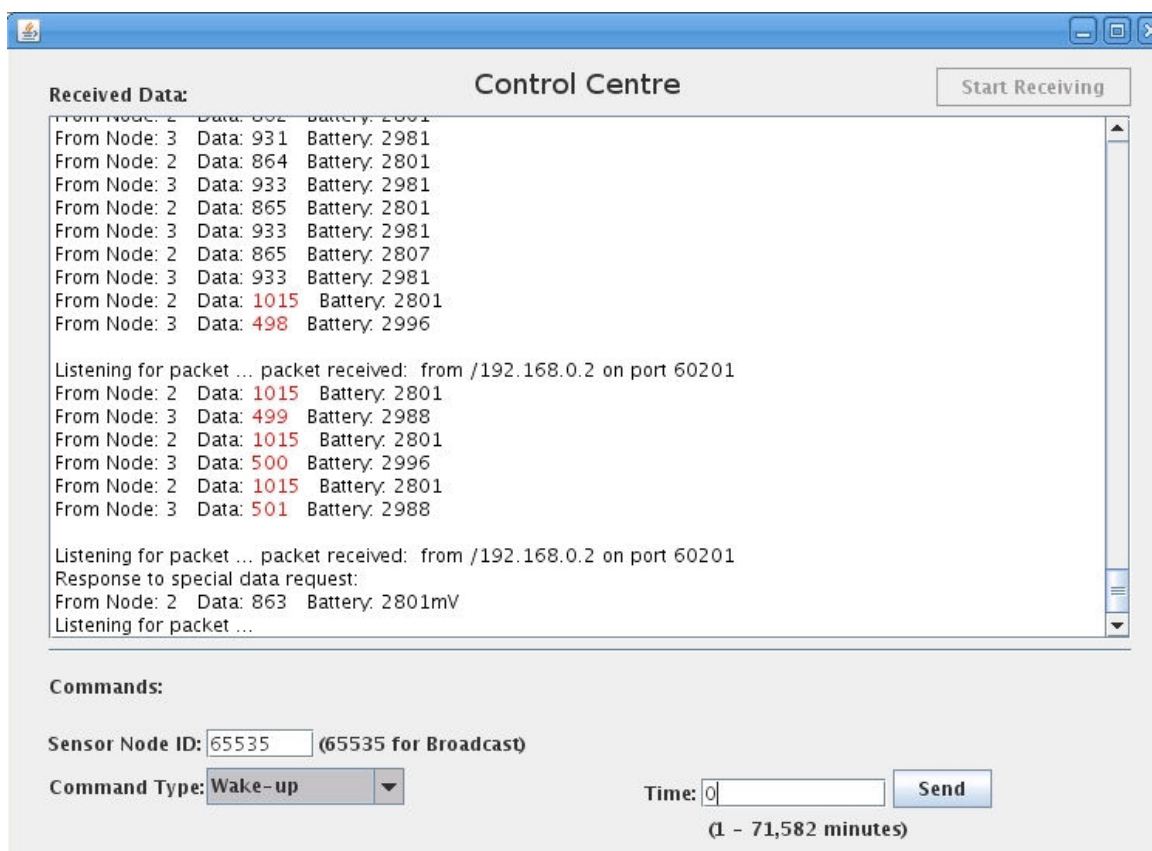


Figure 5.3: CC's GUI Sample Screen Shot.

The bottom portion of the CC's GUI window is for sending commands to one or

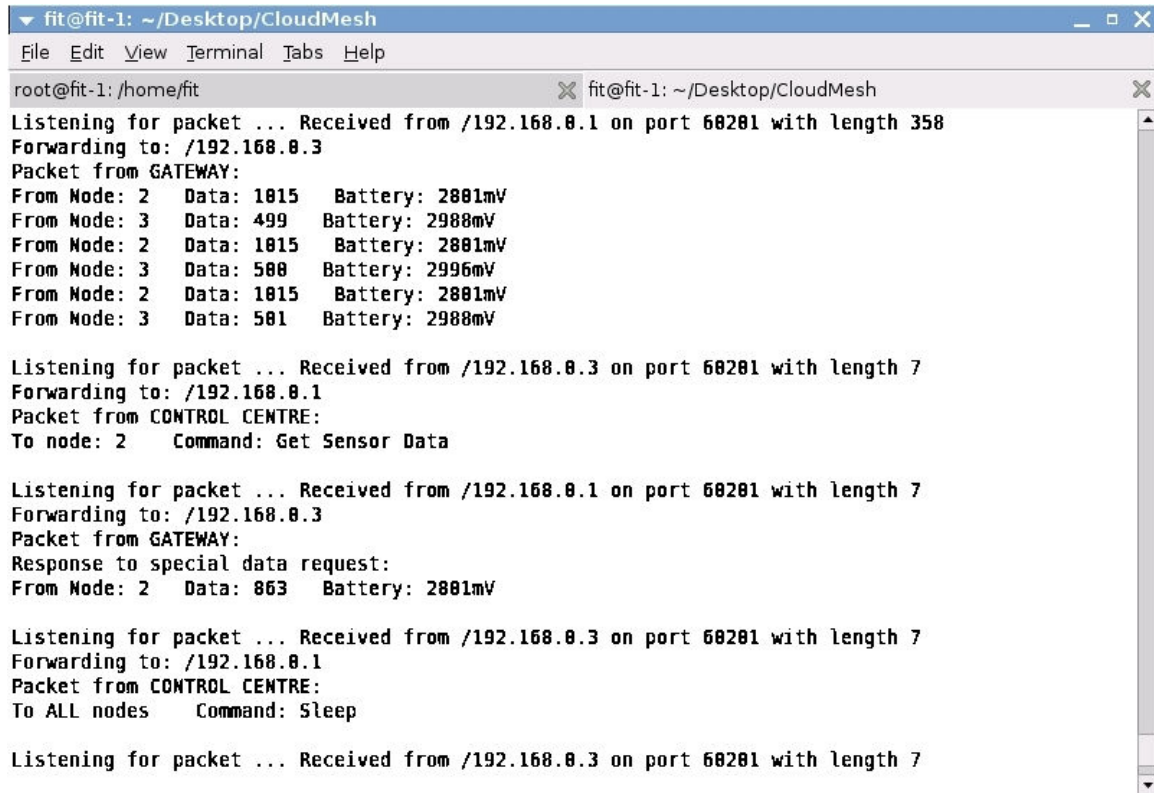
all sensor nodes. In order to send a command, the user must fill in the two data fields and select a command type prior to pressing the “Send” button. The type of command to send can be selected from the list in the pull-down menu. This allows for complete protection from users entering erroneous command types. The SN ID number in which the command is desired to go must be entered into its field. If the command is directed to all nodes that are associated with the GW, the decimal number 65535 is to be entered. This is equivalent to two bytes of all 1's, which is the broadcast address. If a command is sent with a SN ID that nobody has been assigned, nothing will happen. This packet will simply not be picked up by any node that receives it. Lastly, the number of minutes for inactivity must be entered only if the sleep command is being issued. The “Send” button triggers a new thread to send the desired packet.

### **5.5 – Cloud Network (CM) Node Functionality**

The CM node is a simple node that represents a WLAN mesh network. The idea proposed in this thesis is that the mesh network is the one created from a neighbourhood of smart meters, however this does not need to be the case. Any devices that use WLAN technology to form a self configuring, static, and reliable mesh network could be used.

The program methodology for this node is similar to the CC node. *CloudMesh.java* is also a double threaded program that monitors incoming packets from a single interface. As with the CC, one thread of CM is always listening for incoming packets. When a packet does arrive, it starts a second thread passing it a copy of the information. The first thread then displays the received data and loops back to the listening state. See Figure 5.4 for a screen shot of the CM program running in a terminal. There are two procedures for displaying the packet data. Which one is used depends on the structure of the data inside the packet payload. If the data came from the GW and was originally an array list object (variable sized array), then it must be converted back into an array list and displayed using the appropriate array list methods. Otherwise the data was originally in the form of a byte array, and when CM tries to convert it into an array list, an exception will be thrown. If this is the case then the byte array is

reconstructed back into its integer values and displayed. This same procedure is implemented in the CC program to display the packet data. The listening thread is always running, right from program execution until termination.



```

fit@fit-1: ~/Desktop/CloudMesh
File Edit View Terminal Tabs Help
root@fit-1: /home/fit fit@fit-1: ~/Desktop/CloudMesh
Listening for packet ... Received from /192.168.0.1 on port 60201 with length 358
Forwarding to: /192.168.0.3
Packet from GATEWAY:
From Node: 2 Data: 1015 Battery: 2801mV
From Node: 3 Data: 499 Battery: 2988mV
From Node: 2 Data: 1015 Battery: 2801mV
From Node: 3 Data: 500 Battery: 2996mV
From Node: 2 Data: 1015 Battery: 2801mV
From Node: 3 Data: 501 Battery: 2988mV

Listening for packet ... Received from /192.168.0.3 on port 60201 with length 7
Forwarding to: /192.168.0.1
Packet from CONTROL CENTRE:
To node: 2 Command: Get Sensor Data

Listening for packet ... Received from /192.168.0.1 on port 60201 with length 7
Forwarding to: /192.168.0.3
Packet from GATEWAY:
Response to special data request:
From Node: 2 Data: 863 Battery: 2801mV

Listening for packet ... Received from /192.168.0.3 on port 60201 with length 7
Forwarding to: /192.168.0.1
Packet from CONTROL CENTRE:
To ALL nodes Command: Sleep

Listening for packet ... Received from /192.168.0.3 on port 60201 with length 7

```

Figure 5.4: CM's Sample Screen Shot.

When the second thread is started, it is in charge of routing the received packet to its next hop. This is done by looking at the IP address the packet came from and forwarding it to the opposite IP address. CM only routes packets back and forth between the GW and the CC. Once the packet has been forwarded, this thread closes the sending socket and terminates. In the event that more than one CM is desired to be used to create its own mesh network, its routing algorithm would be placed in this second thread.

## 5.6 – Gateway (GW) Node Functionality

Although *Gateway.java* is also a double threaded program like the CM and CC nodes, it functions differently because it has two types of interfaces to supervise. On one

side it interacts with the SN's through the ZigBee radio, and on the other side it interacts with the mesh network through the WLAN radio. Both threads manipulate both interfaces, however once initialized, their operations are independent of each other. The process of each thread will be explained separately.

### **5.6.1 – WLAN Thread**

This program thread listens on the WLAN socket for command packets coming from the CC. This thread is activated from the ZigBee thread near the beginning of the program once the USB port information has been set up. During activation, the WLAN thread is given the pointer to the USB port information, and this is the extent of their interaction. The port information is used so communication with the ZigBee radio connected to the computer can be made.

At this point, a receiving socket is opened and listening begins. Upon WLAN packet arrival, the necessary information is displayed on the screen, a ZigBee packet is constructed containing only the needed data, and this packet is written to the USB port. Here, the GW's ZigBee mote transmits on its radio whatever is received on its serial port.

As can be seen, this thread is responsible for channelling packets in the downstream. That is, in the direction from the higher up CC, to the lower SN's.

### **5.6.2 – ZigBee Thread**

This thread is in charge of much more functionality than the other since it must interact with the SNs in addition to forwarding information in the upstream. Both threads in the GW node are always running and are constantly listening for incoming packets on their respective interfaces. When a ZigBee packet is transmitted to the GW, the mote captures the packet from the radio, appends this packet's RSSI value to the end, and sends the resulting packet through the serial port to the computer. This ZigBee thread picks up the packet and processes it accordingly. The processing action that is taken depends on

the value in the “Type” field of the packet payload. This field is the first piece of data that is checked when a ZigBee packet is received. The actions that are taken depending on the packet type are discussed next.

### **Data Packet:**

There are two types of data packets. They both have the same packet structure and differ only in the type field identification. One type is for sampled data during regular operation, and the other is for sampled data in response to a command from the CC.

If a data packet is received that is apart of the regular sensing cycle, this data is saved in a variable sized array. Every sensor data packet that comes in has its payload information appended to the end of this array. Sensor data packets are small and it is not necessary for the GW to forward each data packet individually into the mesh network. In the case of the smart meters, the collector transmits data to the CC using the cellular phone network, so every transmission costs money. This is why the GW collects many data packets and then can forward them all as one block of data at a later time. Of course if a fault does occur the CC would need to be notified as soon as possible. For this reason certain send conditions are exploited so the GW knows when it is ideal to forward this data.

It may be desired by the monitoring company to receive sensor data information every certain number of hours. The GW does not need a countdown timer to accomplish this task. As will be explained, the SNs run a timer for periodic transmission. The GW simply needs to count how many data packets it has received before it needs to forward the data. For example, if the GW forwards data once every hour and three sensor nodes are associated with this GW that each sample every 10 minutes, the GW forwards the block of data to the CC after receiving 18 packets. At this point, all the array entries are cleared and the process starts over.

If fault data is received, it is not forwarded right away for two reasons. First, if a short power outage occurs that only lasts a couple of seconds or less, this is not necessary to forward it to the CC immediately. It will be sent in the next block of data that gets forwarded. Secondly, if a severe fault has occurred it gives the remaining SNs a chance to sample their data so that a more complete picture of the situation can be reported to the CC in one shot. This is done by the GW keeping track of how many packets it receives that contain fault data, and then forwarding a small block after receiving so many. For example, if the SNs transmit fault data every second, and there is a SN for each phase, a forwarding delay of 3 seconds can be achieved by the GW collecting 9 fault data packets. This is how the GW fulfills the requirement of providing a forwarding delay.

If a data packet is received that is in response to a command from the CC, this data is not entered into the variable sized array. This single packet is simply forwarded toward the CC for immediate response.

### **Broadcast Request ID Packet:**

The GW will receive this kind of packet from a SN which has just been activated and is looking for a GW to associate with. This means that the SN needs to be assigned a node ID so it can operate in the personal area network (PAN). When a broadcast request ID packet is received, the GW first checks to see if this particular SN had previously requested a node ID and has one being reserved by checking its sequence number. If it has, then the GW re-offers the same ID number the SN had last time. Conversely, if this is an ID request from a new SN, then the GW searches its ID table to find an available ID number. Once it is found, the ID number is marked as taken to reserve it, the sequence number of the requesting SN is coupled with this new ID along with a random number that the GW generates for its own sequence number. All of these numbers are then used to build an “Offer ID” packet which is then sent to the SN. See Figure 5.5 for an illustration of the behaviour between the address ID table and the sequence number table.

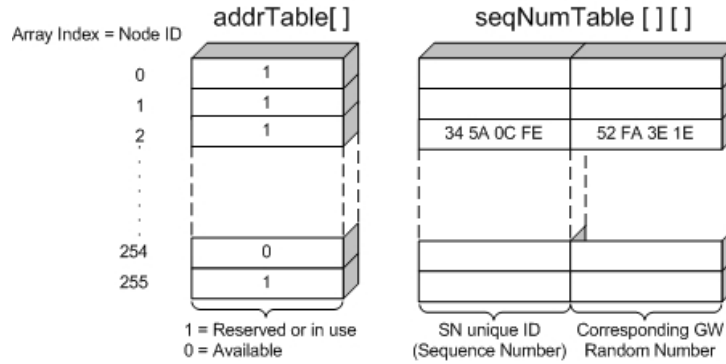


Figure 5.5: GW's Address ID Table and the Corresponding Sequence Number Table Structures. Node ID 2 has been reserved for a SN with sequence number 0x345A0CFE and a reply has been made with random number 0x52FA3E1E.

