A Green Approach to a Multi-Protocol Wireless Communications Network

A Major Qualifying Project

Submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree in Bachelor of Science

in

Electrical and Computer Engineering

By

_____

Travis Collins

_____

Patrick DeSantis

_____

David Vecchiarelli

Date: 3/15/11

Sponsoring Organization:

University of Limerick:

Project Advisors:

_____

Professor Alexander Wyglinski, Advisor

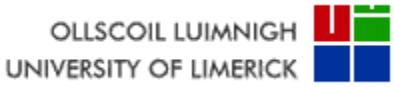
Abstract

The goal of this project is to increase the battery life of mobile wireless devices. This is achieved by having the wireless device select between two wireless protocols, ZigBee and Wi-Fi, based on transmission energy and bandwidth requirements. Using the concepts of sensing and adaptation from cognitive radio, the system monitors the bandwidth and selects the lowest power intensive wireless protocol while still maintaining an acceptable quality of service for the desired task.

Acknowledgements

Without the help from certain individuals the completion of this project would not have been possible. Some of these people have helped us by giving us guidance while others have helped us by supplying us the resources to get us through the day.

First, we would like to thank Worcester Polytechnic Institute and the Interdisciplinary and Global Studies Division for making the necessary arrangements for us to come to Ireland and because without them we would not have been given the opportunity to study in Ireland. Most of all we would like to thank Professor Alexander Wyglinski for advising our project and providing us with guidance and encouragement each week.



We would like to thank the Univserity of Limerick for providing us with a place to work everyday and the resources needed to construct our project. Special thanks to Dr. Sean McGrath for overseeing our project and acting as oour academic advisor during our time at UL. Thank you Liaoyuan Zeng “Brunt” for helping us with any questions, co-advising our project, and your speedy acquisition of materials. We would also like to thank Niall Brownen for answering any questions we had and for co-advising our project.

Finally, we thank Charlotte Tuohy, our local coordinator for the project center, for arranging and managing our housing as well as assisting us in grocery shopping each week.

Executive Summary

With the ever-increasing amount of laptops, cell phones, portable media players, global positioning systems, and other mobile devices each accessing the internet around the world, the power consumed by wireless communications systems is only increasing. According to IEEE ICC 2009 Panel on Green Communications "*currently 3% of the world-wide energy is consumed by the ICT (Information & Communications Technology) infrastructure that causes about 2% of the world-wide CO_2 emissions, which is comparable to the world-wide CO_2 emissions by airplanes or one quarter of the world-wide CO_2 emissions by cars*" [1]. Current methods for saving power in electrical devices that use radios, such as cell phones and laptops, include Power Saving Mode (PSM), sleep states, and user controlled actions like putting a cell phone in airplane mode. All of these techniques are effective at reducing the energy consumption of radios on a small scale, but even these methods do not reduce the pollution resulting from te ICT industry. One solution to this growing energy problem is to increase the battery life of mobile devices. By making batteries last longer the carbon footprint resulting from the manufacturing of batteries can be reduced.

The goal of this project was to develop a multi-protocol wireless network, that combines the energy efficiency of the ZigBee protocol IEEE 802.15.4 and the speed and high bandwidth of the Wi-Fi protocol IEEE 802.11 in order to improve the energy efficiency of current Wi-Fi only networks. The network also possesses cognitive radio attritubes that analyze and react to the surrounding radio environment. The ZigBee radios would be used only for low bandwidth applications, for which Wi-Fi would be in a power save or low traffic state. Then when additional bandwidth is needed the Wi-Fi connection is initiated. The benefits of this network are an extended battery life as well as a decreased impact on the environment, through the conservation of electrical energy.

The project was broken up into three main modules; (1) the ZigBee communication network, (2) the Wi-Fi-ZigBee switching algorithm, and (3) the power management and evaluation module. The ZigBee communication section comprised of the development of the ZigBee wireless network. This included setting up a an adhoc network between two ZigBee development board modules. Once a network was set up, a method for transferring files was constructed. Finally, the ZigBee network was integrated with the Wi-Fi network via the Wi-Fi-ZigBee Switching Algorithm. The Wi-Fi ZigBee Switching Algorithm consisted of an algorithm, written in java, that monitors the bandwidth capacity of both protocols and switches data transfer between the two. If the bandwidth reaches full capacity the algorithm turns Wi-Fi on, and data is sent and received via Wi-Fi. If the bandwidth capacity is empty, such as when the network is idle and no data transfer is taking place, the algorithm turns ZigBee on and Wi-Fi off, and any data that needs to be transferred is sent/received via ZigBee. The last block of the project was the power

management section. This section worked in parallel with the other two. While the ZigBee and Wi-Fi were switching on and off and transmitting and receiving, the power management system was monitoring the power and energy efficiency of the multi-protocol network. These measurements were then compared to measurements taken from a control experiment. The control experiment was a typical Wi-Fi only network running the same tasks. Figure 1 is the result of the power efficiency evaluation of the tested wireless networks.

An individual mobile device with a radio is not constantly transmitting. For example, a cell phone is, most of the time, resting in the pocket of the owner. The cell phone only turns its radio on to check for incoming data or to send data. To calculate how much power is consumed by the radio of the mobile device for a period of time when it is running, it is necessary to know what percentage of that time the radio spends transmitting. Figure 1 is a plot of the percentage, of a period of time, in which the radio of an Asus Eee PC is transmitting data versus the average power consumed during the entire period of time. The different radios tested were Wi-Fi (in Continuously Active Mode), Wi-Fi (in Power Saving Mode), and WiZ (Wi-Fi CAM combined with ZigBee), and are shown in the legend of Figure 1.

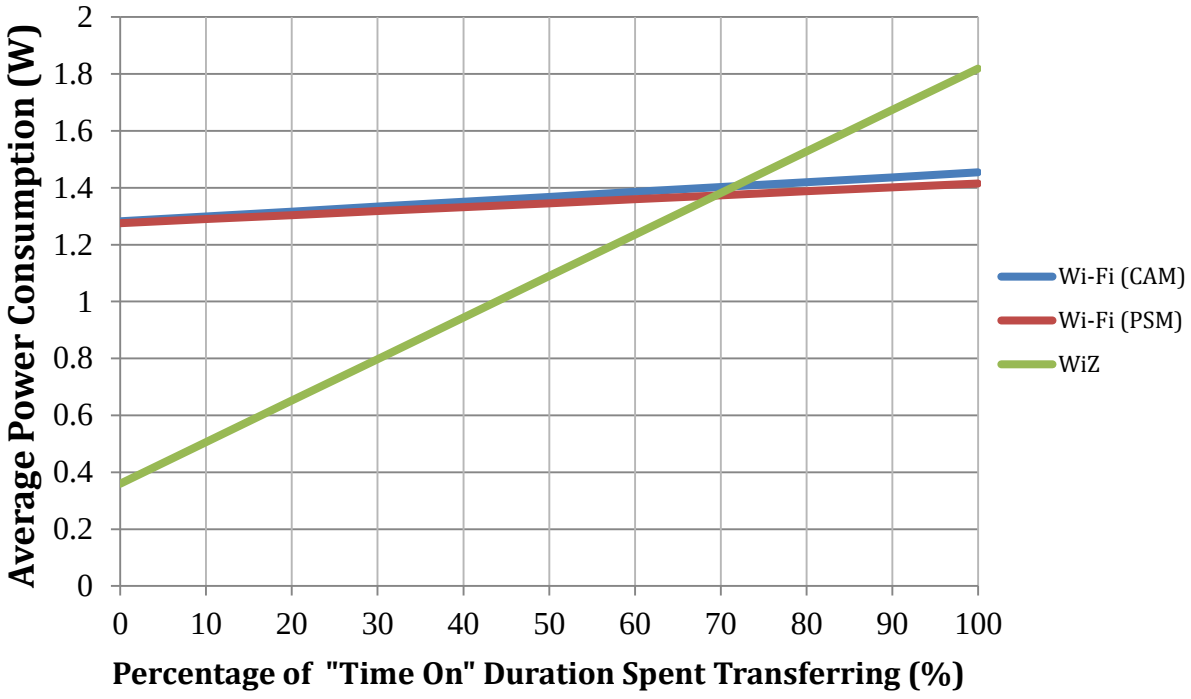


Figure 1: Power usage as a function of "time on" percentage. Plot of the percentage of time a mobile device spends transferring while on vs. the average power consumed during that setting. If a device is on and transferring for less than 70% of the time it remains on, then this graph claims that WiZ will save more energy than CAM or PSM. The device used to acquire this data was an Asus Eee PC. The Asus used a Linksys WUSB54G wireless interface adapter for Wi-Fi, and a XBee Series 2 OEM RF Module for ZigBee.