### ID Acknowledgement:

An “ID Acknowledgement” packet will be sent from the SN to a GW when it receives an “Offer ID” packet. This acknowledgement packet is the final step in the 3-way handshake method for a SN to associate with a GW. Once the GW determines that it has received an “ID Acknowledgement” packet, it checks to see if the received sequence numbers match the ones that were sent in the “Offer ID” packet. If they match, then the SN has successfully joined the network managed by this GW. If the sequence numbers do not match, then the SN has associated with another nearby GW and its reserved information (node ID number, etc...) is cleared so it can be used for another SN.

The sequence numbers used in order for a SN to associate with a GW are necessary to avoid some potential problems. This would be especially true for situations where many of these SNs and GWs are deployed in close proximity. An explanation of why they are necessary and the problems it allows the system to avoid is discussed next.

### Dynamic Node-Gateway Association:

The Dynamic Node-Gateway Association (DNGA) scheme created for this thesis is modelled after the Dynamic Host Configuration Protocol (DHCP) for LAN networks. In fact, it is a simpler version of DHCP where some unnecessary features for this ZigBee



network are taken out. DNGA is used for the same purpose as DHCP which is dynamically assigning node ID numbers (or IP addresses) to new SNs (or hosts) that join the network. Figure 5.6 shows the packet communication procedure for a SN to join the network. The following are two problems that may arise which are avoided using this DNGA scheme.

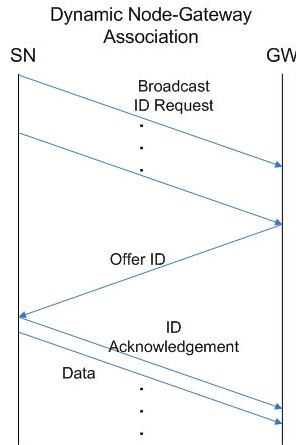


Figure 5.6: DNGA Packet Communication Procedure.

Problem 1: The following situation could take place when two, or more, SNs are in proximity of a single GW. As shown in Figure 5.7a), SN1 is activated and broadcasts an “ID Request” packet. Before the GW has a chance to respond however, SN2 is activated and broadcasts its “ID Request” packet as shown in Figure 5.7b). Note that for the ease of deployment, all SNs are uploaded with the same program and their default node ID number is 1. Therefore, both SNs transmit a request packet with source address ID of 1, and destination address of “broadcast”.

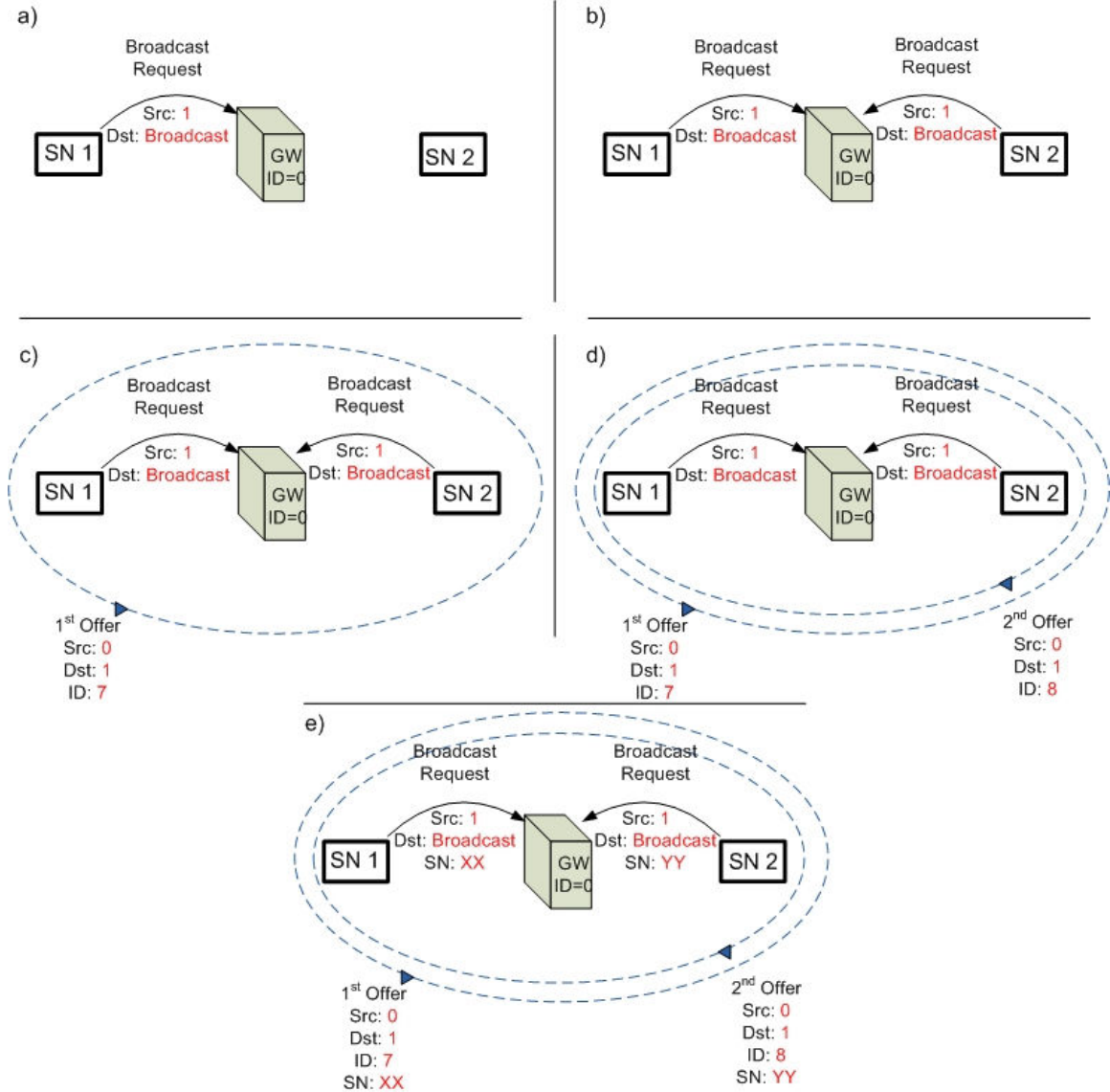


Figure 5.7: DNGA Problem 1 Depicting Node Addresses.

When the GW responds with an “Offer ID” packet to the first request of SN1, both SNs will process and accept the same offer packet, as shown in Figure 5.7c). At this point both SNs will have the same node ID address 7. It can be harmful for networking applications if two network devices have the same address. At this point, the second “Offer ID” packet with ID equal to 8 that is intended for SN2 would be ignored, as shown in Figure 5.7d).

What is needed is some way to uniquely identify each SN even though they are all uploaded with the same program prior to deployment. This identity is achieved through

the use of sequence numbers in the packet data. Each ZigBee mote has a small chip on it that contains a number that is guaranteed to be unique. This is analogous to MAC addresses for network cards. These unique numbers are utilized as each SN's sequence number. When the GW responds with an "Offer ID" packet, this packet must contain the requesting SN's sequence number so other SNs understand it is not directed to them. The resulting diagram is shown in Figure 5.7e).

Problem 2: In this situation a single SN is in proximity to two, or more, GW nodes. When the SN is activated and broadcasts its "ID Request" packet containing its sequence number, both GW's will pick up this message as shown in Figure 5.8a). GW1 and GW2 will each reserve an ID and respond with their own "Offer ID" packet containing the SN's sequence number. This is shown in Figure 5.8b). The node IDs that are offered may be the same ID or two different ID's. Either way, the SN will have to pick a GW to associate with and broadcast an "ID Acknowledgement" packet. Both GW's will hear this acknowledgement, and both will also assume the SN associated with them. This would especially be true if the GWs happen to offer the same node ID value to begin with. Figure 5.8c) depicts the resulting problem.

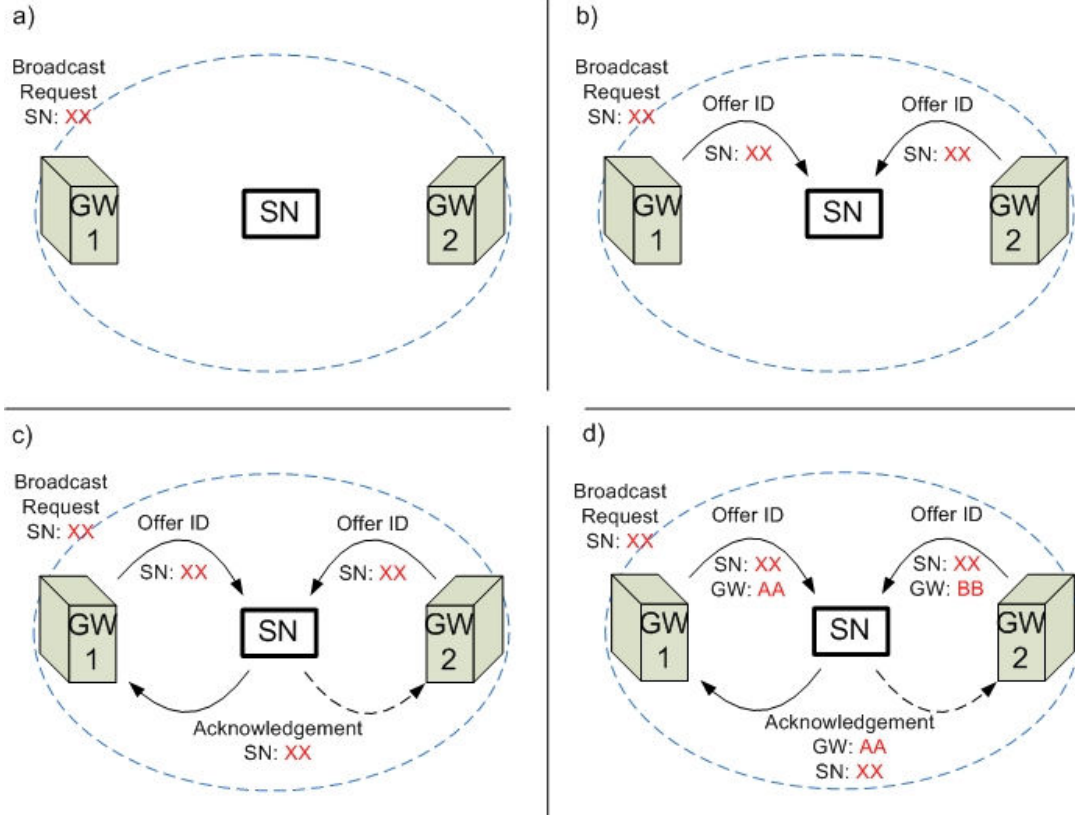


Figure 5.8: DNGA Problem 2. Depicting packet sequence numbers, node addresses remain the same as in problem 1.

A way to uniquely identify the conversations between the SN and both GW's is needed. Once again a sequence number will be employed, this time in the GW. When a GW receives an “ID Request” packet it generates a random number using the current system time as the random number seed. This random number is included in the “Offer ID” packet and is also saved and used for the entire DNGA conversation. Therefore when the SN responds with an “ID Acknowledgement” it includes in the packet the sequence number for the GW that it wishes to associate with, as shown in Figure 5.8d). This way GW2 knows that it was not chosen by the SN, and it can release the ID it previously reserved.

Figure 5.9 shows a sample screen shot of the GW's terminal window during program execution. Also, Figure 5.10 illustrates the different ZigBee packet structures that are transmitted back and fourth between the SNs and the GWs and Table 5.1

describes their “Type” field values.

```

fit@fit-sony: /opt/tinyos-2.0.2/support/sdk/java
File Edit View Terminal Tabs Help
root@fit-sony: /home/fit fit@fit-sony: /opt/tinyos-2

00 BB BB 00 03 06 00 06 44 01 F6 01 A2 F0
Sensor Data converted to decimal = 502 Battery level = 2996mV

00 BB BB 00 02 06 00 06 44 03 56 01 BF 02
Sensor Data converted to decimal = 854 Battery level = 2801mV

00 BB BB 00 03 06 00 06 44 03 A7 01 A3 FA
Sensor Data converted to decimal = 935 Battery level = 2988mV

00 BB BB 00 02 06 00 06 44 03 5E 01 BF 02
Sensor Data converted to decimal = 862 Battery level = 2801mV

00 BB BB 00 03 06 00 06 44 03 A8 01 A3 FD
Sensor Data converted to decimal = 936 Battery level = 2988mV

WLAN: Received command packet for node: 2 to retrieve sensor data.
00 BB BB 00 02 06 00 06 64 03 5F 01 BF 02
Response to special data request: Sensor Data converted to decimal = 863 Battery level = 2801mV
-----GATEWAY: Sending gathered sensor data to Control Centre-----

00 BB BB 00 03 06 00 06 44 03 A8 01 A3 FD
Sensor Data converted to decimal = 936 Battery level = 2988mV

00 BB BB 00 02 06 00 06 44 03 5F 01 BF 02
Sensor Data converted to decimal = 863 Battery level = 2801mV

00 BB BB 00 03 06 00 06 44 03 A8 01 A3 FD
Sensor Data converted to decimal = 936 Battery level = 2988mV

WLAN: Received command packet for ALL nodes to be in-active for 10 minutes.
WLAN: Received command packet for ALL nodes to Wake up.
00 BB BB 00 02 06 00 06 44 03 64 01 BF 02
Sensor Data converted to decimal = 868 Battery level = 2801mV

00 BB BB 00 03 06 00 06 44 03 A9 01 A3 FA
Sensor Data converted to decimal = 937 Battery level = 2988mV
    
```

Figure 5.9: GW's Sample Screen Shot.

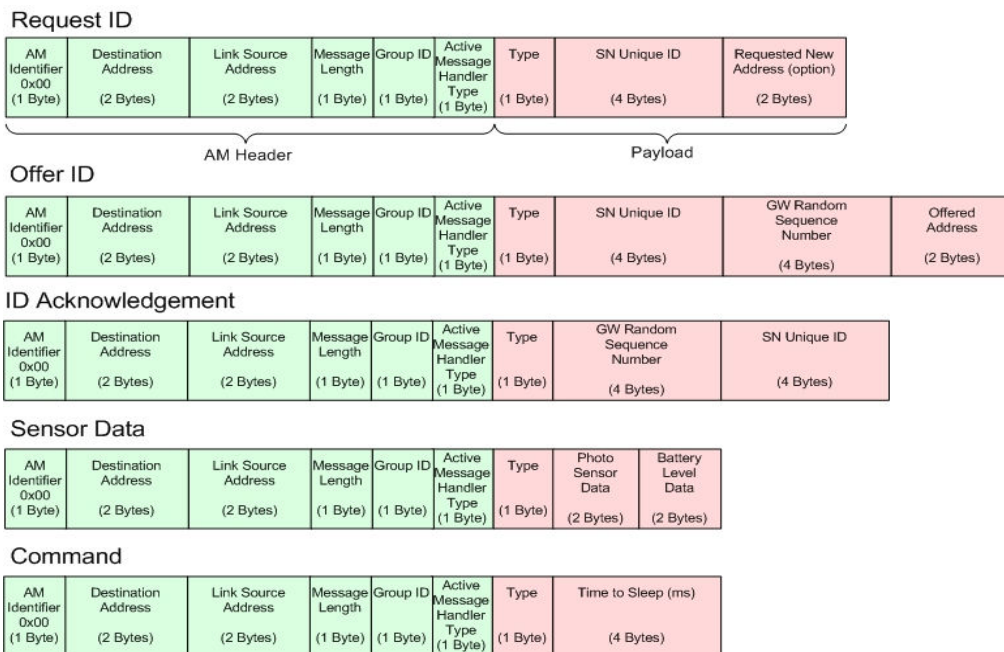


Figure 5.10: ZigBee Packet Structures.