The result of this project is a multi-protocol wireless network, called WiZ, that uses ZigBee and Wi-Fi (CAM), in cooperation with a software algorithm that cognitively monitors the bandwidths of the two protocols in order to switch between the protocols based on which of the two protocols will result in the greatest power savings, while maintaining equivalent throughput. WiZ's transmissions consume 28% more power than that of the conventional wireless protocol Wi-Fi (PSM), however, the idle state of WiZ is vastly superior. By using ZigBee as its idle radio state, WiZ consumes 28% of the power consumed by Wi-Fi (PSM), while idle. By using ZigBee as its idle radio state and Wi-Fi (CAM) as its active state WiZ is able to have the equivalent data rate/bandwidth of Wi-Fi and be more energy efficient. The plot in Figure 1 shows that if a radio is turned on all day, and spends less than 70% of its time transmitting data, the WiZ network will result in power savings.

There are some future directions for this project that the team considered. One possible piece of work for the future would be to use this energy efficient multi-protocol network for other communications applications such as VoIP, web browsing, or any other communications task. This project was limited by time and resources, if more time had been provided it would have been ideal to move forward from raw data transfer to other applications such as those just mentioned. Another possibility would be to add another wireless protocol such as UWB or Bluetooth. Doing this would expand the adaptability of the network and perhaps result in a greater energy saving.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

The communications industry has seen an exponential increase in both technology and sales [2]. As a result of this ever-increasing number of mobile devices, global power consumed by these wireless communications systems has also grown significantly. According to the IEEE ICC 2009 Panel on Green Communications "*currently 3% of the world-wide energy is consumed by the ICT (Information & Communications Technology) infrastructure that causes about 2% of the world-wide CO_2 emissions, which is comparable to the world-wide CO_2 emissions by airplanes or one quarter of the world-wide CO_2 emissions by cars"* [1]. This rising need for mobility coupled with increasing demand for bandwidth, from such applications as video and music streaming, cause great stress for these wireless networks [3]. As a result, this equates to a massive amount of energy being consumed by the mobile devices themselves and the network providers transmitting data to and from the users. To keep up with such demand, transmission power needs to be increased to reach the large number of users. All of this data being transmitted from satellites, cellular towers, wireless routers, and other wireless radio devices consumes a substantial amount of power. Network providers need to find a way to meet the increasing demand for bandwidth by its mobile users while keeping the network's power consumption at a minimum. On the other hand, battery technology fails to meet the increase of power demanding applications such as video streaming and mobile videogames as a result mobile users are finding the battery life of their devices decreasing sharply [4].

Today, there has been a focus on keeping the earth "green" and avoiding excess use of resources that cause harm to the environment. In order to effectively reduce the carbon footprint created by the wireless industry it is important to look for low power solutions. One option is to employ a cognitive radio with the goal of reducing power consumption. A cognitive radio is defined as "a computer-intensive technology to balance a user's communications and computing goals against those of a variety of networks with which that user could operate" [5]. A cognitive device would be able to select the lowest power intensive wireless protocol while still maintaining an acceptable quality of service for the desired task.

One cognitive approach for power saving is already built in to many existing wireless technologies. This is in the form of power saving modes (PSM) of certain protocols [6]. These generally work by allowing the hardware to "sleep" or enter a low-power states for short intervals of time. This increases battery life without greatly effecting performance. The approach examined by this project is to intelligently use existing wireless radio protocols in cooperation with one another to reduce the power consumed by the wireless network interface of a mobile device.

1.2 Making Wireless Communications "Green"

The main focus in the wireless communications industry has been on developing protocols with higher data throughput, wireless ubiquity, and transmission range. However, this project concerns itself with the power efficiency of wireless devices. The wireless protocols used in popular consumer devices prioritize speed and range over power efficiency. For example, Wi-Fi cards are able to transmit with a relatively high data rate, but consume a substantial amount of power when active. To conserve power, Wi-Fi cards implement different states of operation. The Power Saving Mode of Wi-Fi goes into deeper effort to conserve power by switching between active and sleep state many times per second during transmission [6].

By building upon this concept of switching between states of operation this project proposes switching between entirely different wireless protocols to save energy. This goal is reached by using a much lower power consuming wireless protocol for applications that require little bandwidth. The IEEE 802.15.4 standard, which focuses on low power consumption and low data rate, was completed in May 2003. Typical applications of the protocol include industrial control, embedded sensing, home and building automation, medical data collection, smoke and intruder warning [7].

Although IEEE 802.15.4 is very low power, it has not been used for any popular consumer mobile devices such as laptops or cell phone. Consumers demand data rates capable of small applications like email access to bandwidth demanding applications such as streaming media content, so in turn higher data rate protocols such as Wi-Fi and 3G meet most of the market's needs. However, it has not been the focus of the wireless sector to develop a device that, based on bandwidth needs, intelligently switches between multiple networking protocols to take advantage of the most power efficient protocol suited for the current bandwidth need.

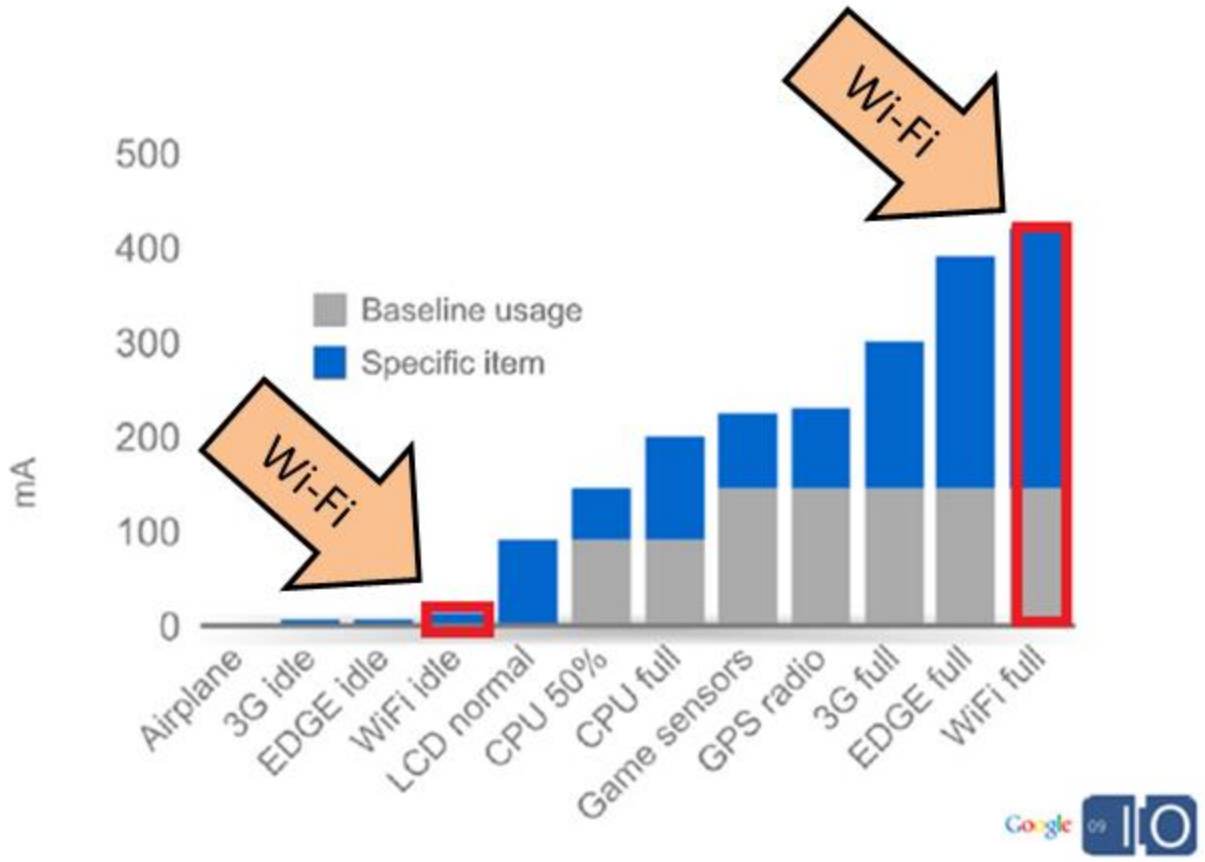


Figure 2: Average current drawn for a given task. The note to the right of the 500 mA area states "Source Values measured using an industrial power monitor at 5 kHz sampling rate, and taking average power with lowest standard deviation." From [8].