Table 5.1: ZigBee Packet “Type” Field Values

Packet Type	Type Field		Description
	ASCII Character	Value (Hex)	
Request ID	R	52	–
Offer ID	O	4F	–
ID Acknowledgement	A	41	–
Sensor Data	D	44	Data packet for normal operation.
	d	64	Data packet in response to CC command
CC Command	d	64	“Get Sensor Data” Command
	S	53	“Sleep” Command
	W	57	“Wake Up” Command

Note: The system can tell the difference between the two “d” type packets because of the destination they are intended for.

## 5.7 – Sensor Node (SN) Functionality

### 5.7.1 – Normal Operation

The SNs are a different kind of animal compared to the nodes discussed so far because they are written in the TinyOS language and not in Java. TinyOS is an event-driven language designed for sensing applications. As a result, all blocks of code in the program do not run until an event triggers them to do so.

After activation, the ZigBee radio is called upon to be initialized. Once this is successfully completed, another function is called (*AMControl.startDone()*) to access the DS2401 chip and obtain the SN's unique sequence number. Now a function to build and broadcast an “ID Request” packet is called (*BroadcastRequestID()*). After it sends the packet it starts a timer for 3 seconds and ends the thread.

A new thread is spawned when either the 3 second timer completes or a packet is received on the radio. If an “Offer ID” packet is not received before the 3 second timer is completed (*Timer0.fired()*), it recalls the block of code to broadcast another “ID Request”

packet. Otherwise, if an “Offer ID” packet was received (*Receive.receive()*) then the timer is stopped, the GW's mote ID is saved, it changes its node ID to the one that was offered to it, and a function is called to send an “ID Acknowledgement” packet (*OfferAcknowledge()*).

Once the “ID Acknowledgement” packet is sent to the GW, the normal sensing cycle begins. A block of code is called to sample the SN's battery level and then *ReadBattery.readDone()* is called. The battery value is saved and a read of the photosensor is called. Upon completion, the next block of code (*ReadSensor.readDone()*) is responsible for constructing the “Data” packet and sending it to the radio. Next, a check is done to determine if the most recent sensor value is between the operating thresholds. If so then the timer is set for a longer time period than if the value is outside the thresholds. This is how the SN accomplishes its dynamic sensing cycle period requirement.

### **5.7.2 – Received Commands**

There are three valid commands that the SN can receive and process. The SN would receive a “Get Sensor Data” command if the CC wishes to probe this node for its current sensor values. Once received, the SN simply runs through the same procedure it does when sampling during its normal cycle. The only difference is that a flag is set so that when the data packet is being put together, it knows to change the “Type” field to indicate that this is the data for the special data request. This special sampling operation does not interfere with the regular sampling cycle.

In the event that a “Sleep” command is received by the SN, the procedure taken is very simple. The SN will stop the timer that is keeping track of the sampling cycle, set the “idle” flag to indicate to the program that sleep mode is in effect, and then restart the timer for the indicated amount of time to sleep. Once in sleep mode, the only command that the SN will respond to is the “Wake up” command. There are two ways to escape sleep mode (without turning the device off), one of which is by use of the “Wake up”

command. The other is the SN will automatically wake up when the sleep timer expires. In either case the “idle” flag is reset to zero and the sampling cycle is restarted. Note that only one timer is used for all SN operations. The SN's functionality combined with the CC's ability to issue commands, fulfills the system requirement of allowing a user to remotely manipulate the SNs.

At this point the curious reader is referred to [47] which explains the ZigBee's microcontroller power management. In TinyOS-2.x the programmer cannot directly tell the ZigBee device to go into sleep mode. Power management is done automatically in the background and the microcontroller will always go into the lowest power state possible depending on what is in its program queue. The SN in this application is still running a timer during “sleep” mode so this would not be the lowest power state. For this reason, this state is sometimes referred to as being “in-active” in the program code.

### 5.7.3 – LED States

As an additional feature, the three light emitting diodes (LEDs) on the ZigBee motes have been programmed to give a visual feedback as to what the SN is doing. This would be useful for work crews when installing and working around these devices. There is a red, green, and a yellow LED on the devices. Table 5.2 illustrates the LED's behaviour during different SN activities.

Table 5.2: SN LED States and Description.

Activity/State	Red	Green	Yellow
Sending “ID Request” packet. (Not yet received an “Offer ID” packet)	Flashing	Off	Off
Obtained new ID, and normal operation	Off	On (solid)	Off
Fault detected	Off	On (solid)	Flashing
During In-active/Sleep mode	Off	Off	On (solid)



### 5.8 – System Robustness

In this section, a highlight of some system features will be discussed that improve the robustness of the system. The most prominent feature is the DNGA process in which the SNs are automatically configured to join the network. This is extremely important for deploying a properly structured sensor network. Also, extracting the number from inside the DS2401 chip on the SN, as the source for a unique sequence number, allows the GW to confidently offer a unique node ID address. The fact that the SN does not assume the GW's node ID during start-up is also advantageous. This means that a SN will be able to associate with any GW regardless of its address. Additionally, if a SN gets reset and tries to re-associate with the same GW, it will be offered the same node ID it previously had. This is of use to the CC operators who monitor the data that is received from each SN. It would help to avoid confusion if the SN ID's are not often changing.

Another useful piece of information that the CC would receive is the battery levels of all SNs. This way it can be seen well in advance if a SN will need a battery change so it does not mysteriously drop out of the network. In addition, there are routing schemes oriented for ad-hoc sensor networks that utilize the energy level of the node as a routing decision. For example, during route assessment, a path is chosen that routes through nodes that have higher battery levels and avoids the nodes that are running low. This way the network as a whole is able to stay alive for a longer period of time. This may prove to be a useful asset in the event that many nodes are associated with a single GW through a multi-hop fashion.

The GW's ability to simultaneously control the upstream and downstream as two separate threads greatly increases the performance capabilities of this crucial node. If a single programming thread was responsible for every GW function, then a lag in response could occur during high traffic times. This may also result in dropped or missed packets that contain vital information for CC feedback.

Lastly regarding the WLAN test network, UDP sockets are used as the connecting

sockets. As a result, say for example the CM node drops out of the network to restart, a program error does not occur in the remaining nodes that are running a Java program (GW and CC). This would be the case if a TCP socket is utilized because it is a connection-oriented socket. Therefore the program would terminate and await a restart. In the case of this system, packets would simply be lost while the CM node is out of the network and normal operation would resume once the node returns.

### **5.9 – Performance Metrics**

Referring back to Section 5.1, the purpose of this thesis is to develop a prototype for a transmission line monitoring communication scheme. Therefore, the main performance metric is whether the prototype works and completes the tasks it is intended to do. As seen in the demonstration, the ZigBee functionality and the WLAN backbone network all operate as described in Sections 5.5 through 5.8. Nevertheless, some system analysis is conducted.

The ZigBee testbed described in Chapter 4 is employed again, but this time to test the maximum throughput from one mote to another. The transmit and receive buffer sizes for the CC2420 radio on the MicaZ motes are the same. As a result, the motes are capable of receiving data at the same rate they can send data. The transmitter is the sender and of course, the BS is the receiver. With only one transmitter sending data to the BS, a maximum transmit rate of 19.5 kbps is found. This is done by the transmitter sending 1000 packets of 32 bytes each in a minimum of approximately 13 seconds. Since the SN data packets are only 13 bytes long, this computes to the GW being able to process 189 data packets/sec.

Throughput becomes a little more complicated when repeating this same test with two or more transmitting motes. At its best, the BS will receive all the transmitted packets as long as the combined sending rate of all transmitters are 19.5 kbps or below. Once the BS needs to receive data faster than this, then packets will start to be dropped. Figure 5.11 illustrates this behaviour which also provides a limit that will determine how

fast SNs will be able to sample and transmit during their faster fault cycle depending on how many SN are associated with a single GW.

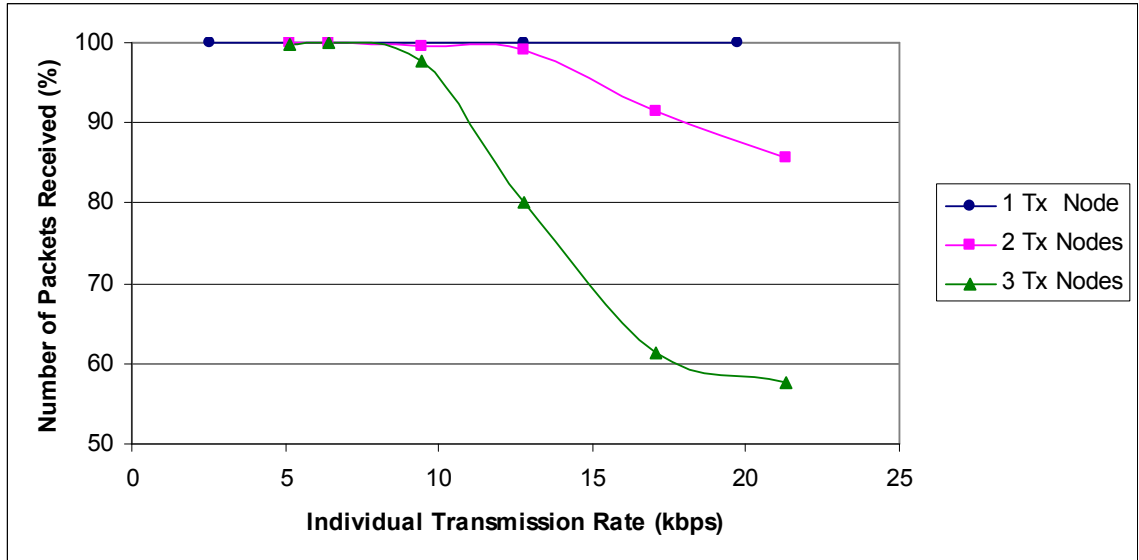


Figure 5.11: ZigBee Receiver Throughput.

On the down side, the ZigBee motes only implement carrier sense multiple access (CSMA). This means that before the radio transmits, it senses the wireless channel to see if it is currently being used. If not then transmission occurs, and if so, then it will wait a small random amount of time and then try again. The draw back transpires when two motes want to transmit at the same time. Both motes will sense that the channel is not being used, they will transmit, and a collision will take place. This can be avoided as long as the SN transmissions are offset from each other. Otherwise, a scheduling scheme or a more sophisticated multiple access scheme will need to be implemented.

Testing the throughput of the developed WLAN backbone does not make sense in this case for two reasons. Firstly, this backbone is simply used for testing purposes and would not accurately mimic the network that the ZigBee devices are intended to be used in. More importantly, WLAN technology has much higher throughput capabilities and would be able to easily handle the traffic given to it from the ZigBee network. In terms of throughput, the GW's receiving ZigBee radio is the bottleneck of the system.

The reliability of this monitoring communication scheme depends on the abilities of both ZigBee and WLAN. WLAN has been around for some time now and is proven to be an excellent close range wireless communication technology. Its ability to operate efficiently and reliably is one of the reasons for its wide spread popularity. In addition, ZigBee's wireless communication ability was considerably tested in Chapter 4. Results show that ZigBee's reliability is suitable for use in a monitoring network in a transmission line environment.

### **5.10 – Extra System Information**

#### **5.10.1 – Overall System**

The programs written for all nodes in this system are application layer programs. This makes it easier for development and debugging purposes, especially since this is a first prototype.

The WLAN network that was created for this project is intended for testing the ZigBee system's functionality. The programs in the CM and CC nodes would not be used in actual deployment. Nonetheless, the program for the CM node could be altered to model the functionality of a smart meter if the designer companies decide to release such details. Since a smart meter network would perform its own route discovery for forwarding packets, and only one hop is being utilized within this cloud, this thesis assumes that route discovery has already taken place. This is enough to prove that the GW can construct and send/receive WLAN packets which make it feasible to be used with a smart meter network. Table 5.3 illustrates the IP addresses used for the three WLAN nodes in the network.

Table 5.3: IP Addresses Used for the WLAN Interfaces.

Node	IP Address
GW	192.168.0.1
CM	192.168.0.2
CC	192.168.0.3

In the event that the CM node continues to be used, whether it is modelled as a smart meter or not, it is not necessary for the program to print the traffic information to the screen. This is also true for the GW node. The displaying portions of these programs take up processing time and are only useful to the programmer for debugging purposes. They may be commented out for any system throughput or efficiency evaluations, or any test deployments.

The power requirements for the SNs and the GW are not addressed in this thesis. Nor is any energy consumption evaluations performed. The amount of research needed to properly tackle this problem would be a project in itself. For the GW, properly tuning the garbage collection settings in Java could reduce power consumption and improve throughput. The garbage collector manages system memory automatically by periodically checking program objects. If an object can no longer be referenced then the memory it was consuming is freed. It would also be necessary to analyze the programs in the ZigBee motes, and alter them if needed, so that the SN would always been in the lowest power state possible. This could drastically increase the nodes lifetime and would require a thorough understanding of the microcontroller's power state analyzer. Furthermore, an efficient and convenient power source would have to be used or developed so that the SN has a way to continuously recharge its batteries. This could be a converter that draws power from small solar panels or from the electromagnetic field surrounding the transmission line.

### 5.10.2 – Gateway Node

It was mentioned in Section 5.7.2 that *Gateway.java* uses a random number as its sequence number during the DNGA process, and this random number is generated using the system time as the seed. This could potentially result in a problem if a single SN is in proximity to two or more GWs because the clock value used for the seed is accurate to 1 millisecond precision. If the two adjacent GWs have synchronized system times accurate to the millisecond, then it is possible that a SN will receive an “Offer ID” packet from two GWs with the same sequence number. If it is likely to happen that the GW nodes will have the same system times, then the GW's unique sequence number seed can come from the WLAN card's MAC address or the ZigBee mote's unique ID. Then future sequence numbers can just be an increment of the previous one.

The network embedded systems C (nesC) programming language is used in TinyOS to program the ZigBee devices. nesC is an unsigned language where Java is a signed language. This presented a problem in the GW node when data needs to be sent to, or received from, the ZigBee mote. All data is broken down to bytes (8 bits) for transmission through the serial port between the two GW devices. In the case of receiving an integer from the ZigBee mote, four consecutive bytes need to be combined to form that integer. In *Gateway.java* this short converter function is called *bytesToInt()*. Each individual byte needs to be checked to see if Java perceives it as a negative number. If so then the decimal number 256 needs to be added to each negative byte. Then three of the bytes need to be shifted left by the number of positions that corresponds with the byte significance order within the integer, and then all of them added together. Additionally, for sending an integer to the ZigBee mote, it must be broken down into bytes if it hasn't been already, as in the case of a WLAN packet. To break the integer down, the AND operator (&) is used to capture the individual bytes and then they are unsigned shifted right and casted to a byte memory type. These conversions are needed in order to ensure correct data interpretation between the two programming languages.

Lastly for the GW, is why a variable sized array is used when this creates more

work to convert from objects to byte arrays, and then back again? The answer is simple, efficiency and flexibility. Instead of declaring a static sized array, the packet accumulation array in the GW changes in size as needed. This way if a fault occurs after only 15 packets then only 15 packets are sent, instead of the entire array with 85 packets of blank data if a static sized array of 100 packets is used. Also if the monitoring company decides to increase the interval in which the GW forwards the accumulated data packets, the GW program does not have to be re-compiled and installed to account for the larger array size that would be needed. The variable sized array allows the operators to be free in choosing different system parameters as needed.

### 5.10.3 – Sensor Node

The version of TinyOS used in this project came with the support for the MicaZ devices to sample their battery supply. An input to the internal analog to digital converter (ADC) is connected to the batteries. The result from the ADC must be modified using a short formula to get the battery reading in millivolt (mV) units. This conversion formula is explained in [48]. Therefore, it is just a matter of properly working the functionality into the SN program.

The driver code for the DS2401 chip did not come included in the TinyOS environment. Eleven separate nesC files were obtained from the TinyOS website [45]. Once they were properly connected to the sensor application program, the SN was able to retrieve its unique node ID sequence number.

It was mentioned in problem 2 of the DGNA process (Section 5.7.2) that if a SN received an “Offer ID” packet from two different GWs then it has to pick one. Currently the SN picks on a first come first serve basis. If desired, in the future is it possible to program the SN to decide based on the signal strength of the offer packets using their RSSI values. Therefore, stronger connections are always chosen.

There are two bytes at the end of the “Request ID” packet that have not yet been

discussed in Figure 5.10. These two bytes are intended to be used in the event that a SN needs to request a node ID but has not been powered down prior to this. Using this field the SN may specify a node ID it would like to have and the GW may offer this ID depending on its availability. If this field is not used it remains as all zeros. These extra two bytes could also be used for other extended applications that may arise in the future.

#### **5.10.4 – ZigBee Testbed**

In Section 4.3 it was stated that modifications to the TinyOS driver files for the radio chip were needed to allow error packets to pass through up the radio stack. Table 5.4 will describe what those modifications were and what files they were made in. These changes were only compiled for the BS node in the channel BER tests. Each files' changes were switched back to program the transmitter node since only the BS needed to accept erroneous packets. Some other key files that future students are likely to utilize in the TinyOS tree are mentioned in Table 5.5.



Table 5.4: Modified Files and Description for BS in ZigBee Testbed.

File	Line(s)	Change	Description
CC2420ActiveMessageC.nc	66-68	Added: components CC2420ActiveMessageP; components LedsC as Leds;	Allows the radio's AM layer to change the LED's states. This is useful for determining which part of the program the packets are travelling through since printf statements cannot be used.
	118	Added: CC2420ActiveMessageP.Leds -> Leds;	
CC2420ActiveMessageP.nc	53	Added: interface Leds;	Same as above description.
	190-201	Commented out all except kept line 192.	Always send packets through.
CC2420ReceiveP.nc	185	Added: call Leds.led0Toggle(); //Red LED	To be notified any time the interrupt fired that the radio received something.
	304	Added inside the 'if' statement:    !(( buf[ rxFrameLength ] >> 7 ) && rx_buf)	To always bypass the CRC check.

Table 5.5: Useful TinyOS Parameters.

File	Line	Description
AM.h	19	Defines the motes active message group ID.
	23	Defines the motes active message node ID address.
CC2420.h	100	Defines the radio frequency/channel that the mote will transmit
	104	Defines the power level that the mote will transmit with.

## Chapter 6

### Conclusion and Future Work

---

#### 6.1 – Conclusion

A practical, self configuring, sampling and forwarding scheme for a transmission line monitoring system was developed and a hierarchical communication topology involving smart meters was proposed. The sensor nodes are able to be self configured with the help of the gateway device through a protocol similar to the popular DHCP. Unique sequence numbers are utilized in both the sensor node and in the gateway to differentiate between distinct conversations from multiple devices. Through this process the gateway is able to assign unique node ID values to each sensor node in the network. Once initialized, the sensor nodes cycle through their periodic sensing cycle looking for signs of a fault. A dynamic cycle period is implemented so that a more rapid sensing cycle occurs when a fault is present. Additionally, in the interest of saving bandwidth, the gateway devices do not forward each sensor data packet individually. A collection of data packets are gathered and then forwarded together when a fault occurs or when a specified amount of time has elapsed.

A graphical user interface was developed for the control centre. This allows a user to easily monitor and remotely communicate with the sensor nodes. As with all the communication nodes in this system, the control centre also has bi-directional communication capabilities. Specific commands can be sent to the sensor nodes for various reasons to help control the system as needed. In addition, received data that is an indication that a fault has occurred is highlighted so it can be easily identified by the user. The developed prototype operates successfully and its functionality was fully documented.

A ZigBee testbed was also developed and used to determine its ability to perform

in harsh environments, such as the one that may be found in a transmission line or power station environment. With a transmitter antenna height of 1.5m off the ground, it was found that the ZigBee radios had a reliable range of 70m without any data correction techniques utilized. An impressive maximum packet error rate of 4.8% was experienced when the transmitter radio was under the hood of a car in front of the engine. This figure includes city and high speed driving. Furthermore, in a machine shop with many concrete and steel obstructions a maximum packet error rate of 7% was discovered within a radius of 10m from the receiver. These findings collectively demonstrate that ZigBee devices are able to perform suitably with harsh surroundings.

### **6.2 – Future Work:**

Given that this project completes a first prototype, there are many more features that would be helpful in making the system more flexible and efficient. Next is a list of tests and ideas for system features that would be a good start for furthering the abilities of transmission line monitoring systems. This list is not intended to be in any particular order of importance.

1. Conduct a BER test for a ZigBee mote attached to a transmission line to see if the electromagnetic field interferes with packet transmissions.
2. The monitoring system can be tested using an actual smart meter backbone.
3. Address the power requirements for the prototype system as discussed in Section 5.10.1.
4. Security/Encryption can be added to the ZigBee signals to help prevent foul play.
5. As this project is implemented in the application layer, the same project could be implemented in the network layer to save power and increase speed by reducing the amount of computing time.
6. Since a communication system would already be in place it can be used to support additional applications. A wider range of sensors can be used to detect overheating, collapse, conductor deterioration or icing, or even weather patterns.
7. As of this first prototype, the CC is able to send a command to a single SN or all

## 6. CONCLUSION AND FUTURE WORK

SNs. Flexibility to allow the CC to send a command to a group of SN's would also be beneficial.

8. When the GW is collecting SN data and grouping it together for a later transmission, it could append a time stamp to the sample so the CC would know more information as to when each piece of data was logged.
9. To further expand this monitoring system by increasing the size of the network, it will be necessary to add additional GW nodes. In this case it will be required for the CC to know which GW the groups of sensor data came from. Furthermore, the CC will need to be able to specify which SN associated with which GW to send a command packet.
10. It is well known that the amount of load on a transmission line varies for many different reasons. It may be desired by the CC operators to be able to remotely change the threshold levels within the detecting devices to indicate a fault. Another command can also be added to remotely change how many packets the GW must wait for before transmitting the collected block of sensor data, thus varying the transmission intervals.
11. Aside from resetting the GW node, it currently has no way to clear a SN's ID from being marked as in use once association is confirmed. For future prototypes, the GW should keep track of how many scheduled data packets it receives, and if two or more sampling cycles are missed then it can send a "still alive?" message to that SN or just clear it from the address ID list.
12. Currently the ZigBee motes employ CSMA as their multiple access technique. CSMA with collision detection or some other scheduling scheme may be applied to reduce the amount of collisions of ZigBee packets arriving at the GW during high traffic periods.
13. Lastly, each SN node runs a timer to notify itself when it is time to sample the transmission line and the SNs are not necessarily in sync with each other. Therefore, it may be advantageous for a SN to transmit a beacon when it detects a fault to interrupt the remaining SNs to sample at that time. In order to prevent a flood of beacons for a single fault, a SN only transmits one beacon for every consecutive stream of faults. Once a normal reading occurs, then it starts over.

---

**References:**

---

- [1] Statistics Canada, [Online] Available: <http://www40.statcan.gc.ca/101/cst01/demo02a-eng.htm> Accessed July 2009.
- [2] Ontario Power Authority, “OPA Progress Report on Electricity Supply, First-Quarter 2009” [Online] Available: [http://www.powerauthority.on.ca/Storage/100/9571\\_2009\\_Q1\\_A\\_Progress\\_Report\\_On\\_Electricity\\_Supply\\_\(2\).pdf](http://www.powerauthority.on.ca/Storage/100/9571_2009_Q1_A_Progress_Report_On_Electricity_Supply_(2).pdf) Accessed July 2009.
- [3] Hydro One, [Online] Available: <http://www.hydroone.com/en/about/history/timeline/> Accessed July 2009.
- [4] Centre for Energy, [Online], Available: <http://www.centreforenergy.com/AboutEnergy/Electricity/Transmission/History.asp> Accessed July 2009.
- [5] Ontario Energy Board, “2007 Yearbook of Electricity Distributors” August 2008, [Online] Available: <http://www.oeb.gov.on.ca/OEB/About+the+OEB/Energy+Statistics+and+Maps> Accessed July 2009.
- [6] M. E. El-Hawary “Introduction to Electrical Power Systems” Piscataway, NJ: John Wiley & Sons, Inc., 2008.
- [7] Essex Power, [Online] Available: <http://www.essexpower.ca/> Accessed: July 2009.
- [8] Arc Advisor, [Online] Available: [http://www.arcadvisor.com/faq/mva\\_to\\_ka.html](http://www.arcadvisor.com/faq/mva_to_ka.html) Accessed July 2009.
- [9] Energy Access “Article Connecting with Value of Smart Metering” [Online] Available: [http://www.energyaxis.com/pdf/Article\\_Connecting\\_with\\_value\\_of\\_smart\\_metering.pdf](http://www.energyaxis.com/pdf/Article_Connecting_with_value_of_smart_metering.pdf) Accessed July 2009.
- [10] Ontario Energy Board “Smart Meter Implementation Plan: Report of the Board to the Minister” January 2005, [Online] Available: [http://www.oeb.gov.on.ca/documents/communications/pressreleases/2005/press\\_release\\_sm\\_implementationplan\\_260105.pdf](http://www.oeb.gov.on.ca/documents/communications/pressreleases/2005/press_release_sm_implementationplan_260105.pdf) Accessed July 2009.
- [11] Ontario Ministry of Energy and Infrastructure, [Online] Available: <http://www.mei.gov.on.ca/english/energy/electricity/?page=smart-meters> Accessed July 2009.

- [12] Y. Yang, D. Divan, R.G. Harley, T.G. Habetler “*Power line sensornet - A New Concept for Power Grid Monitoring*” (2006) 2006 IEEE Power Engineering Society General Meeting, Montreal, Quebec, Canada.
- [13] J.H. Rodriguez, J.C. Tello “*Intelligent Wireless System to Monitoring Mechanical Fault in Power Transmission Lines*” (2008) Proceedings - Electronics, Robotics and Automotive Mechanics Conference, CERMA 2008, Cuernavaca, Morelos, Mexico, September 2008, pp. 99-104.
- [14] S. Gumbo, H.N. Muyingi “*Performance Investigation of Wireless Sensor Network for Long Distance Overhead Power Lines; Mica2 Motes, a Case Study*” (2008) Proceedings - 3rd International Conference on Broadband Communications, Informatics and Biomedical Applications, BroadCom 2008, Pretoria, South Africa, November 2008, pp. 443-450.
- [15] J. Chen, K. Shubhalaxmi, A.K. Somani “*Energy Efficient Model for Data Gathering in Structured Multiclustered Wireless Sensor Networks*” (2006) Conference Proceedings of the IEEE International Performance, Computing, and Communications Conference, 2006, Phoenix Arizona, USA, 2006, pp. 381-388.
- [16] X. Li, K. Fang, J. Gu, L. Zhang “*An Improved ZigBee Routing Strategy for Monitoring System*” (2008) Proceedings - The 1st International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2008, Wuhan, China, 2008, pp. 255-258.
- [17] G. Qing, H. Jingquan, H. Hongzhi “*Design and Implementation of Testing Network for Power Line Fault Detection Based on nRF905*” (2007) 2007 8th International Conference on Electronic Measurement and Instruments, ICEMI, Xi'an, China, August 2007, pp. 3513-3517.
- [18] R.A. León, V. Vittal, G. Manimaran “*Application of Sensor Network for Secure Electric Energy Infrastructure*” (2007) IEEE Transactions on Power Delivery, 22 (2), pp. 1021-1028.
- [19] D.J. Marihart “*Communications Technology Guidelines for EMS/SCADA Systems*” (2001) IEEE Transactions on Power Delivery., 16 (2), pp. 181-188
- [20] B. Cole “*Utilities Look to the Skies for Monitoring the Power Grid*” (2004) IEEE Distributed Systems Online, 5 (11).
- [21] W. Zhao, F.E. Villaseca “*Byzantine Fault Tolerance for Electric Power Grid Monitoring and Control*” (2008) Proceedings of The International Conference on Embedded Software and Systems, ICESS 2008, Chengdu, Sichuan, China, July 2008, pp. 129-135.
- [22] X. Huang, Q. Sun, J. Ding “*An On-line Monitoring System of Transmission Line*

- Conductor De-icing*” (2008) 2008 3rd IEEE Conference on Industrial Electronics and Applications, ICIEA 2008, Singapore, June 2008, pp. 891-896.
- [23] M. Lin, Y. Wu, I. Wassell “*Wireless Sensor Network: Water Distribution Monitoring System*” (2008) 2008 IEEE Radio and Wireless Symposium, RWS, Orlando, Florida, USA, January 2008, pp. 775-778.
- [24] E. Montón, J.F. Hernandez, J.M. Blasco, T. Hervé, J. Micallef, I. Grech, A. Brincat, V. Traver “*Body Area Network for Wireless Patient Monitoring*” (2008) IET Communications, 2 (2), pp. 215-222.
- [25] IEEE “*IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements, Part 11: Wireless LAN MAC and PHY Specifications*” June 2007, [Online] Available: <http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=35824> Accessed July 2009.
- [26] S. Kim, S.-J. Lee, S. Choi “*The impact of IEEE 802.11 MAC strategies on multi-hop wireless mesh networks*” (2007) 2006 2nd IEEE Workshop on Wireless Mesh Networks, WiMESH 2006, Reston, Virginia, USA, September 2006, pp. 38-47.
- [27] Unwired Adventures, “*DEFCON WiFi Shootout 2005*” [Online] Available: [http://www.unwiredadventures.com/unwire/2005/12/defcon\\_wifi\\_sho.html](http://www.unwiredadventures.com/unwire/2005/12/defcon_wifi_sho.html) Accessed July 2009.
- [28] T. Ireland, A. Nyzio, M. Zink, J. Kurose “*The Impact of Directional Antenna Orientation, Spacing, and Channel Separation on Long-distance Multi-hop 802.11g Networks: A Measurement Study*” (2007) Proceedings of the 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOpt 2007, Limassol, Cyprus, April 2007.
- [29] K. Shuaib, M. Boulmalf, F. Sallabi, A. Lakas “*Co-existence of Zigbee and WLAN, a Performance Study*” (2006) 2006 IFIP International Conference on Wireless and Optical Communications Networks, Bangalore, India, April 2006.
- [30] G. Yang, Y. Yu “*ZigBee Networks Performance Under WLAN 802.11b/g Interference*” (2009) 2009 4th International Symposium on Wireless and Pervasive Computing, ISWPC 2009, Melbourne, Australia, February 2009.
- [31] F. Zhang, T.D. Todd, D. Zhao, V. Kezys “*Power Saving Access Points for IEEE 802.11 Wireless Network Infrastructure*” (2006) IEEE Transactions on Mobile Computing, 5 (2), pp. 144-156.
- [32] A. Farbod, D.T. Todd “*Resource Allocation and Outage Control for Solar-Powered WLAN Mesh Networks*” (2007) IEEE Transactions on Mobile Computing, 6 (8), pp. 960-970.