This project lays the foundation down for an effective way to reduce the energy usage of radio devices, through the use of multiple wireless protocols that cooperate in a manner that results in a smaller amount of energy consumption than current commercial wireless radio networks. Even though this project has been implemented in laptop computers, if the same wireless network were to be moved over to other mobile device platforms the battery life of those mobile radios could be extended, and the carbon footprint of battery intensive protocols could be significantly reduced. For example smartphones would benefit greatly from this project. A smartphone is a mobile phone that provides more advanced computing, internet access, and wireless connectivity capabilities, both short (Bluetooth) and cellular network connectivity [9]. The iPhone and Blackberry are examples of smartphones. At any given time a smartphone could have a GSM radio, 3G radio, Wi-Fi radio and Bluetooth radio all on and listening for data. Even though they are not actively transmitting or receiving, they still consume considerable power and reduce the battery life of the device a significant amount. For the average 3G cell phone, with the

device in "airplane mode," with no radios on, the phone draws about 2mA of power. With the 3G radio turned on and idling, the device draws about 5mA [8]. Just by having a radio on and idle, it cuts the battery life in half. With just the Wi-Fi radio on and idling, the device draws about 12mA of power. A graph of this is displayed in Figure 3. Therefore, having multiple radios on and idling at the same time draws significant power and drastically shortens the battery life of the device. The project team's green approach to a multi-protocol wireless communications network intends to solve this problem by employing a software algorithm that monitors the bandwidth of a wireless network, reacting to increases and decreases in network activity. This algorithm controls a wireless network consisting of Wi-Fi and ZigBee that uses the low power, low bandwidth protocol IEEE 802.15.4 (ZigBee) to listen for incoming bandwidth requests and low data transmissions, and switches to Wi-Fi when the network activity increases beyond a specific threshold.

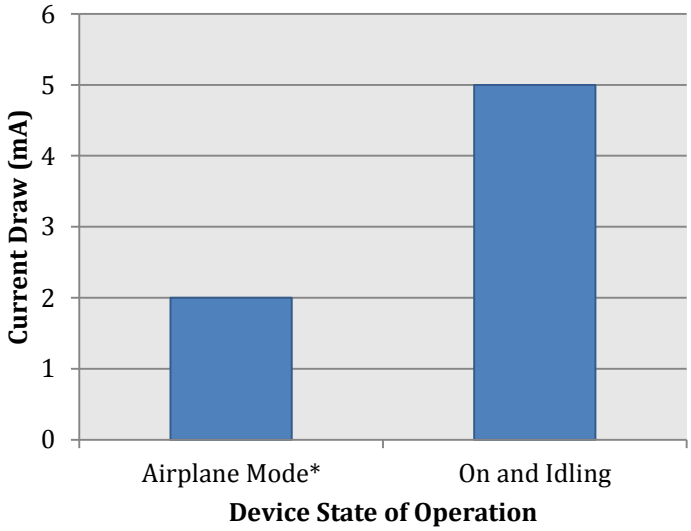


Figure 3: Average 3G Cell Phone Current Draw: *Airplane mode is an option on some smart phones that turns off all the radios. "On and idling" refers to when the 3G cell phone is turned on and the radios are connected to an AP but no data transfer is taking place. The difference between "airplane mode" and "on and idling" is that "airplane mode" has no radios on and "on and idling" has its radios turned on. The additional current drawn is the result of the radios being on. From [8].

1.3 Current State-of-the-Art

With the advent of smartphones, laptops, netbooks, and portable tablet computers, saving power to extend battery life is a hot topic of research. In terms of reducing wireless communications power consumption, there are two areas of current research. One area of research is utilizing multiple wireless protocols to achieve the lowest possible power consumption while still maintaining an acceptable

bandwidth for the current tasks. Several research projects have been conducted proposing and or implementing these multiprotocol systems.

CoolSpots is a project similar to this one that incorporates a multi-protocol wireless network that switches between two wireless protocols in order to save power [10]. The Bluetooth radio acts as the idle and default wireless protocol. When a threshold is passed, that Bluetooth is incapable of handling, the system turns on the Wi-Fi radio. CoolSpots is almost identical to the network proposed in this project except that CoolSpots uses Wi-Fi in conjunction with Bluetooth, instead of ZigBee. The advantage to using Bluetooth is that it has a higher bandwidth than ZigBee, but a much shorter transmission range and slightly higher transmission power consumption. Bluetooth was developed to replace standard wired connections. For this reason Bluetooth has a very short transmission range making much less effective than ZigBee for large wireless networks. The short range of Bluetooth restricts CoolSpots to personal area networks (PAN).

CoolSpots experimented with a number of switching policies to see which resulted in the best power savings. The policies used a number of measurement techniques, such as the measured received signal strength indicator (RSSI), transmit power, and link quality, to indirectly determine available bandwidth capacity. These measurements were taken to determine the best time to switch between wireless protocols, however none of them proved successful due to the underlying metrics not sufficiently correlating to the actual bandwidth capacity. The importance of the bandwidth capacity is that if the bandwidth of the Bluetooth is full then the system needs to switch to a higher bandwidth protocol, such as Wi-Fi. The same concept applies if the bandwidth of Wi-Fi is nearly empty, or no network activity is taking place, then the system needs to turn off Wi-Fi and turn on Bluetooth.

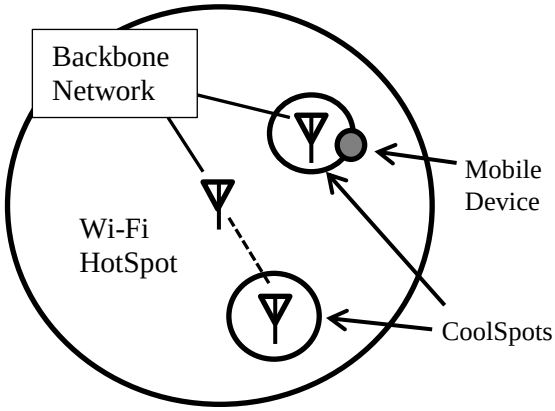


Figure 4: Multiple Bluetooth-enabled CoolSpots, inside of a traditional Wi-Fi Hotspot, allow mobile devices to connect other devices through the backbone network. CoolSpots are connected to the backbone network either directly (wired) or through the Wi-Fi___33 network (wireless). Taken from [10].

In [11] a switch agent for a multi-protocol wireless network that uses Wi-Fi and ZigBee to save energy is proposed. The primary concept is identical to this project except that the switch agent was never implemented and it focused much more on the software aspect of the system. *A Switch Agent for Wireless Sensor Nodes with Dual Interfaces: Implementation and Evaluation* also never developed a working prototype; it only produced results based on computer simulations. The main contributions of the switch agent was creating a system that is capable of activating high power radios when only necessary, reducing power through enhancements to the schemes for maintaining routing cache, and developing simulations that prove that such a network reduces energy consumption by a significant amount. The result of the simulations stated that the energy consumed in the network using the developed interface switch framework is a fraction of that consumed in a network of the IEEE 802.11 nodes and is comparable to that of nodes using only IEEE 802.15.4 radios [11].

1.4 Proposed Design and Contributions

This project aims to create an energy efficient wireless network, through the implementation two wireless protocols. The two protocols chosen optimize the power efficiency of the communications system by using a low data rate, low power consumption protocol (ZigBee) for minimal network activity and a high power, high data rate protocol (Wi-Fi) for heavy network traffic. The two wireless protocols provide power efficiency without sacrificing performance relative to a single protocol network, such as Wi-Fi. These protocols are controlled through a cognitive algorithm that monitors the bandwidth of a wireless network, reacting to increases and decreases in network activity, deciding when to turn either protocol on or off. The algorithm monitors the bandwidth of the active radios. When the wireless

network requires a fast data rate the algorithm switches Wi-Fi on, and when little to no data rate is needed the algorithm switches Wi-Fi off and ZigBee takes control.

This project overcomes the short comings in the current state-of-the-art by the following approaches:

1. Developing a greener wireless standard that performs data transfer as well as Wi-Fi.
2. This project intelligently monitors the bandwidth of multiple radios and reacts to changes in the bandwidth.
3. This project switches between two different wireless standards. The first standard is used to set up the second connection and Zigbee is used to set up wifi, so wifi is set up quicker.
4. This project is environmentally aware. It makes decisions and switches automatically based on environment, bandwidth needs, battery level, etc

This system will prove the advantages of employing a multi-protocol network to reach a specialized goal. The network design will be realized with common off-the-shelf parts. Through the use of two protocols a network can become more versatile. In this case, allowing for significant power savings without sacrificing quality of performance.

1.5 Report Organization

This report is broken into six distinct chapters. Chapter 1: Introduction, introduces the purpose of the report, starting by providing the motivation for the development of this project. Aside from motivation the introduction discusses other state-of-the-art technology currently in the market today. Next, the proposed design and contributions are explained. Chapter 2: Adaptive Wireless Transceivers is the next section and provides background information for the topics involved in the development of this project. This chapter covers 2.1 Cognitive Radio, 2.2 Commercial Wireless Standards considered for use in the creation of the WiZ network, a study on 2.3 Mobile Device Power Management, and 2.4 Green Communications. The paper then proceeds to Chapter 3: Proposed Design and Project Logistics to document the methodology used to produce the final cognitive ZigBee and Wi-Fi network. The first step in creating the network was to achieve communication between the two ZigBee nodes. This was followed by the development of a local interface between the Wi-Fi and ZigBee. The third step in the process was to successfully transfer data between the two computers using the combined ZigBee-Wi-Fi network. After this, the next step was to configure the network so that Wi-Fi would only activate when the need for a larger bandwidth was required for sending and receiving data, this is the cognitive switching algorithm. Finally, the last step of the methodology was to monitor and measure the current, power, and energy consumption of the system in order verify that it resulted in an energy saving.

After the design, the report transitions to Chapter 4: Implemetation, this describes the process followed to develop the WiZ network. The WiZ network is broken into three sections; 4.1 Setup and Installation of Equipment, 4.2 Software Radio Controller, 4.3 Power Measurement System. These sections were developed in three separate partitions and combined upon the completion of each partition. The first section is the 4.1 Setup and Installation of Equipment, which provides a step-by-step guide for the implementation process and construction of the Wi-Fi portion of the WiZ radio network. The second section is the 4.2 Software Radio Controller. This consists of the file transfer software created to act as the method of transferring data for this project since ZigBee has no pre-existing firmware to transfer data. It then proceeds with the implementation of the program that performs the cognitive aspect of the WiZ radio network, which switches the Wi-Fi and ZigBee radios on and off depending on the network activity of the WiZ system. The third and final section of the implementation is the 4.3 Power Measurement System portion. This section provides information for reproducing the methods used to monitor the power consumption of the wireless networks considered in this project. The networks evaluated are Wi-Fi (CAM), Wi-Fi (PSM), ZigBee, and the WiZ wireless networks. The next chapter of the report, Chapter 5: Results, analyzes the experimental results and also provides detailed documentation of how each experiment was performed. The experimental results are compared to the power usage of a standard Wi-Fi networks and a conclusion regarding the effectiveness of the Wi-Fi-ZigBee (WiZ) network is formulated. The last chapter of the report, Chapter 6: Conclusions and Future Work, discusses possible future work related to this project.

Chapter 2: Adaptive Wireless Transceivers

Before designing a multiprotocol green energy communications system, several areas had to be researched to gain an understanding of the underlying technologies and the state-of-the-art research. Cognitive radios and software defined radios were studied to gain an understanding of how radios can intelligently adapt to the environment. The current state-of-the-art in cognitive green communication systems were investigated next. Then, several mainstream commercial wireless standards were surveyed to decide which ones to implement.

2.1 Cognitive Radio

The concept of the cognitive radio (CR) was first conceived in 1998 by Joseph Mitola III and Gerald Maguire Jr. The plan was to develop a radio with the ability to sense the different frequency bands available, determine which spectrum band would be best suited for the intended application, and do this all seamlessly without any input from the user. Mitola defined cognitive radio as "a computer-intensive technology to balance a user's communications and computing goals against those of a variety of networks with which that user could operate" [5]. In other words, a cognitive radio is able to observe, learn, and react to changes incurred by users in order to more effectively achieve its goal.

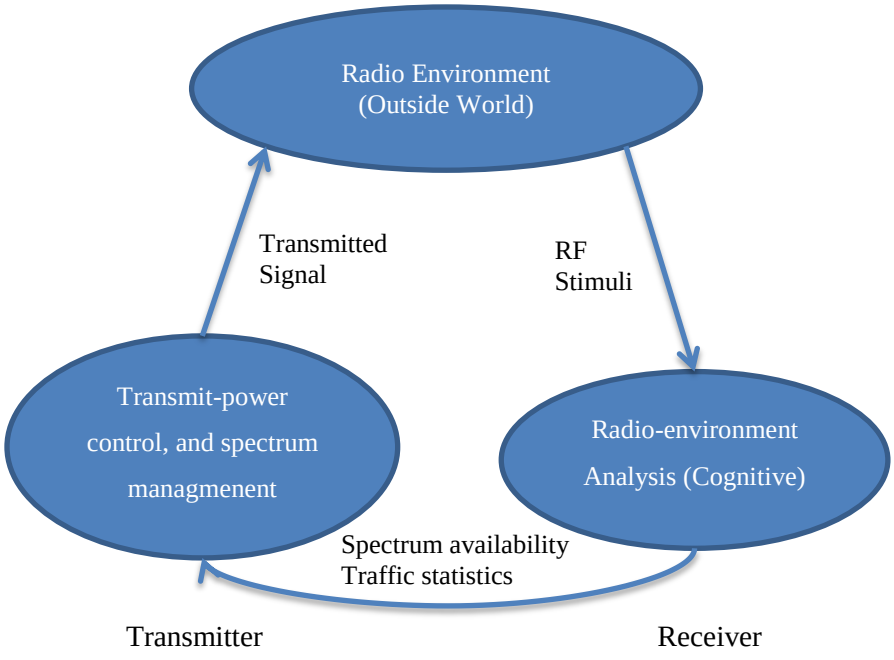


Figure 5: Basic cognitive radio cycle. The radio environment is the radio waves in the air. The radio antenna receives signals in the air, the signals are then analyzed by the cognitive intelligence and a response is formulated by this intelligence. Possible parameters that may be changed are the transmission power and spectrum band frequency. Adapted from [12].

However, today cognitive radio encompasses many different technologies which enable radios to perform several roles such as automated radio resource optimization and dynamic spectrum access. Dynamic spectrum access, or spectrum sensing, has been studied to alleviate the spectrum scarcity problem in the United States [13]. However, since this project focuses on power efficiency the concept of automated radio resource optimization will be explored in greater detail. In general, cognitive radios are capable of adapting to the available transmission parameters in order to achieve a specific performance goal by combining several adaptation techniques to form a decision making engine with several dimensions of transmission control [12] [13] [14] [15] [16]. There are many parameters that can be taken into account to reconfigure and improve the power performance of radios. The most common parameters that cognitive radios reconfigure are spectrum frequency and transmission power. Radio resource optimization intelligently adapts to the environment through the monitoring of these parameters and changes their operation according to some goal driven algorithm [14]. In theory this ability results in significant improvements in the performance of the system. Some possible parameters radio resource optimization addresses are transmit power, modulation, system throughput, and bit error rate (BER) [14]. Examples of radio resource optimization exist in present-day technology, the IEEE 802.11 wireless standard employs adaptive modulation to monitor the signal-to-noise ratio (SNR) of the communications signal and adjusts the power and modulation in a manner that results in the best possible throughput [14].

By definition cognitive radios are aware of their environment and react to them accordingly. A great amount research has been put into cognitive radio, the major difference between each research topic is the cognitive method employed that makes the decisions. Most of these cognitive decision making techniques are algorithms of varying complexities, ranging from simple algorithms that have preset reactions for specific situations, to complex cognitive infrastructures similar to Artificial Intelligence (AI) [10] [11] [12] [15] [17]. Some development has been conducted that aims to create a radio that has the ability to learn. This method is generally called machine learning. Machine learning uses math based algorithms that enable radios to remember lessons learned in the past and act quickly in the future [15]. One such approach to implement machine learning is through Genetic Algorithms [17]. Genetic Algorithms define a radio by a chromosome, the genes of the chromosome represent the parameters in a radio that can be adjusted, and by modifying the chromosome genes, the genetic algorithm can optimize the radio to meet the current user's needs. The algorithm is meant to mimic the behavior of DNA, and as such the program "evolves", or learns. A selection mechanism determines whether or not a chromosome will survive from generation to generation. This selection is based on an evaluation function that determines the fitness of the chromosome. Specifically, the Wireless System Genetic Algorithm (WSGA) [17] is a method to utilize cross-layer optimization and also a method of adaptive waveform control. In the WSGA, radio behavior is represented by traits encapsulated in the genes of a chromosome. Other

radio parameters are also included as possible genes in the chromosome for evolution and growth purposes.