- [33] S. Siwamogsatham, K. Hiranpruck, C. Luangingsakut, S. Srilasak “*Revisiting the Impact of Encryption on Performance of IEEE 802.11 WLAN*” (2008) 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, ECTI-CON 2008, Krabi, Thailand, May 2008, 1, pp. 381-384.
- [34] IEEE “*IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks-Specific Requirements, Part 15.4: Wireless MAC and PHY Specifications for Low-Rate WPANs*” September 2006, [Online] Available: <http://ieeexplore.ieee.org/xpl/standardstoc.jsp?isnumber=35824> Accessed July 2009.
- [35] M. Hameed, H. Trsek, O. Graeser, and J. Jasperneite “*Performance Investigation and Optimization of IEEE 802.15.4 for Industrial Wireless Sensor Networks*” (2008) IEEE Symposium on Emerging Technologies and Factory Automation, ETFA, Hamburg, Germany, September 2008, pp. 1016-1022.
- [36] M. Zeghdoud, P. Cordier, and M. Terré “*Impact of Clear Channel Assessment Mode on the Performance of ZigBee Operating in a WiFi Environment*” (2006) 2006 1st Workshop on Operator-Assisted (Wireless-Mesh) Community Networks, OpComm 2006, Berlin, Germany, September 2006.
- [37] M.U. Ilyas, and H. Radha “*Measurement Based Analysis and Modeling of the Error Process in IEEE 802.15.4 LR-WPANs*” (2008) Proceedings - IEEE INFOCOM, Phoenix Arizona, USA, April 2008, pp. 1948-1956.
- [38] M. Ahmed, C.U. Saraydar, T. ElBatt, J. Yin, T. Talty, and M. Ames “*Intra-vehicular Wireless Networks*” (2007) GLOBECOM - IEEE Global Telecommunications Conference, Washington DC, USA, November 2007.
- [39] D. Sexton, M. Mahony, M. Lapinski, and J. Werb “*Radio Channel Quality in Industrial Wireless Sensor Networks*” (2005) Proceedings of the ISA/IEEE 2005 Sensors for Industry Conference, Sicon'05, Houston Texas, USA, February 2005, pp. 88-94.
- [40] C. Reinisch, W. Kastner, G. Neugschwandtner, and W. Granzer “*Wireless Technologies in Home and Building Automation*” (2007) IEEE International Conference on Industrial Informatics (INDIN), Vienna, Austria, July 2007, 1, pp. 93-98.
- [41] H.S. Kim, J.-H. Song, and S. Lee, “*Energy-Efficient Traffic Scheduling in IEEE 802.15.4 For Home Automation Networks*” (2007) IEEE Transactions on Consumer Electronics, 53 (2), pp. 369-374.
- [42] A.D. Siuli Roy, and S. Bandyopadhyay “*Agro-sense: Precision agriculture using sensor-based wireless mesh networks*” (2008) International Telecommunication Union - Proceedings of the 1st ITU-T Kaleidoscope Academic Conference, Innovations in NGN, K-INGN, Geneva, Switzerland, May 2008.



[43] L. Tang, K.-C. Wang, Y. Huang, and F. Gu “*Channel Characterization and Link Quality Assessment of IEEE 802.15.4-Compliant Radio for Factory Environments*” (2007) IEEE Transactions on Industrial Informatics, 3 (2), pp. 99-110.

[44] Crossbow Technology, [Online] Available:  
<http://www.xbow.com/Products/productdetails.aspx?sid=226> Accessed: May 2009.

[45] TinyOS, [Online] Available: <http://www.tinyos.net/> Accessed: May 2009.

[46] Chipcon, “*CC2420 Data Sheet*” [Online] Available:  
<http://focus.ti.com/docs/prod/folders/print/cc2420.html> Accessed: May 2009.

[47] R. Szewczyk, P. Levis, M. Turon, L. Nachman, P. Buonadonna, V. Handziski “*TinyOS Extension Proposal (TEP) 112*” [Online] Available:  
[http://tinyos.cvs.sourceforge.net/\\*checkout\\*/tinyos/tinyos-2.x/doc/html/tep112.html](http://tinyos.cvs.sourceforge.net/*checkout*/tinyos/tinyos-2.x/doc/html/tep112.html)  
Accessed: July 2009.

[48] Crossbow “*MIB Series Users Manual*” page 21, Available:  
[http://www.xbow.com/Support/Support\\_pdf\\_files/MPR-MIB\\_Series\\_Users\\_Manual.pdf](http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf)  
Accessed: July 2009.

[49] TinyOS Tutorials, [Online] Available:  
[http://docs.tinyos.net/index.php/TinyOS\\_Tutorials](http://docs.tinyos.net/index.php/TinyOS_Tutorials) Accessed July 2009.

---

## Appendix:

---

### A – Programming MicaZ motes

The TinyOS source tree for this project was located in the Linux root directory at */opt/tinyos-2.x/*. At this location a directory called *apps* contains all the application programs that can be compiled and uploaded into the ZigBee devices. In order to program the ZigBee mote, the following steps were conducted:

- The ZigBee mote must be properly plugged into the programming board, and the programming board must be connected to the computer. For this project, the board is connected to one of the USB ports.
- The programming board needs a power source. This could be from the power adaptor that plugs into the wall or from batteries connected to the mote. **Warning:** Both power sources cannot be on at the same time or the mote will be fried. If the batteries are still in the mote and the power from the adaptor is being used, then double check that the power switch on the mote is turned off. This will not interfere with uploading the application.
- Make sure that the permissions on the serial port will allow you to read and write from it. To check, open up a terminal window and use the command,

```
ls -l /dev
```

In the case of this project the name of the USB port that the programming board is connected to is called *ttyUSB0*. In order to shorten the list for the */dev* directory use the command,

```
ls -l /dev | grep ttyUSB
```

- If it is necessary to change the permissions then do this as **root user** with the command,

```
chmod a=rw /dev/ttyUSB0
```

You can double check that the permissions have properly been change by

repeating the list command.

- Now that everything is set up, the mote can now be programmed. Change to the directory where the desired application to upload is located using the terminal window. This should be

```
.../tinycos-2.x/apps/application
```

To compile and upload the program use the command,

```
make mote_type install programming_board_type,serial_port
```

Specifically for this project using the MicaZ motes, with the MIB510 programming board connected to the ttyUSB0 port, the command is

```
make micaz install mib510,/dev/ttyUSB0
```

- If it is desired to simply compile the application program to check for errors and not upload the program use,

```
make micaz
```

Of course, the full command will also display if any syntax errors are present and will not follow through with uploading to the device.

Lastly, to run the uploaded program simply, power up the mote and it will automatically initialize and start running the uploaded application.

For a full tutorial and a more in depth explanation on the TinyOS environment and programming the motes, see [49].

## **B – Configuring WLAN Settings**

To set up the WLAN card parameters so that they operate in the ad-hoc mode needed for this project, follow these steps:

- Disable and unplug any other wired or wireless network connections. The wireless USB card should be on but not connected to any WLAN access points.
- In a terminal window as **root user**, type the command,

```
ifconfig
```

to determine the system name of the wireless interface for the USB WLAN devices. It should be one of the interfaces listed on the left, such as wlan0.

- Once the proper wireless interface name is known (for this example wlan0 is used), make sure it is not connected to any access point by typing,

```
ifconfig wlan0 down
```

- Now change the parameters of the *wlan0* interface by using the *iwconfig* command. For this project three parameters are set, the operating mode, the ESSID, and the frequency channel. Set them by using the following command,

```
iwconfig wlan0 mode ad-hoc essid desired_name channel ch_number
```

As an example, for this project the following command was used,

```
iwconfig wlan0 mode ad-hoc essid pat channel 5
```

- Now all the necessary WLAN card parameters are set. Next is to set the desired IP address and turn on the device. This can be done all at once by typing,

```
ifconfig wlan0 desired_IP up
```

As an example,

```
ifconfig wlan0 192.168.0.1 up
```

- To double check that these parameters were successfully changed, simply retype the *iwconfig* and the *ifconfig* commands.

## C – ZigBee Testbed Results Chart

Table C.1: ZigBee Testbed Locations and Results.

Transmitter Location (Description)	Approx. Distance (m)	BER	PER	PLR	Max. RSSI	Min. RSSI	Average RSSI	Number of Partial Packets Dropped
<b>Indoor: BS in Kitchen</b>								
1 (Below One Floor)	3.69	0.00176	0.05527	0.024	-37	-51	-48.838	1
2 (Dishwasher)	2.00	0.00101	0.00702	0.003	-30	-47	-38.772	0
3 (Basement Fridge)	8.27	No Reception						
4 (Kitchen Fridge)	2.50	0	0	0	-27	-29	-28.745	0
5 (Fusebox)	3.50	0.00062	0.00502	0.004	-27	-29	-28.964	0
6 (Furnace)	6.08	No Reception						
7 (Hot Water Heater)	8.47	No Reception						
8 (Hydro Meter)	3.00	0.00119	0.01103	0.003	-31	-43	-41.686	0
9 (On Stove)	2.80	0	0	0	-27	-38	-29.109	0
<b>Indoor: BS at Fuse Box</b>								
1 (Dishwasher)	1.90	0.00228	0.02010	0.006	-29	-46	-40.785	1
2 (Basement Fridge)	8.50	0.00299	0.03473	0.021	-33	-50	-47.753	0

## APPENDIX

3 (Kitchen Fridge)	4.58	0.00292	0.02823	0.008	-29	-44	-42.812	0
4 (Furnace 1)	6.73	0.00197	0.03644	0.012	-35	-50	-47.868	0
4 (Furnace 2)	6.73	0.00513	0.05555	0.029	-35	-51	-48.157	1
5 (Hot Water Heater Trial 1)	8.63	0.00660	0.32110	0.136	-35	-51	-49.257	8
5 (Hot Water Heater Trial 2)	8.63	0.00638	0.11207	0.074	-36	-51	-48.640	2
6 (Hydro Meter)	2.00	0.00155	0.01515	0.010	-41	-46	-45.102	0
7 (TV)	7.30	0.00185	0.01301	0.001	-23	-31	-30.957	0
8 (Up Two Floors)	4.33	0.00243	0.02020	0.010	-33	-47	-46.021	0
9 (Washing Machine)	6.05	0.00184	0.01511	0.007	-27	-38	-36.992	0
<b>Indoor: BS at Furnace</b>								
1 (Thermostat)	4.26	0.00095	0.00903	0.003	-37	-50	-44.742	0
<b>Outdoor</b>								
	2	0	0	0	-20	-29	-23.054	0
	10	0	0	0	-31	-35	-33.311	0
	20	0	0	0	-39	-45	-43.725	0
	25	0.03134	0.79167	0.978	-48	-50	-49.318	2
(1.2m Tx Height)	25	0.00050	0.03704	0.029	-44	-49	-46.765	1
	30	0.00004	0.00502	0.004	-44	-49	-47.444	0
(1.2m Tx Height)	30	0	0	0	-42	-47	-44.640	0
	40	0	0	0	-43	-47	-44.859	0
	50	0	0	0	-44	-46	-45.101	0
	60	0.00002	0.00100	0.003	-45	-49	-47.039	0
	70	0.00010	0.00906	0.007	-46	-49	-48.048	0
	80	0.00688	0.48765	0.241	-46	-51	-49.501	10
	85	0.03819	0.96970	0.822	-48	-52	-50.820	20
	90	0.00375	0.34777	0.157	-48	-51	-49.491	11
	95	0.09005	1.00000	0.997	-51	-52	-51.333	0
<b>Vehicle Idle: Engine Off</b>								
1 (Behind Driver's Visor)		0	0	0	-6	-15	-10.476	0
2 (Behind Peddles)		0	0	0	-8	-21	-12.385	0
3 (Inside Door Handle)		0	0	0	-11	-43	-21.746	0
4 (Inside Door Handle)		0	0	0	-9	-30	-16.027	0
5 (Inside Door Handle)		0	0	0	-10	-26	-17.794	0
6 (Inside Door Handle)		0	0	0	-7	-21	-12.102	0
7 (On Centre of Dash Board)		0	0	0	-5	-11	-8.898	0
8 (In the Trunk)		0	0	0	-8	-15	-11.766	0
9 (Under Driver's Seat)		0	0	0	-9	-40	-18.677	0
10 (Under Hood, Drivers Side)		0	0	0	-23	-35	-26.287	0
11 (Under Hood, Bottom of Grill)		0	0	0	-37	-43	-39.781	0
12 (Under hood, Passengers Side)		0	0	0	-31	-33	-31.340	0
<b>Vehicle Idle: Engine On</b>								
1 (Behind Driver's Visor)		0	0	0	-8	-33	-15.565	0
2 (Behind Peddles)		0	0	0	-4	-41	-19.350	0
3 (Inside Door Handle)		0	0	0	-8	-21	-11.545	0
4 (Inside Door Handle)		0.00003	0.00050	0.002	-9	-46	-18.018	0
5 (Inside Door Handle)		0	0	0	-8	-19	-11.689	0
6 (Inside Door Handle)		0	0	0	-5	-16	-10.363	0
7 (On Centre of Dash Board)		0	0	0	-5	-12	-9.232	0
8 (In the Trunk)		0	0	0.001	-9	-19	-14.925	0
9 (Under Driver's Seat)		0	0	0	-8	-36	-13.164	0
10 (Under Hood, Drivers Side)		0	0	0	-23	-30	-26.169	0
11 (Under Hood,		0.00052	0.03711	0.030	-43	-51	-46.409	0

Bottom of Grill								
12 (Under hood, Passengers Side)		0	0	0	-27	-31	-29.126	0
<b>Vehicle Driving: Street</b>								
11 (Central to Walker)		0	0	0.003	-36	-50	-42.355	0
11 (Across City)		0.00024	0.00881	0.016	-35	-52	-42.880	1
9 (Across City)		0.00002	0.00067	0	-14	-50	-22.095	0
11 (Walker Rd.)		0.00021	0.00672	0.01	-31	-51	-36.367	1
11 (Across City)		0.00032	0.00517	0.011	-37	-51	-43.620	1
11 (Riverside Dr.)		0.00085	0.03706	0.113	-32	-52	-43.735	4
11 (Howard Ave.)		0.00025	0.01690	0.013	-32	-51	-44.320	0
<b>Vehicle Driving: Expressway</b>								
11 (Central to Lesperance)		0.00017	0.01076	0.009	-37	-51	-41.985	1
11 (Lesperance to Central)		0.00125	0.04766	0.193	-37	-52	-44.097	6
11 (Central to Lesperance)		0.00128	0.04348	0.082	-37	-52	-44.903	2
11 (Banwell to Walker)		0.00005	0.00250	0.001	-32	-51	-37.704	0
11 (Central to Banwell)		0.00025	0.01087	0.019	-37	-51	-43.200	0
11 (Banwell to Howard)		0	0	0	-32	-43	-37.357	0
<b>Machine Shop: BS in Office</b>								
1 (Head Height)	10.6	0.00022	0.00503	0.006	-35	-50	-38.354	0
2 (Shoulder Height)	11.5	0.00387	0.13726	0.493	-41	-52	-47.809	3
2 (On Light Banister: 2.4m)	11.5	0.00047	0.00804	0.005	-41	-51	-44.994	0
3 (Shoulder Height: Off)	7.6	0	0	0	-33	-46	-38.079	0
3 (Shoulder Height: On)	7.6	0	0	0	-33	-48	-36.855	0
4 (Shoulder Height)	13.6	0.00051	0.02156	0.026	-40	-51	-45.188	0
5 (Shoulder Height)	15.4	0.00119	0.05606	0.130	-40	-51	-46.939	4
6 (Shoulder Height)	10.9	0.08955	1.00000	0.976	-46	-52	-51.125	6
7 (Waist Height)	5.8	0.00010	0.00201	0.004	-33	-51	-39.376	0
8 (Head Height)	6.2	0.00423	0.07000	0.101	-35	-52	-42.795	1
9 (Head Height)	8.6	0.00220	0.04158	0.064	-36	-53	-44.581	2
<b>Machine Shop: BS in Centre of Shop Floor</b>								
1 (Head Height)	4.0	0	0	0	-26	-33	-28.839	0
2 (Shoulder Height)	4.0	0	0	0	-29	-37	-33.631	0
3 (Shoulder Height)	3.5	0.00001	0.00100	0.001	-33	-49	-39.208	0
4 (Shoulder Height)	5.6	0	0	0	-29	-37	-31.397	0
5 (Shoulder Height)	7.8	0	0	0	-33	-42	-36.819	0
6 (Shoulder Height)	8.7							No Reception
7 (Waist Height)	10.7	0.00657	0.23366	0.502	-44	-52	-48.588	7