2.2 Commercial Wireless Standards

In order to maximize energy efficiency and power savings, several common commercial wireless standards were considered for possible implementation in the project. Commercial wireless standards were chosen because they are easy to acquire, inexpensive, and substantial documentation is available. The goal of this research project was to not to come up with a new, energy efficient wireless standard, but rather to modify current commercial wireless protocols in a way to achieve greater power savings.

Wi-Fi Background

The most widely implemented and unlicensed form of radio communication is Wireless Fidelity, or more commonly known as Wi-Fi [18]. This is the protocol classified under the overarching IEEE 802.11 standard similar to the ETSI European standards for broadband radio access networks that include such protocols as HiperLAN and HiperLAN 2 [19]. Since IEEE 802.11's ratification in 1997 it has become the most developed wireless technology in the world [20]. The Wi-Fi IEEE standard includes several revisions, including 802.11a, 802.11b, 802.11g, and 802.11n. As of 2010 the number of wireless devices using the standard has been growing at rate of 2.2 million per month [21]. Figure 6 below displays the global hotspots alone for Wi-Fi access, reaching across most of the developed world.

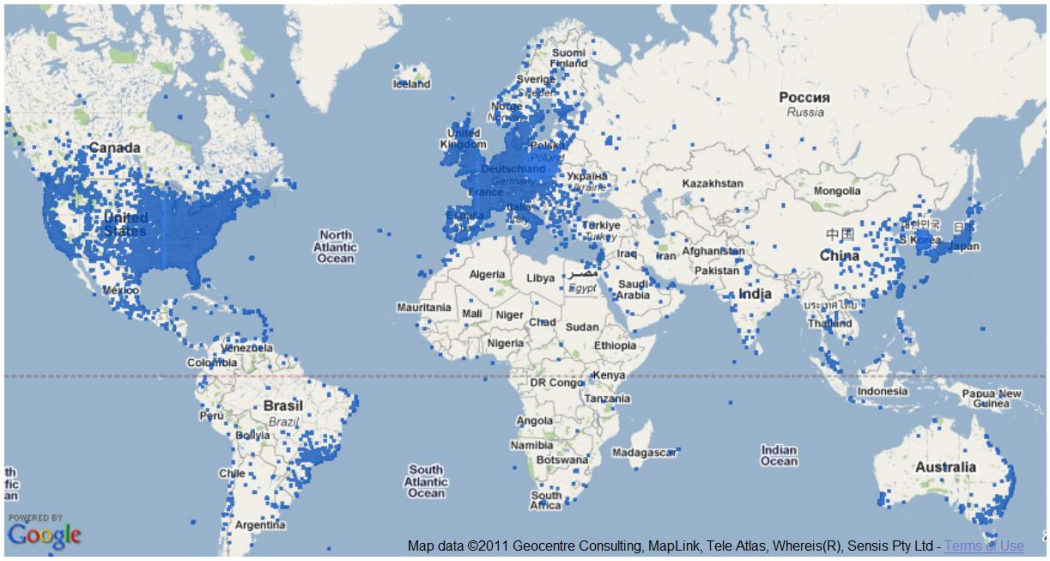


Figure 6: Global Wi-Fi Deployment. Blue dots indicate Wi-Fi access points and cell towers. Taken from [22].

Wi-Fi was born out of the deregulation of certain radio-frequencies which occurred on May 9, 1985 for unlicensed spread spectrum use. This ruling stipulated the use of unlicensed radio in the 902-928, 2400-2483.5 and 5725-5850 MHz bands on a noninterference basis to other authorized users of these

bands. The opening of this band paved the way for a large amount of radio technologies which evolved into the IEEE 802.11 standard today. These bands at 900MHz, 2.4GHz and 5.8GHz, were initially allocated to equipment that used radio-frequency energy for purposes other than communications, such as microwave ovens. However, the Federal Communcations Comission (FCC) made these bands available for communications purposes as well, with regulations put in place that made sure that there would be no interfere between other bands and among the same bands themselves. The FCC did this by using spread spectrum, which spreads a radio signal out over a wide range of frequencies, in contrast to the usual approach of single carrier transmission, making the signal less susceptible to interference [23]. However, Wi-Fi only was able grab hold because of the creation of an industry-wide standard. Initially, vendors of wireless equipment for local-area networks, such as Proxim and Symbol, developed their own kinds of proprietary equipment that operated in the unlicensed bands. As a result, equipment from one vendor could not talk to equipment from another. Inspired by the success of Ethernet, a wireline-networking standard, several vendors realised that a common wireless standard should be realized. Consumers would be more likely to adopt the technology if they were not locked in to a particular vendor's product. Therefore, in 1988 Victor Hayes, along with Bruce Tuch of Bell Labs, approached the Institute of Electrical and Electronics Engineers (IEEE), where a committee called IEEE 802.3 had defined the Ethernet standard. A new committee called IEEE 802.11 was constructed to develop a similar standard for Wireless networks. In 1997, the committee agreed on a basic specification, this specification allowed for a data-transfer rate of two megabits per second, using either of two spreadspectrum technologies, frequency hopping or direct-sequence transmission.

As Wi-Fi developed it grew into several subprotocols within the IEEE 802.11 standard. The most prominent modes include IEEE 802.11 a, b, g, and n. Currently, Wi-Fi has a maximum data rate of 54 Mb/s for 802.11g and 150 Mb/s for IEEE 802.11n, and a typical range of 100 meters (inside) and 300 meters (outside) [24]. Additional details can be seen on the published versions of Wi-Fi in Table 1.

Table 1: Wi-Fi Protocols. Taken from [25].

IEEE 802.11 Protocol	Release	Freq. (GHz)	Bandwidth (MHz)	Data rate per stream (Mbit/s)
–	Jun-97	2.4	20	1, 2
a	Sep-99	5	20	6, 9, 12, 18, 24, 36, 48, 54
b	Sep-99	3.7	20	5.5, 11
g	Jun-03	2.4	20	6, 9, 12, 18, 24, 36, 48, 54
n	Oct-09	2.4	20	7.2, 14.4, 21.7, 28.9, 43.3, 57.8,
		2.4/5	40	65, 72.2
				15, 30, 45, 60, 90, 120, 135, 150

The instances of Wi-Fi shown in Table 1all share a common transmitter and receiver design, varying slightly depending on modulation scheme, throughput, and several other factors. Generally, most digital wireless communication systems follow this generalized design represented in the figure below because of its simiplity and robustness. The overall goal of this system is to send data efficiently and with as little error as possible. In order to prepare data for wireless transmission the communication system data must first passthrough several functional blocks. This begins with some degree of sources encoding, which removes redundencies with the source data itself. Then this data undergoes channel encoding which introduces control redundancies to help minimize errors when passing data through the channel. Next, the signal is modulated with a carrier frequency up to RF. Finally, the signal reaches the analog domain from the analog to digital converter and is projected as electromagnetic radioation through the RF frontend and into the channel to the receiver. This same process happens in reverse at the receiver. This is a very generalized explaination of a digital communication system, when in reality additional complex system are needed to compensate for non-idealities such as channel distortion and carrier frequency offset.

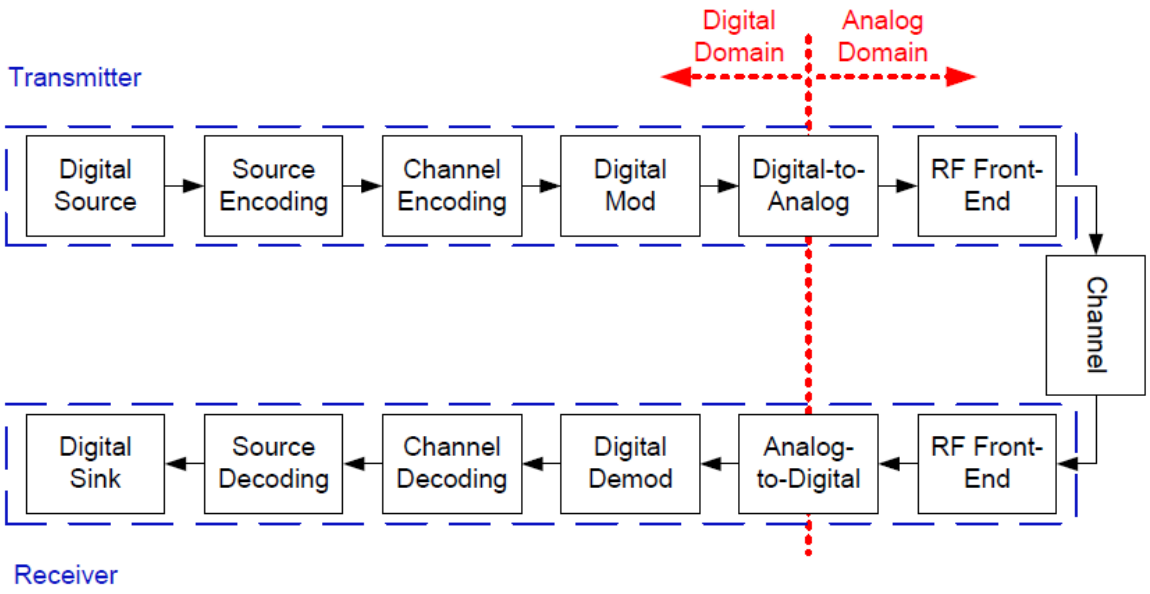


Figure 7: General Wireless Receiver Transmitter pairing [26]

Along with the functional blocks seen above Wi-Fi has five different eneryconsumption states that effect the operation of these functions. Each states operates the device for different purposes and as a result in different power draws. In current Wi-Fi interface cards, these energy states are dynamically adjusted to help save power, primarily when the card is in a rather idle state [27]. The five different states of operation are:

- Active Receive: The radio is listening and deciphering packets from surrounding nodes.
- Transmit: The radio is broadcasting data.
- Sleep: Certain radio functionality is turned off to save power. There are two different levels of sleep. One is the deepest sleep mode, which turns off the oscillator and voltage regulator. The other is light sleep mode, which keeps these components energized [27].
- Idle/Listening: The radio is waiting for an event from the user or surrounding network. This is usually to maintain association with access point.
- Off: The radio is fully powered off.

Each of these energy states have consequences that impact the overall radio's energy consumption and as a result effect the overall energy consumption of a mobile wireless device. When the radio is active transmitting it is using the most power, but it also does use a considerable amount of power just in an idle or sleep state. These sleep states must wake up in predetermined intervals to examine the surrounding spectrum for available packets. The best possible scenario would allow the card to be completely off, using no power, until it actually needs to transmit or receive. This would completely eliminate the need for sleep states. However, since the card cannot inherently know when it needs to be used current MAC protocols put the radio in sleep mode while there is no data to send or receive, in order to minimize energy consumption. Wi-Fi relies on a static low power mode, which involves an energy and delay tradeoff. The deepest sleep mode provides the lowest current draw of all low power modes. However, it also involves the highest energy cost and the longest latency for switching the radio back to active mode. In contrast, the lightest sleep mode provides a transition to active mode that is quick and energy inexpensive, but this mode has a higher current draw. Sleep mode switching is generally determined by the amount of traffic during a given period of time. The more traffic the less often the card will be put into deep sleep mode [27]. These sleep modes are generally hardware dependent, meaning that they can differ from card to card.

Table 2: Wi-Fi Sleep Modes.

Mode:	Purpose:
Sleep	Short latency, least power advantage
Deep Sleep	Long latency, best power advantage, most hardware element powered off

On the higher networking level when multiple nodes exist, several network configurations can exist with the Wi-Fi standard. These configurations are ad-hoc and infrastructure networks. Ad-hoc networks are point to point networks, which are primarily used in decentralized networks. They have the distinct advantage of node independence, meaning that in general one node cannot disable the entire

network, but can affect it significantly depending on location. Infrastructure networks on the other hand contain a certain router or access point in which all traffic passes through. Interactions among nodes generally follow the same process with minor variations. First of all, nodes in some manner or interaction must associate themselves with the network, which is generally done with a beaconing system. After, association authentication must be provided at some level and then a link synchronizes with the network. All of these processes happen at the lower layers of the OSI model and are generally considered independent from the computer and end-user, beyond what network to associate yourself with [28]. For example, when a connection to your home wireless access point is made, most of the connection and synchronization work is solely done by the networking card, with no help from the user or operating system.

Control of most aspects of a wireless interface are completely autonomous and completely tranparent the user. To the user most of the hardware control has been abstracted for the sake of simplicity. Due to the complexity of the wireless interface, this control would be rather overwhelming to the user. Therefore, this simplified perception is extremely powerful because it allows user applications to focus on higher layers without worrying about how the lower layers will react. The most import aspect of this domain is its independence from hardware. For example, this means that any system that contains this internet protocol can communicate with that system [28].

Another example of this abstraction is point to point data transmissions among computers network through a communication flow method known as sockets. The term sockets is used as a name for an application programming interface for the TCP/IP protocol stack, usually provided by the operating system. This means that these sockets are completely managed or mapped by the operating system. Sockets constitute of a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote internet protocol addresses and port numbers. Therefore, these sockets allow systems to synchronize processes on remote systems and communicate in a fully duplexed manner. This entire functionality is abstracted from the hardware. The socket is primarily a concept used in the Transport Layer of the OSI. Networking equipment such as routers and switches do not require implementations of the Transport Layer, as they operate on the Link Layer level or at the Internet Layer. This method of using sockets is used as the foundations of most file transfers in computer systems today.

IEEE 802.15.4: ZigBee

ZigBee is a low power, low bandwidth wireless radio standard targeted towards applications requiring low data rate and extremely low power consumption and is designed to have low cost and very simple setup and integration [29]. ZigBee devices typically have a range of around 100 meters, and a

single ZigBee network can contain up to 65,536 devices. Usually, ZigBee is used for commercial devices such as automated lights or appliances in houses. This project examines ZigBee for its low power usage capabilities.

One of the strengths of the ZigBee protocol is its low power consumption for wireless communications. The average transmit power of a ZigBee radio ranges from 0.001mW to 0.003mW. On an alkaline cell battery, an average ZigBee radio will be able to remain powered for two years, assuming routine data transfer [29]. It accomplishes such low energy usage by minimizing the amount of time the radio is on. Bluetooth and Wi-Fi radios spend more time awake and, therefore, drawing more power. On the other hand, the ZigBee protocol minimizes the amount of time the radio needs to be on, reducing power as much as possible. The proportion of time the radio is active to the time radio is asleep is defined as the radio's duty cycle. ZigBee is optimized for very low duty-cycle operations. For some applications the duty-cycle can drop below 0.1% for maximum power conservation. The biggest drawback of ZigBee is its low data rate. Other wireless communications technologies, such as Bluetooth or Wi-Fi, prioritize higher data rates at the cost of higher energy costs. ZigBee, meanwhile, purposefully keeps its data rate low. It has a maximum theoretical throughput of 250 kilobits per second, as opposed to 1 megabit per second for Bluetooth and 600 megabits per second for Wi-Fi IEEE 802.11n. By keeping the data rate low, ZigBee radios consume only a fraction of the power that Bluetooth or Wi-Fi radios consume.

Another unique ability of ZigBee devices is that they are optimized to quickly and efficiently join networks as well as change to and from sleep modes. A typical ZigBee end device takes on average 30 milliseconds to join a network, and 15 milliseconds to and from active and sleep mode [30]. For comparison, Bluetooth devices on average take 20 seconds to join a network 3 seconds to change to and from sleep modes. For an operation of joining a network, transferring a small amount of data, and then going into sleep mode, a Bluetooth device will require about one hundred times the amount of energy to complete this operation [29].

ZigBee is capable of multiple network profiles. To effectively use ZigBee it is necessary to understand these profiles and they strengths and weaknesses. ZigBee network profiles are capable of both beacon and non-beacon enabled networks. In non-beacon networks, ZigBee routers constantly have their receivers active and listening. The radio on the end device can remain off until it needs to transmit data. When a data transmission is required, the radio wakes up, sends its transmission, receives an acknowledgement, and then returns to sleep. The advantage of this is that no power is used until transmissions are required. However, the disadvantage is that the router needs to remain constantly on and that the end device cannot receive messages. In beacon-enabled networks, routers transmit periodically transmit beacons to confirm their network status to other nodes in the network. Since

beacons are transmitted at a specified time interval, devices may sleep in between beacons to lower their duty cycle.