## D – Program Codes

### D.1 – Control Centre Node

```

/*
 * ControlTerminal.java
 *
 * Created by Patrick Casey for his Thesis project, 2009.
 */

```

```
//package controlcentre;
```

```
import java.io.*;
import java.awt.Color;
```

```

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.ArrayList;
import javax.swing.text.AttributeSet;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyleContext;
import javax.swing.text.JTextComponent;

//import org.jdesktop.beansbinding.Converter;
/**
 *
 * @author fit
 */

class DisplayPkts implements Runnable {
    private javax.swing.JTextPane receivedDisplay;
    private int carPos = 0; // Caret position of previous print

    public DisplayPkts(javax.swing.JTextPane area) {
        receivedDisplay = area;
    }

    public void affix(Color colour, String text) {
        int len;

        StyleContext style = StyleContext.getDefaultStyleContext();
        AttributeSet attr = style.addAttribute(SimpleAttributeSet.EMPTY, StyleConstants.Foreground, colour);

        receivedDisplay.setCaretPosition(carPos);
        receivedDisplay.setCharacterAttributes(attr, false);
        receivedDisplay.replaceSelection(text);

        carPos = carPos + text.length(); // Increment caret position to end of current print
    }

    public void run() {
        String tempString;

        affix(Color.black, "Initializing...\n");
        byte[] localIPAddress= new byte[4]; // Local interface
        localIPAddress[0] = (byte) 192;
        localIPAddress[1] = (byte) 168;
        localIPAddress[2] = (byte) 0;
        localIPAddress[3] = (byte) 3;

        ByteArrayInputStream byteInStream = null; // Declare byte input stream
        ObjectInputStream objInStream = null; // Declare object input stream
        DatagramSocket socket = null; // Declare socket
        DatagramPacket inBuffer = null; // Declare input buffer

        ArrayList<Byte> clientData = new ArrayList<Byte>(); // Define array list (Only needed to display data)
        int nodeID, nodeData, btryLev, i; // Only needed to display data
        int numBytesInDataPkt = 7; // Number of bytes in a zigbee data packet. Change here if introduce new data

        try {
            socket = new DatagramSocket(60200, InetAddress.getByAddress(localIPAddress)); // Open a new socket on
            port number 60200 for local IP
            inBuffer = new DatagramPacket(new byte[1000], 1000); // Create receiving buffer
        }
        catch(IOException e) {

```

```

    System.out.print(e);
}
while (true) {
    try {
        while (true) {
            affix(Color.black, "Listening for packet ... ");
            socket.receive(inBuffer);          // Program hangs here until packet comes in
            affix(Color.black, "packet received: ");
            affix(Color.black, " from "+inBuffer.getAddress()+" on port "+inBuffer.getPort()+"\n");

            /* --- For displaying purposes --- */
            ByteArrayInputStream(inBuffer.getData()); // Give received packet data to Byte input
stream
            objInStream = new ObjectInputStream(byteInStream);          // Deserialize byte array back into objects
            // ^ StreamCorruptedException will be thrown here if it received a packet in response to "Get Sensor Data"
command b/c not an object
            clientData = (ArrayList<Byte>)objInStream.readObject();    // Read data from object

            i = 0;
            while ( i < clientData.size() ) {
                // Convert proper unsigned node ID (2 bytes)
                if( (Byte)clientData.get(i+1)<0 ) nodeID = ((Byte)clientData.get(i)<<8) + (Byte)clientData.get(i+1) + 256;
                else nodeID = ((Byte)clientData.get(i)<<8) + (Byte)clientData.get(i+1);

                // Convert proper unsigned sensor value (2 bytes)
                if ( (Byte)clientData.get(i+3)<0 ) nodeData = ((Byte)clientData.get(i+2)<<8) + (Byte)clientData.get(i+3) +
256;
                else nodeData = ((Byte)clientData.get(i+2)<<8) + (Byte)clientData.get(i+3);

                // Convert proper unsigned battery value (3 bytes)
                if ( (Byte)clientData.get(i+5)<0 & (Byte)clientData.get(i+6)<0 ) {
                    btryLev = ((Byte)clientData.get(i+4)<<16) + (((Byte)clientData.get(i+5) + 256)<<8) +
(Byte)clientData.get(i+6) + 256;
                }
                else if ( (Byte)clientData.get(i+5)<0 & (Byte)clientData.get(i+6)>=0 ) {
                    btryLev = ((Byte)clientData.get(i+4)<<16) + (((Byte)clientData.get(i+5) + 256)<<8) +
(Byte)clientData.get(i+6);
                }
                else if ( (Byte)clientData.get(i+5)>=0 & (Byte)clientData.get(i+6)<0 ) {
                    btryLev = ((Byte)clientData.get(i+4)<<16) + ((Byte)clientData.get(i+5)<<8) + (Byte)clientData.get(i+6) +
256;
                }
                else btryLev = ((Byte)clientData.get(i+4)<<16) + ((Byte)clientData.get(i+5)<<8) + (Byte)clientData.get(i+6);

                affix(Color.black, "From Node: "+nodeID+" Data:");
                if ((nodeData<700) || (nodeData>990)) {          // Display fault values in red text
                    affix(Color.red, " "+nodeData);
                }
                else affix(Color.black, " "+nodeData);
                if (btryLev<1000) {          // Display low battery values in red text
                    affix(Color.black, " Battery: ");
                    affix(Color.red, btryLev+"\n");
                }
                else affix(Color.black, " Battery: "+btryLev+"\n");

                i = i + numBytesInDataPkt;    // Number of bytes in each individual data packet inside the received array
            }
            affix(Color.black, " \n");
        }
        /* ----- */

        // Clear buffer and streams
        clientData.clear();

```



```

objInStream.close();
byteInStream.close();
    }
    }
    catch (StreamCorruptedException e) {
        byte[] spcldata = new byte[numBytesInDataPkt];
        spcldata = inBuffer.getData();
        affix(Color.black, "Response to special data request:\n");

        // Convert to proper unsigned node ID
        if (spcldata[1]<0) affix(Color.black, "From Node: "+((spcldata[0]<<8)+(spcldata[1]+256)) );
        else affix(Color.black, "From Node: "+((spcldata[0]<<8)+spcldata[1]) );

        // Convert to proper unsigned sensor data value
        if (spcldata[3]<0) affix(Color.black, " Data: "+((spcldata[2]<<8)+(spcldata[3]+256)) );
        else affix(Color.black, " Data: "+((spcldata[2]<<8)+spcldata[3]) );

        // Convert to proper unsigned battery value
        if (spcldata[5]<0 && spcldata[6]<0) affix(Color.black, " Battery:
"+((spcldata[4]<<16)+(spcldata[5]+256)<<8)+(spcldata[6]+256))+mV\n" );
        else if (spcldata[5]>0 && spcldata[6]<0) affix(Color.black, " Battery:
"+((spcldata[4]<<16)+(spcldata[5]<<8)+(spcldata[6]+256))+mV\n" );
        else if (spcldata[5]<0 && spcldata[6]>0) affix(Color.black, " Battery:
"+((spcldata[4]<<16)+(spcldata[5]+256)<<8)+spcldata[6])+mV\n" );
        else affix(Color.black, " Battery: "+((spcldata[4]<<16)+(spcldata[5]<<8)+spcldata[6])+mV\n" );

        affix(Color.black, " \n");
    }
    catch (IOException e) {
        System.out.print(e);
    }
    catch (ClassNotFoundException e) {
        System.out.print(e);
    }
    }
    try {
        byteInStream.close(); // Still close if went into 'catch' then loop back and start listening again
    }
    catch (IOException e) {
        System.out.print(e);
    }
    }
}

public class ControlTerminal extends javax.swing.JFrame {
    /** Creates new form ControlTerminal */
    public ControlTerminal() {
        initComponents();
    }

    /** This method is called from within the constructor to
    * initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is
    * always regenerated by the Form Editor.
    */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents
    private void initComponents() {

        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();

```

```

jLabel2 = new javax.swing.JLabel();
jSeparator1 = new javax.swing.JSeparator();
zigbeeNodeID = new javax.swing.JTextField();
jLabel3 = new javax.swing.JLabel();
sendCommand = new javax.swing.JButton();
jLabel4 = new javax.swing.JLabel();
jLabel5 = new javax.swing.JLabel();
commandType = new javax.swing.JComboBox();
jLabel6 = new javax.swing.JLabel();
timeMinutes = new javax.swing.JTextField();
startDisplay = new javax.swing.JButton();
jLabel8 = new javax.swing.JLabel();
jLabel7 = new javax.swing.JLabel();
jScrollPane1 = new javax.swing.JScrollPane();
receivedDisplay = new javax.swing.JTextPane();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jLabel1.setText("Received Data:");

jLabel2.setText("Commands:");

jLabel3.setFont(new java.awt.Font("DejaVu Sans", 0, 18));
jLabel3.setText("Control Centre");

sendCommand.setText("Send");
sendCommand.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        sendCommandActionPerformed(evt);
    }
});
sendCommand.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        sendCommandKeyPressed(evt);
    }
});

jLabel4.setText("Sensor Node ID:");

jLabel5.setText("Command Type:");

commandType.setBackground(java.awt.SystemColor.activeCaptionBorder);
commandType.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Get Sensor Data", "Sleep",
"Wake-up" }));

jLabel6.setText("Time:");

startDisplay.setText("Start Receiving");
startDisplay.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        startDisplayActionPerformed(evt);
    }
});
startDisplay.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        startDisplayKeyPressed(evt);
    }
});

jLabel8.setText("(65535 for Broadcast)");

jLabel7.setText("(1 - 71,582 minutes)");

```



```

        .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 11,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
        .addGroup(jPanel1Layout.createSequentialGroup()
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jLabel2)
        .addGroup(jPanel1Layout.createSequentialGroup()
        .addGap(35, 35, 35)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel4)
        .addComponent(zigbeeNodeID, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jLabel8))))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel5)
        .addComponent(commandType, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addContainerGap(35, Short.MAX_VALUE))
        .addGroup(jPanel1Layout.createSequentialGroup()
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(sendCommand, javax.swing.GroupLayout.Alignment.TRAILING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jLabel6)
        .addComponent(timeMinutes, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(jLabel7)
        .addGap(12, 12, 12))))
    );

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
        .addGroup(layout.createParallelGroup()
        .addContainerGap()
        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addContainerGap())
        );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
    );

    pack();
} // </editor-fold> // GEN-END: initComponents

private void sendCommandKeyPressed(java.awt.event.KeyEvent evt) { // GEN-FIRST: event_sendCommandKeyPressed

} // GEN-LAST: event_sendCommandKeyPressed

private void sendCommandActionPerformed(java.awt.event.ActionEvent evt) { // GEN-
FIRST: event_sendCommandActionPerformed

    int numBytesInCommandPkt = 7; // Number of bytes sent in a command packet from CC. Change here if
introduce new data

```

```

byte[] localIPAddress= new byte[4]; // Local interface
localIPAddress[0] = (byte) 192;
localIPAddress[1] = (byte) 168;
localIPAddress[2] = (byte) 0;
localIPAddress[3] = (byte) 3;

byte[] IPAddressToGW= new byte[4]; // Next hop interface
IPAddressToGW[0] = (byte) 192;
IPAddressToGW[1] = (byte) 168;
IPAddressToGW[2] = (byte) 0;
IPAddressToGW[3] = (byte) 2;

// Capture data in text fields
int timeToSleep_ms = 0;
int nodeID = Integer.parseInt(zigbeeNodeID.getText());
Object command = commandType.getSelectedItemAt();
if ( command.equals("Sleep") ) {
    timeToSleep_ms = Integer.parseInt(timeMinutes.getText()) * 60 * 1000;
}

// Create byte array WLAN payload
byte[] payload = new byte[numBytesInCommandPkt];

payload[0] = (byte) ( (nodeID & 0x0000FF00)>>>8 ); // Node ID
payload[1] = (byte) ( (nodeID & 0x000000FF) );
if (command.equals("Get Sensor Data")) payload[2] = (byte) 0x64; // Command Type
else if (command.equals("Sleep")) payload[2] = (byte) 0x53;
else if (command.equals("Wake-up")) payload[2] = (byte) 0x57;
payload[3] = (byte) ( (timeToSleep_ms & 0xFF000000)>>>24 ); // Time to sleep
payload[4] = (byte) ( (timeToSleep_ms & 0x00FF0000)>>>16 );
payload[5] = (byte) ( (timeToSleep_ms & 0x0000FF00)>>>8 );
payload[6] = (byte) ( (timeToSleep_ms & 0x000000FF) );

try {
    // Turn byte array into DatagramPacket
    DatagramPacket outPacket = new DatagramPacket(payload, payload.length,
InetAddress.getByAddress(IPAddressToGW), 60200);

    // Create socket
    DatagramSocket cloudSoc = new DatagramSocket(60201, InetAddress.getByAddress(localIPAddress));

    // Send packet to socket
    cloudSoc.send(outPacket);

    // Close the socket
    cloudSoc.close();
}
catch (IOException e) {
    System.out.println(e);
}
} //GEN-LAST:event_sendCommandActionPerformed

private void startDisplayKeyPressed(java.awt.event.KeyEvent evt) { //GEN-FIRST:event_startDisplayKeyPressed

} //GEN-LAST:event_startDisplayKeyPressed

private void startDisplayActionPerformed(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_startDisplayActionPerformed

    startDisplay.setEnabled(false); // Disable button (Only need to press once)
    DisplayPkts disp = new DisplayPkts(receivedDisplay);

```

```

Thread pkts = new Thread(dispatcher);           // Create new thread to display packets
pkts.start();                                   // Start the new thread
} //GEN-LAST:event_startDisplayActionPerformed

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ControlTerminal().setVisible(true);
        }
    });
}

// Variables declaration - do not modify //GEN-BEGIN:variables
private javax.swing.JComboBox commandType;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JPanel jPanel1;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JTextPane receivedDisplay;
private javax.swing.JButton sendCommand;
private javax.swing.JButton startDisplay;
private javax.swing.JTextField timeMinutes;
private javax.swing.JTextField zigbeeNodeID;
// End of variables declaration //GEN-END:variables
}