To understand how the ZigBee networks work three types of logical devices must be explained; coordinators, routers, and end devices. Every network must have a coordinator; coordinators create the network by selecting a personal area network identifier (PAN ID) and a channel. In secure networks, the coordinator also contains the trust center and a repository for network keys. Any device trying to join the network needs to be authenticated by the trust center. Routers can relay messages to other nodes, including the coordinator, other routers, and end devices. They can be used to extend the network coverage to areas outside of the coordinator's range. Routers also add redundancy to the network so that in the event that one router gets disconnected, powered off, overwhelmed with traffic, another router within range can take over. Finally, devices looking to join the network do not need to connect directly to the coordinator; instead they can connect to the router closest to it. End devices can only talk to one other device, its parent device. They cannot route data to other nodes. In terms of physical hardware, there are two physical ZigBee device types; a full function device (FFD), and a reduced function device (RFD). Full function devices are capable of being coordinators and routers and reduced function devices are limited to being just end devices. Reduced function devices have a reduced stack size, which translates to them requiring less memory and therefore being cheaper to produce [31].

There are three types of networks that the ZigBee protocol supports as well. The first is a star network. It consists of a coordinator and multiple end devices connected to the coordinator, and is the simplest network to form in that it does not need any routers. All messages pass through the coordinator and then to their destination. An example of a star network is shown below in Figure 8.

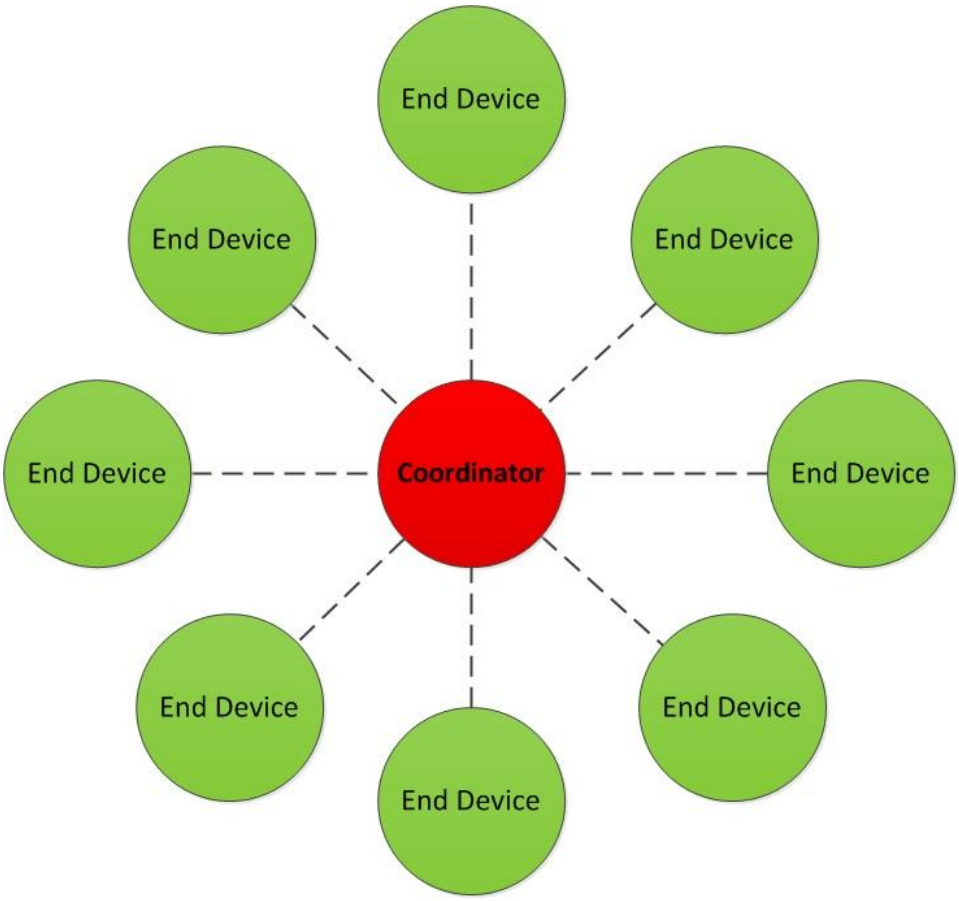


Figure 8: ZigBee Star Network [32].

The second type of network ZigBee can form is a tree network. A tree network consists of a coordinator as the top node with a branch and leaf structure below it. Routers are connected to the coordinator and to one or more end devices. Messages travel up the tree as far as necessary, and then back down it to reach their destination. An example of a tree network is shown below in Figure 9.

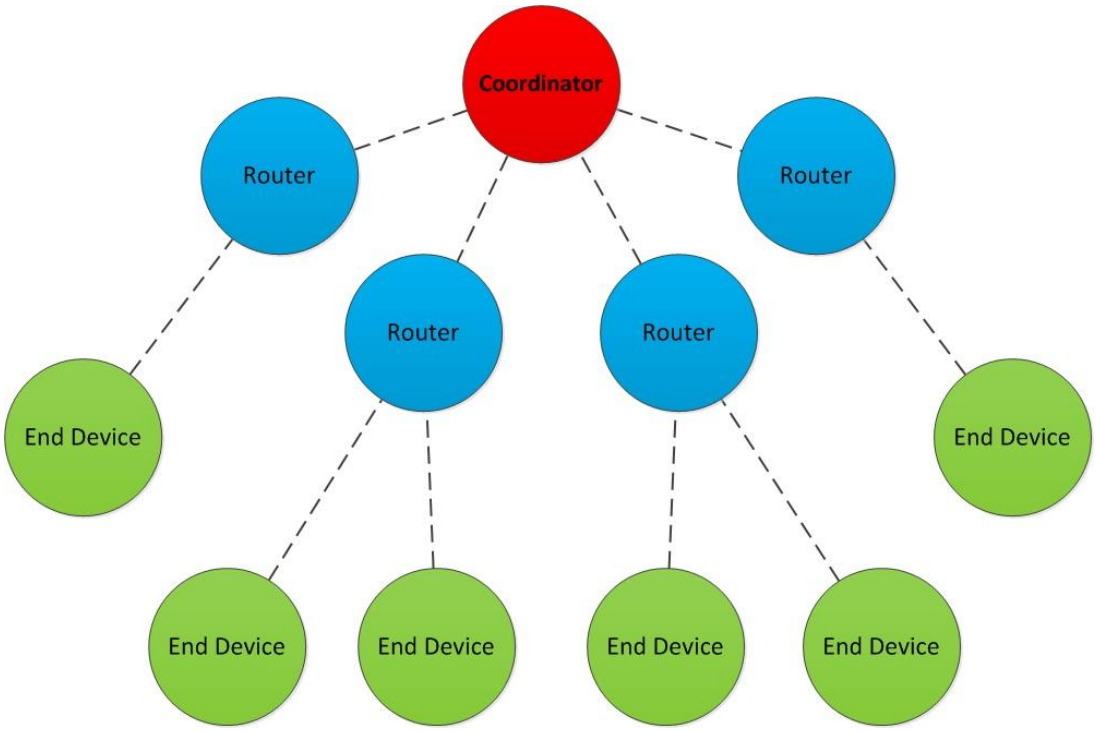


Figure 9: ZigBee Tree Network Topology [32].

The third type of network ZigBee can form is a mesh network. A mesh network consists of a coordinator, routers, and end devices interconnected to each other. There are multiple pathways to reach each node. Connections are updated and optimized dynamically through routing algorithms. An example of a mesh network is shown below in Figure 10.

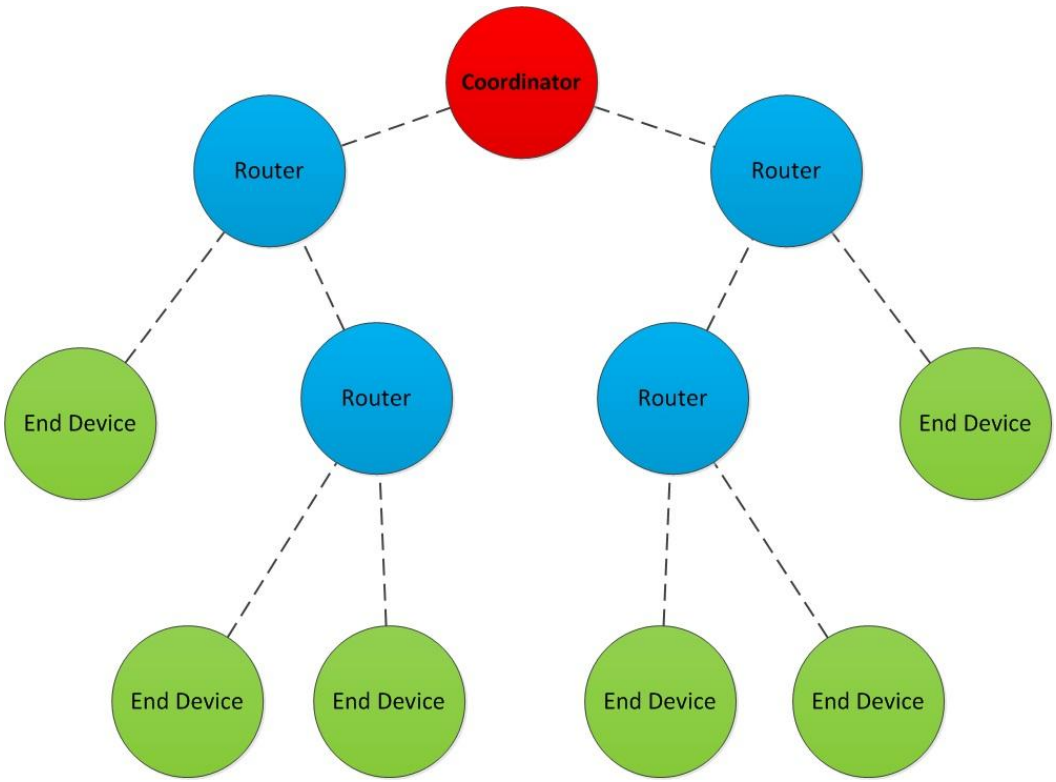


Figure 10: ZigBee Mesh Network Topology [32].