```

## D.2 – Cloud Mesh Node

```

/*
 * CloudMesh.java
 *
 * Created by Patrick Casey for his Thesis project, 2009.
 */

import java.io.*;
import java.lang.Thread;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.util.ArrayList;

public class CloudMesh implements Runnable {

    /* Thread to forward WLAN packet */
    private byte[] data;
    private String fromIP;

    public CloudMesh(byte[] buffer, String IPAddr) {
        data = buffer;
        fromIP = IPAddr;
    }
}

```

```

public void run() {
    DatagramPacket outBuffer = null;

    byte[] localIPAddress = new byte[4];
    localIPAddress[0] = (byte) 192;    // Local interface
    localIPAddress[1] = (byte) 168;
    localIPAddress[2] = (byte) 0;
    localIPAddress[3] = (byte) 2;

    byte[] IPaddressGW = new byte[4];
    IPaddressGW[0] = (byte) 192;    // Remote interface towards GW node
    IPaddressGW[1] = (byte) 168;
    IPaddressGW[2] = (byte) 0;
    IPaddressGW[3] = (byte) 1;

    byte[] IPaddressCC = new byte[4];
    IPaddressCC[0] = (byte) 192;    // Remote interface towards Control Centre (CC) node
    IPaddressCC[1] = (byte) 168;
    IPaddressCC[2] = (byte) 0;
    IPaddressCC[3] = (byte) 3;
    try {
        if ( ( InetAddress.getByAddress(IPaddressCC) ).equals(InetAddress.getByAddress("", IPaddressCC)) ) { // If from
Control Centre
            outBuffer = new DatagramPacket(data, data.length, InetAddress.getByAddress(IPaddressGW), 60200);
        // Set destination to Gateway
        }
        else {
            outBuffer = new DatagramPacket(data, data.length, InetAddress.getByAddress(IPaddressCC), 60200);
        // Set destination to Control Centre
        }
        System.out.println("Forwarding to: "+outBuffer.getAddress());

        // Create socket
        DatagramSocket cloudSoc = new DatagramSocket(60201, InetAddress.getByAddress(localIPAddress));

        // Forward 'outBuffer' to next hop
        cloudSoc.send(outBuffer);

        // Close socket
        cloudSoc.close();
    }
    catch (IOException e) {
        System.out.println(e);
    }
}
/* ----- */

public static void main(String args[]) throws IOException {

    ByteArrayInputStream byteInStream = null;    // Declare byte input stream
    ObjectInputStream objInStream = null;    // Declare object input stream
    DatagramSocket socket = null;    // Declare socket
    DatagramPacket inBuffer = null;    // Declare input buffer

    ArrayList<Byte> clientData = new ArrayList<Byte>();    // Define array list (Only needed to display data)
    int nodeID, nodeData, btryLev, i;    // Only needed to display data
    byte[] payload = new byte[7];

    byte[] localIPAddress = new byte[4];
    localIPAddress[0] = (byte) 192;    // Local interface
    localIPAddress[1] = (byte) 168;

```

```

localIPAddress[2] = (byte) 0;
localIPAddress[3] = (byte) 2;

byte[] IPAddressGW = new byte[4];
IPAddressGW[0] = (byte) 192; // Remote interface towards GW node
IPAddressGW[1] = (byte) 168;
IPAddressGW[2] = (byte) 0;
IPAddressGW[3] = (byte) 1;

byte[] IPAddressCC = new byte[4];
IPAddressCC[0] = (byte) 192; // Remote interface towards Control Centre (CC) node
IPAddressCC[1] = (byte) 168;
IPAddressCC[2] = (byte) 0;
IPAddressCC[3] = (byte) 3;

try {
    socket = new DatagramSocket(60200, InetAddress.getByAddress(localIPAddress)); // Open a new socket
on port number 60200 for local IP
    inBuffer = new DatagramPacket(new byte[1000], 1000); // Create receiving buffer
}
catch (IOException e) {
    System.out.println(e);
}

while (true) {
    try {
        while (true) { // Listens on 60200
            System.out.print("\nListening for packet ... ");
            socket.receive(inBuffer);
            System.out.println("Received from "+inBuffer.getAddress()+" on port "+inBuffer.getPort()+" with
length "+inBuffer.getLength());

            // Call new thread to handle forwarding the packet
            CloudMesh sendPkt = new CloudMesh(inBuffer.getData(),
(inBuffer.getAddress()).getHostAddress() );
            Thread forward = new Thread(sendPkt); // Call new thread giving it the payload data and dest.
IP as string
            forward.start();

            /* --- For displaying purposes --- */
            if (inBuffer.getAddress().equals( InetAddress.getByAddress("", IPAddressGW) )) { // Packet from
GW - display ArrayList
                System.out.println("Packet from GATEWAY:");
                byteInputStream = new ByteArrayInputStream(inBuffer.getData()); // Give received packet data
to Byte input stream
                objInputStream = new ObjectInputStream(byteInputStream); // Deserialize byte array
back into objects
                clientData = (ArrayList<Byte>)objInputStream.readObject(); // Read data from object

                i = 0;
                while ( i<clientData.size() ) {
                    // Convert proper unsigned node ID (2 bytes)
                    if ( (Byte)clientData.get(i+1)<0 ) nodeID = ((Byte)clientData.get(i)<<8) +
(Byte)clientData.get(i+1) + 256;
                    else nodeID = ((Byte)clientData.get(i)<<8) + (Byte)clientData.get(i+1);

                    // Convert proper unsigned sensor value (2 bytes)
                    if ( (Byte)clientData.get(i+3)<0 ) nodeData = ((Byte)clientData.get(i+2)<<8) +
(Byte)clientData.get(i+3) + 256;
                    else nodeData = ((Byte)clientData.get(i+2)<<8) + (Byte)clientData.get(i+3);

                    // Convert proper unsigned battery value (3 bytes)

```







```

}

/* --- Thread to listen to WLAN packets --- */
private PacketSource writer;
public Gateway(PacketSource usbPort) {
    writer = usbPort;
}

public void run() {
    int numBytesInCommandPkt = 7;    // Number of bytes sent in a command packet from CC. Change here if
introduce new data
    int i;
    byte[] commandPkt = new byte[13];
    byte[] payload = new byte[numBytesInCommandPkt];

    byte[] localIpAddress = new byte[4];
    localIpAddress[0] = (byte) 192;
    localIpAddress[1] = (byte) 168;
    localIpAddress[2] = (byte) 0;
    localIpAddress[3] = (byte) 1;

    try {
        while (true) {

            // Open socket on WLAN port
            DatagramSocket cloudSoc = new DatagramSocket(60200, InetAddress.getByAddress(localIpAddress));
            // Open a new socket to listen on port 60200 for local IP
            DatagramPacket inBuffer = new DatagramPacket(new byte[numBytesInCommandPkt],
numBytesInCommandPkt);    // Create receiving buffer

            // Listen on WLAN port for new packets
            cloudSoc.receive(inBuffer);
            payload = inBuffer.getData();

            /* --- For displaying purposes only --- */
            if (payload[1]<0) {
                if ( ((payload[0]<<8)+payload[1]+256) == (-1)) System.out.print("WLAN: Received command
packet for ALL nodes to");
                else System.out.print("WLAN: Received command packet for node:
"+((payload[0]<<8)+payload[1]+256)+" to ");
            }
            else System.out.print("WLAN: Received command packet for node:
"+((payload[0]<<8)+payload[1])+" to ");

            if (payload[2]==0x64) System.out.print(" retrieve sensor data.\n");
            else if (payload[2]==0x57) System.out.print(" Wake up.\n");
            else if (payload[2]==0x53) System.out.print(" be in-active for "+(bytesToInt(payload[3], payload[4],
payload[5], payload[6])/1000/60)+" minutes.\n");
            else System.out.print("...\nWLAN: Received unknown command:"+payload[2]+" . WLAN packet must
have been corrupted.\n");
            /* ----- */

            // Add packet data to AM header for tinys
            // message_t header information
            commandPkt[0] = 0x00;
            commandPkt[1] = payload[0];    // 1-2 Destination address
            commandPkt[2] = payload[1];
            commandPkt[3] = 0x00;    // 3-4 Link source address
            commandPkt[4] = 0x00;
            commandPkt[5] = 0x05;    // Message length is 5 bytes
            commandPkt[6] = 0x00;    // Group ID
            commandPkt[7] = 0x06;    // Handler

```

```

// Data fields
commandPkt[8] = payload[2]; // Type field
commandPkt[9] = payload[3]; // 9-12 Time for no-activity (= 0 if not sleep command)
commandPkt[10] = payload[4];
commandPkt[11] = payload[5];
commandPkt[12] = payload[6];

// Write to USB port
writer.writePacket(commandPkt);

// Close socket
cloudSoc.close();
}
}
catch (IOException e) {
    System.out.println(e);
}
}
/* ----- */

public static void main(String args[]) throws IOException {

    /* --- ZigBee Interface Variables --- */
    int i; // Index variable
    int offerID = 0; // Offered ID address
    byte[] offerPkt; // ID offer packet
    offerPkt = new byte[19];
    byte[] addrTable; // Address ID table. 1 = address used, 0 = address available, array index = ID address
    addrTable = new byte[256]; // ZigBee can have 2 byte long ID number. Increase if want more than 256
    int[][] seqNumTable; // Col 0 = sequence number that was used when contacting the GW about
    getting a node ID (to avoid duplication) // Col 1 = GW's returned random number in response pkt to that node ID
    seqNumTable = new int[256][2]; // ZigBee can have 2 byte long ID number. Increase if want more than 256
    boolean seqMatch = false; // True if sequence numbers match, false otherwise
    String source = null;
    PacketSource reader;
    Random randGen = new Random(); // Random number generator for GW sequence numbers
    int rand = 0; // Temporary storage for generated random number
    int numBytesInDataPkt = 7; // Number of bytes in a zigbee data packet. Change here if introduce new data

    /* --- WLAN Interface Variables --- */
    ByteArrayOutputStream byteOutputStream = null;
    ObjectOutputStream objOutputStream = null;

    DatagramSocket cloudSoc = null;
    DatagramPacket outData = null;
    ArrayList<Byte> toSendBuff = new ArrayList<Byte>(); // Data buffer to send from ZigBee packet to cloud
    network (Variable sized array)

    byte[] toSendBuff_byte; // Buffer byte array to be sent out WLAN socket (normal operation)
    byte[] toSend_byte = new byte[numBytesInDataPkt]; // Byte array to be sent out WLAN socket (for
    special data request from CC)
    int sendBuffNumFault = 0; // Number of fault packets in the "to send" buffer
    int sensorVal = 0;
    int batteryVal = 0;

    byte[] localIPAddress = new byte[4];
    localIPAddress[0] = (byte) 192;
    localIPAddress[1] = (byte) 168;
    localIPAddress[2] = (byte) 0;
    localIPAddress[3] = (byte) 1;

```

```

byte[] remoteIPAddress = new byte[4];
remoteIPAddress[0] = (byte) 192;
remoteIPAddress[1] = (byte) 168;
remoteIPAddress[2] = (byte) 0;
remoteIPAddress[3] = (byte) 2;

// Initialize Address Table and Sequence Number Table
for (i=0; i<256; i++) {
    if (i==0 || i==1 || i==255) addrTable[i] = 1;    // Reserve addresses 0, 1, and 255 (GW mote has
ID=0xBBB, nodes programmed with ID=1)
    else addrTable[i] = 0;                          // If GW has 1 < ID < 255, then this ID should also be reserved

    seqNumTable[i][0] = 0;
    seqNumTable[i][1] = 0;
}

// Process command line arguments
if (args.length == 2) {
    source = args[1];
}
else if (args.length > 0) {
    System.err.println("usage: java net.tinyos.tools.Listen [-comm PACKETSOURCE]");
    System.err.println("    (default packet source from MOTECOM environment variable)");
    System.exit(2);
}

if (source == null) {
    reader = BuildSource.makePacketSource();
    System.out.println("In null");
}
else {
    reader = BuildSource.makePacketSource(source);
}

if (reader == null) {
    System.err.println("Invalid packet source (check your MOTECOM environment variable)");
    System.exit(2);
}

try {
    int rcvdSeq;
    int origRcvdSeq;
    int ID;
    reader.open(PrintStreamMessenger.err);

    // Start new thread to listen for WLAN packets
    (new Thread( new Gateway(reader) )).start();

    while (true) {
        byte[] packet = reader.readPacket();          // Constantly wait for new packets to arrive
        Dump.printPacket(System.out, packet);        // Read packet source (USB port)
        System.out.println();                        // Prints packet to screen, separating bytes, in hex format

        // packet[8] is the packet 'type' field number
        /* --- Is a BROADCASTED ID REQUEST packet. Reply with an ID offer packet --- */
        if (packet[8] == 0x52) {
            rcvdSeq = bytesToInt(packet[9], packet[10], packet[11], packet[12]);    // Get received
sequence number (ZigBee unique node ID)
            //System.out.println("BROADCAST: rcvdSeq = "+rcvdSeq);

            // Future option: Check if requested address is available if != 0x0000

```

```

/* --- CHECK IF THIS SEQUENCE NUMBER HAS ALREADY REQUESTED A NODE ID
AND HAS ONE BEING RESERVED --- */
for (i=2; i<255; i++) { // 0, 1, and 255 are reserved
    if (seqNumTable[i][0] == rcvdSeq) {
        seqMatch = true; // This node has already entered a request
        offerID = i; // Retransmit same offered ID
        rand = randGen.nextInt(2147483647); // Generate random number between [0
,2147483647), (for always positive 2^31)
        seqNumTable[i][1] = rand; // Match with reply sequence number
        System.out.println("Re-Offer ID number "+offerID);
        break;
    }
}

/* --- FIND A NEW AVAILABLE NODE ID --- */
if (seqMatch == false) { // This is a new node
    for (i=2; i<255; i++) {
        if (addrTable[i] == 0) { // Address is available
            offerID = i;
            addrTable[i] = 1; // Mark address as taken
            seqNumTable[i][0] = rcvdSeq; // Remember sequence number sent by this ID
            rand = randGen.nextInt(2147483647); // Generate random number between [0
,2147483647), (for always positive 2^31)
            seqNumTable[i][1] = rand; // Match with reply sequence number
            //System.out.println("Generated random number = "+rand+" for ID "+i);
            break;
        }
    }
}
else {
    seqMatch = false; // Reset flag
}

/* --- PUT TOGETHER "OFFER PACKET" FIELDS --- */
// message_t header information
offerPkt[0] = 0x00;
offerPkt[1] = packet[3]; // 1-2 Destination address
offerPkt[2] = packet[4];
offerPkt[3] = 0x00; // 3-4 Link source address
offerPkt[4] = 0x00;
offerPkt[5] = 0x0B; // Message length is 11 bytes
offerPkt[6] = 0x00; // Group ID
offerPkt[7] = 0x06; // Handler
// Data fields
offerPkt[8] = 0x4F; // Type field
offerPkt[9] = packet[9]; // 9-12 Received Sequence ID
offerPkt[10] = packet[10];
offerPkt[11] = packet[11];
offerPkt[12] = packet[12];
offerPkt[13] = (byte) ((rand & 0xFF000000) >>> 24); // Random number generated from GW
for sequence number
offerPkt[14] = (byte) ((rand & 0x00FF0000) >>> 16); // >>> = unsigned shift right
offerPkt[15] = (byte) ((rand & 0x0000FF00) >>> 8);
offerPkt[16] = (byte) ((rand & 0x000000FF) );
offerPkt[17] = (byte) ((offerID & 0x0000FF00) >>> 8); // 17-18 Offered Address
offerPkt[18] = (byte) ((offerID & 0x000000FF) );

/* --- SEND OFFER PACKET --- */
if (reader.writePacket(offerPkt) != true) {
    System.out.println("UNABLE TO SEND PACKET TO USB PORT");
}
}

```

```

/* --- Is an ACKNOWLEDGEMENT to an Offer ID packet --- */
else if (packet[8] == 0x41) {

    origRcvdSeq = bytesToInt(packet[13], packet[14], packet[15], packet[16]); // Get original sequence
number (ZigBee unique ID)
    rcvdSeq = bytesToInt(packet[9], packet[10], packet[11], packet[12]); // Get received
sequence number (GW random number)
    ID = bytesToInt((byte)0, (byte)0, packet[3], packet[4]); // Get received node ID

    // Assumes new node changed its source address to the correct ID
    // If a miss-match occurs, can change code to re-offer the proper ID by finding its sequence ID
number in seqNumTable

    /* --- CONFIRM ZIGBEE NODE'S DECISION TO JOIN --- */
    for(i=2; i<255; i++) {
        if (seqNumTable[i][0] == origRcvdSeq) { // Find table entry for this original sequence
number

            if (i != ID) System.out.println("Offered-node ID doesn't match with received sequence
number. Taking no action."); // Maybe re-send offer pkt

            if (seqNumTable[i][1] != rcvdSeq) { // Node picked an ID from a different GW
                addrTable[i] = 0; // Clear entry in address ID table and sequence number table
                seqNumTable[i][0] = 0;
                seqNumTable[i][1] = 0;
                System.out.println("Cleared address table ID "+i+": Node picked a different
gateway.");
                break;
            }
            else { // Node picked this GW: Keep sequence number in case node dies and powers back
on, it can receive same ID again
                System.out.println("ACKNOWLEDGEMENT: Node "+i+" joining confirmed.");
                break;
            }
        }
        else if (i==254) { // If checked all sequence number table and couldn't find a match
            System.out.println("Received acknowledgement from a node ID "+packet[4]+" that this
GW did not send an Offer packet to.");
            System.out.println("Sequence number couldn't be found in Node table. Taking no action.");
            // Would happen if this GW wasn't present (or off) during ID REQUEST procedure with
another GW
        }
    }
}
/* --- Is a SENSOR DATA PACKET --- */
else if (packet[8] == 0x44 || packet[8] == 0x64) {

    // Convert to proper unsigned sensor value
    if (packet[10]<0) sensorVal = (packet[9]<<8) + packet[10] + 256;
    else sensorVal = (packet[9]<<8) + packet[10];

    // Convert to proper unsigned battery value
    if (packet[12]<0) batteryVal = (packet[11]<<8) + packet[12] + 256;
    else batteryVal = (packet[11]<<8) + packet[12];

    // Calculate battery level (mV) from measured reference value
    batteryVal = (1223*1024)/batteryVal; // See MIB Series Users Manual pg. 22 for this formula
description

    if (packet[8] == 0x64) System.out.print("Response to special data request: ");
    System.out.println("Sensor Data converted to decimal = "+sensorVal+" Battery level =
"+batteryVal+"mV");
}

```

```

if (packet[8] == 0x44) {
    toSendBuff.add(packet[3]);    // Node ID
    toSendBuff.add(packet[4]);
    toSendBuff.add(packet[9]);    // Sensor data
    toSendBuff.add(packet[10]);
    toSendBuff.add( (byte) ((batteryVal & 0x00FF0000)>>>16) ); // Battery data
    toSendBuff.add( (byte) ((batteryVal & 0x0000FF00)>>>8) );
    toSendBuff.add( (byte) (batteryVal & 0x000000FF) );

    // Future option: Make so CC can send command that can change fault threshold values
    if ((sensorVal<700) || (sensorVal>990) /*|| (other fault conditions)*/) {
        // Pretend fault has occurred if Light sensor reading falls below 700 or above 990
        sendBuffNumFault++;
    }
}

/* --- Conditions to send packet to cloud network --- */
if ( (toSendBuff.size()>=105)*15 packets*/* || (sendBuffNumFault>=6) || (packet[8] == 0x64) ) {
    // ^ Future option: have CC send command to change conditions for sending ^

    if(packet[8] == 0x44) {
        byteOutputStream = new ByteArrayOutputStream();           // Create streams if sending the
ArrayList object to socket
        objOutputStream = new ObjectOutputStream(byteOutputStream);
    }

    cloudSoc = new DatagramSocket(60201, InetAddress.getByAddress(localIPaddress));
    // Bind to local socket to send on port 60201

    if(packet[8] == 0x44) {
        // Convert array_list to byte_array
        objOutputStream.writeObject(toSendBuff);
        toSendBuff_byte = byteOutputStream.toByteArray();
        //System.out.println("toSendBuff.size() = "+toSendBuff.size()+" toSendBuff_byte.length =
"+toSendBuff_byte.length);

        // Make datagram packet
        outData = new DatagramPacket(toSendBuff_byte, toSendBuff_byte.length,
InetAddress.getByAddress(remoteIPaddress), 60200);
    }
    else if(packet[8] == 0x64) { // Special data request
        toSend_byte[0] = packet[3];    // Node ID
        toSend_byte[1] = packet[4];
        toSend_byte[2] = packet[9];    // Sensor data
        toSend_byte[3] = packet[10];
        toSend_byte[4] = (byte) ((batteryVal & 0x00FF0000)>>>16); // Battery data
        toSend_byte[5] = (byte) ((batteryVal & 0x0000FF00)>>>8);
        toSend_byte[6] = (byte) (batteryVal & 0x0000FF);

        // Make datagram packet
        outData = new DatagramPacket(toSend_byte, toSend_byte.length,
InetAddress.getByAddress(remoteIPaddress), 60200);
    }

    cloudSoc.send(outData);
    System.out.println("-----GATEWAY: Sending gathered sensor data to Control
Centre-----");

    if(packet[8] == 0x44) {
        sendBuffNumFault = 0;           // Reset counter
        toSendBuff.clear();           // Clear buffer
        objOutputStream.close();      // Close connections
    }
}

```





```

    call AMControl.start();    // Start up radio
}

void BroadcastRequestID() {

    /* --- SEND ID REQUEST MESSAGE --- */
    requestID* reqID = (requestID*)(call Packet.getPayload(&pkt, NULL));
    reqID->type = 0x52;        // Request type (R), R=0x52
    reqID->sentSeq = uniqID;
    reqID->reqID = 0x0000;

    if (!busy) {              // If radio is not busy
        if ( (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(requestID))) == SUCCESS ) {
            busy = TRUE;
            call Leds.led0Toggle();    // Red LED
        }
    }

    // Start timer in preparation to repeat
    call Timer0.startOneShot(3000);    // Broadcast request every 3 sec
}

event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {        // If radio started successfully
        uint8_t *point;
        struct ieee_eui64 ieeeADR;
        ieeeADR = call LocalEui.getId();    // Read unique ID from DS2401 chip
        point = (uint8_t *)&uniqID;    // Take each byte of ID and put it in packet variable
        point[0] = ieeeADR.data[7];
        point[1] = ieeeADR.data[6];
        point[2] = ieeeADR.data[5];
        point[3] = ieeeADR.data[4];

        BroadcastRequestID();
    }

    else {    // If radio did not start, try and start it again
        call AMControl.start();
    }
}

event void AMControl.stopDone(error_t err) {
}

event void Timer0.fired() {
    if (!idle) {
        atomic if (assignedAddress == FALSE) BroadcastRequestID();
        else call ReadBattery.read();    // Take battery level reading
    }
    else {    // In-active timer finished
        atomic idle = FALSE;
        call Leds.led2Off();    // Yellow LED
        call Leds.led1On();    // Green LED
        call ReadBattery.read();    // Continue sensing operation
    }
}

event void AMSend.sendDone(message_t* msg, error_t error) {
    if (&pkt == msg) { // Check that the message buffer that was signaled, is the same as the local message buffer
        busy = FALSE;    // Clear busy flag so msg buffer can be reused
    }
}

```

```

}

async event void ActiveMessageAddress.changed() {
    atomic assignedAddress = TRUE;
    call Leds.led0Off(); // Red LED
    call Leds.led1On(); // Green LED
}

void OfferAcknowledge(uint32_t seqNum) {

    /* --- SEND ACKNOWLEDGE MESSAGE --- */
    ackOffer* ackID = (ackOffer*)(call Packet.getPayload(&pkt, NULL));
    ackID->type = 0x41; // Acknowledge type (A), A=0x41
    ackID->rcvdSeq = seqNum;
    ackID->sentSeq = uniqID;

    if(!busy) {
        if ( (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(ackOffer))) == SUCCESS ) {
            busy = TRUE;
        }
    }

    call ReadBattery.read(); //Call this first then when it's done call read of sensor data. Then tx data packet
}

event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
    if(!idle) { // Do not process any received packets during idle unless it is a no action cancel command

        // Received Offer reply from gateway
        if(len == sizeof(offerID)) {
            offerID* offer = (offerID*)payload;
            if ((offer->type == 0x4F) && (offer->rcvdSeq == uniqID)) { // If proper type of msg AND received
message has same sequence number that was sent
                if(call Timer0.isRunning()) call Timer0.stop(); // Stop Timer0 if it is running
                GW_ADDR = (call Header.getHeader(msg)->src; // Get GW's mote node ID
                call ActiveMessageAddress.setAddress(0x22, offer->offeredID); // Set new Active Message
address (0x22 = group id; doesn't change)
                OfferAcknowledge(offer->sentSeq); // Call function to acknowledge offer. Return their seq number
in ack packet
            }
        }

        // Receive 'service' command from gateway
        else if(len == sizeof(service)) {
            service* serv = (service*)payload;
            if (serv->type == 0x64) { // Get sensor data command
                atomic spclData = TRUE;
                call ReadBattery.read();
            }
            else if(serv->type == 0x53) { // Be in-active command
                if(call Timer0.isRunning()) call Timer0.stop(); // Stop Timer0 if it is running
                atomic idle = TRUE;
                call Leds.led1Off(); // Green LED
                call Leds.led2On(); // Yellow LED
                call Timer0.startOneShot(serv->timeMS); // Go into idle mode for 'timeMS' milliseconds
            }
        }
    }
}

else {
    if(len == sizeof(service)) {

```

```

        service* serv = (service*)payload;
        if(serv->type == 0x57) {           // In-active cancel command
            call Timer0.stop();           // Cancel timer
            atomic idle = FALSE;          // Reset idle flag
            call Leds.led2Off();          // Yellow LED
            call Leds.led1On();           // Green LED
            call ReadBattery.read();      // Continue sensing operation
        }
    }
}
return msg;
}

event void ReadSensor.readDone(error_t result, uint16_t val) {

    /* --- READ SENSOR DATA and SEND --- */
    if(result == SUCCESS){
        senseData* dataPkt = (senseData*)(call Packet.getPayload(&pkt, NULL));
        if (spclData) {
            atomic spclData = FALSE;     // Reset flag, request has been answered
            dataPkt->type = 0x64;         // Data type (d), d=0x64
        }
        else atomic dataPkt->type = 0x44; // Data type (D), D=0x44
        dataPkt->photoData = val;
        dataPkt->btryLev = battery;

        if (!busy) {
            if ( (call AMSend.send(GW_ADDR, &pkt, sizeof(senseData))) == SUCCESS ) {
                busy = TRUE;
            }
        }

        if ((val < 700) || (val > 990)) fault = TRUE; // Future option: Make so CC can send command that can
change fault threshold values
        else fault = FALSE;

        if (!idle) { // If not in 'in-active' mode
            if (fault) {
                call Leds.led2Toggle(); // Yellow LED
                call Timer0.startOneShot(1000); // Take sensor reading every 1 sec.
            }
            else {
                call Leds.led2Off(); // Yellow LED
                call Timer0.startOneShot(5000); // Take sensor reading every 5 sec.
            }

            // It may be safer to start a Periodic timer. In case the oneShot timer fired interrupt is missed, the periodic timer
            // would fire again. If a oneShot timer interrupt is missed the node might go into a frozen state. The CC would
            // have to sleep it and wake it up again. However this would cause more processing because every time you
            // reset the timer value, you would have to stop the periodic timer then restart it again with a different value,
            // instead of simply restarting a oneShot timer.

        }
    }

    else { // Try again
        call ReadBattery.read();
    }
}

event void ReadBattery.readDone(error_t result, uint16_t val) {

```

```

/* --- READ BATTERY LEVEL SAMPLE ---*/
if(result == SUCCESS){
    battery = val;
    call ReadSensor.read();    //Read sensor data
}
else { // Try again
    call ReadBattery.read();
}
}
}

/*
* SensorRadio.h
*
* Created by Patrick Casey for his Thesis project, 2009.
*/

#ifndef SENSORRADIO_H
#define SENSORRADIO_H

enum {
    AM_SENSORRADIO = 6,
};

/* Broadcast node ID REQUEST packet */
typedef nx_struct requestID {
    nx_uint8_t type;           // Type of packet
    nx_uint32_t sentSeq;      // Sent random sequence number
    nx_uint16_t reqID;        // May request a past known ID address
} requestID;

/* Received OFFER ID packet */
typedef nx_struct offerID {
    nx_uint8_t type;           // Type of packet
    nx_uint32_t rcvdSeq;      // Unique sequence number received by GW
    nx_uint32_t sentSeq;      // New random sequence number sent by GW
    nx_uint16_t offeredID;    // New offered ID address
} offerID;

/* ACKNOWLEDGEMENT to offer packet */
typedef nx_struct ackOffer {
    nx_uint8_t type;           // Type of packet
    nx_uint32_t rcvdSeq;      // Random sequence number received by node
    nx_uint32_t sentSeq;      // Original random sequence number sent by node
} ackOffer;

/* SENSOR DATA packet */
typedef nx_struct senseData {
    nx_uint8_t type;           // Type of packet
    nx_uint16_t photoData;    // Photo sensor data
    nx_uint16_t btryLev;      // Battery level reading
} senseData;

/* Received SERVICE command */
typedef nx_struct service {
    nx_uint8_t type;           // Type of packet
    nx_uint32_t timeMS;       // Amount of time to take no action (in ms)
} service;
#endif

```

```

/*
 * SensorRadio.AppC.nc
 *
 * Created by Patrick Casey for his Thesis project, 2009.
 */

#include <Timer.h>
#include "SensorRadio.h"

configuration SensorRadioAppC {

}

implementation {
  components MainC;
  components LedsC;
  components SensorRadioC as App;
  components new TimerMilliC() as Timer0;
  components ActiveMessageC;
  components new AMSenderC(AM_SENSORRADIO);
  components new AMReceiverC(AM_SENSORRADIO);
  components LocalIeeeEui64C as Eui;
  components ActiveMessageAddressC as AMAddr;
  components CC2420PacketC as PacketHeader;
  components new PhotoC() as Light;
  components new VoltageC() as Battery;

  App.Boot -> MainC;
  App.Leds -> LedsC;
  App.Timer0 -> Timer0;

  App.Packet -> AMSenderC;
  App.AMPacket -> AMSenderC;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;

  App.Receive -> AMReceiverC;
  App.LocalEui -> Eui;
  App.ActiveMessageAddress -> AMAddr;
  App.Header -> PacketHeader;
  App.ReadSensor -> Light;
  App.ReadBattery -> Battery;
}

```

---

**Vita Auctoris:**

---

Patrick Casey grew up in Windsor, Ontario, Canada, where he attended high school at Walkerville Collegiate Institute. He then attained his B.A.Sc degree in Electrical Engineering at the University of Windsor in 2007. Patrick's research areas of interest include wireless sensor networks, power systems, and image recognition systems.