Common applications of ZigBee include industrial control, embedded sensing, home and building automation, medical data collection, and smoke and intruder warning. It is not typically included in mobile consumer communication devices such as cellular phones, laptops, or headsets since it has a very low data rate and does not currently integrate with IP technologies [7]. However, the ZigBee Alliance has formed an "Internet Solutions Initiative" to investigate ways of integrating IP networking into ZigBee. The group aims to "make it easier for developers and system integrators to deploy ZigBee and to add additional features and functions, including IPv6 support" and will allow continued growth of smart grid applications [33], [34]. This research is supported by many leading electronics manufacturers, including Texas Instruments.

Comparison and Selection of Protocols

After surveying the available commercial wireless standards, it was determined that they are all suitable candidates for a multiprotocol green energy communications system. They all have their own strengths and weaknesses. Of them, ZigBee uses the least power but has the lowest data rate. Wi-Fi uses the most power, but also has the highest data rate. Bluetooth was also considered however, since integrating Wi-Fi with Bluetooth to lower communications power has already been proven by other projects such as CoolSpots, it was decided that this implementation would just integrate Wi-Fi and ZigBee. Furthermore, ZigBee has many advantages over Bluetooth, such as a longer range and a ZigBee

network can also contain 65,536 devices, as opposed to the 8 devices in a Bluetooth network. Table 3 summarizes these results. A final implementation of a multiprotocol green energy communications system could incorporate all three protocols. However, this project is just a proof-of-concept with limiting time constraints.

Table 3: Wireless radio comparison table. The table lists the factors deemed important to the project. Scalability was considered due to the desire of wireless ubiquity. Data taken from [6] [10] [35] [36] [37].

Wireless Protocol	Data Rate	Range	Scalability (Max number of cell nodes)	Power (W)
ZigBee	250 Kbits/s	100 m	>65000	0.085
IEEE 802.11a	54 Mbits/s	100 m Inside, 300m Outside	8	1.3
IEEE 802.11b	11 Mb/s	100 m Inside, 300m Outside	8	1.3
IEE 802.11g	54 Mb/s	100 m Inside, 300m Outside	8	1.3
IEEE 802.11n	150 Mb/s	100 m Inside, 300m Outside	8	1.3

2.3 Mobile Device Power Management

The purpose of power management is to avoid excess consumption of power in electrical devices. This practice has recently taken the spotlight due to the boom of interest in environmental dilemmas such as global warming. Power management for computers is an attractive feature for reasons other than environmental impact as well. A power-conscience PC will see benefits such as longer battery life, less heat, and lower power consumptions resulting in lower costs. Heat is one of the biggest problems machines experience, it can often result in component failure and a reduction in overall system performance. Decreasing the heat generated by a device also lessens the need for extraneous cooling systems, such as additional heat sinks, fans, and liquid cooling.

However, in the realm of mobile wireless radio devices the most appealing aspect of power management for low power consumption is longer battery life. According to a study conducted on a Toshiba 410 CDT mobile computer the typical power consumption of a computer is that 36% of the power is consumed by the display, 21% by the CPU and memory, 18% by the wireless interface, and 18% by the hard drive [36].

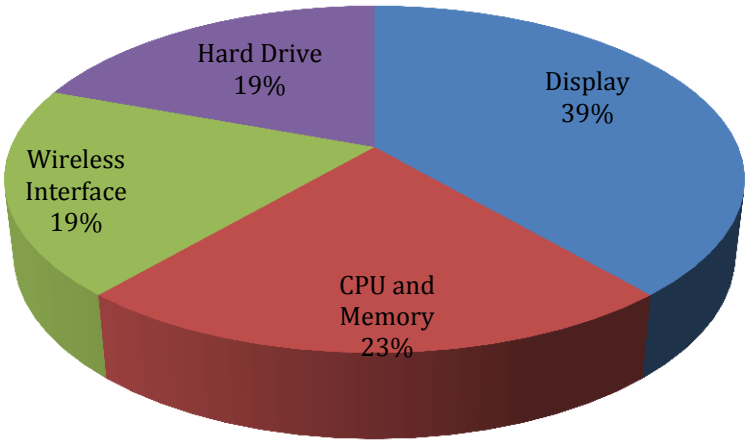


Figure 11: Toshiba 410 CDT mobile computer. Typical power consumption of a Toshiba 410 CDT Mobile Computer [36].

In mobile devices such as smart phones the power break down is slightly different. Figure 12 displays a pie chart of the power consumption break down for a typical mobile device Wi-Fi and Bluetooth enabled. In mobile devices the power consumption of Wi-Fi overshadows that of the other components. In particular the CPU of a mobile device uses much less power. Consequently by enabling Wi-Fi to sleep more often by implementing a default ZigBee communications network that handles the less bandwidth intensive processes it is possible to significantly increase battery life and conserve energy.

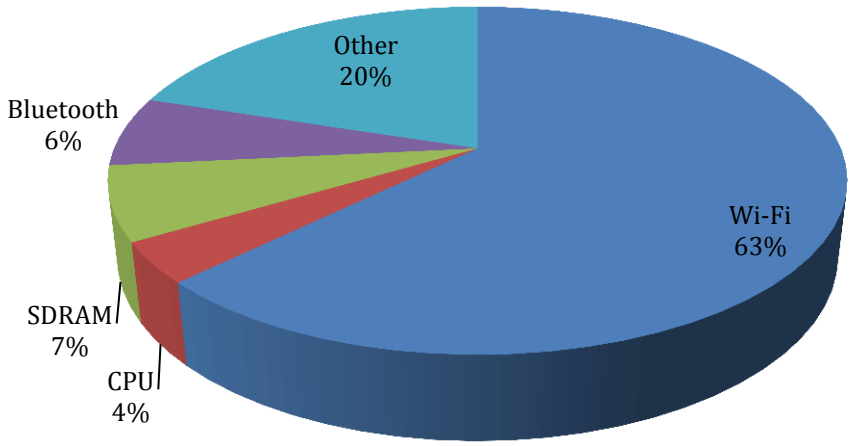


Figure 12: Laptop power breakdown. Power breakdown for a connected mobile device in idle mode. The wireless interfaces consume approximately 70% of the total power. Since the device is idle, the LCD and backlight are turned off – consuming zero power. Other includes power regulation and other smaller subsystems (such as LEDs) [10].

To combat the rate of energy consumption wireless technology experts have developed techniques to reduce this consumption. Wireless radios typically have three states of operation; transmit, receive, and idle. Standby, idle state, and sleep mode are known as low power modes. During all these states the radio is in an extremely low power mode, almost off, but turns on after a predetermined interval of time to receive a beacon from the access point (AP). The beacon will tell the radio whether it has incoming data. If it does have data then it will switch to receive mode. Once it receives the data the radio switches back to sleep mode. Likewise when the radio needs to transmit data it switches to transmit mode, once the data is sent the radio returns to idle mode. Often receive and transmit modes are lumped together in a mode called active transfer mode.

Table 4: Radio states of operation.

State of Operation	Description	Energy Consumption Rate
Transmit	Sends data out of the radio.	Highest consuming power state.
Receive	Receives incoming transmissions.	High power consumption rate, but slightly lower than transmit.
Idle	Transmits and receives no data but turns on receive state periodically to receive beacons.	Lowest power consumption.

As mentioned earlier in sleep mode the radio is on standby, only receiving and transmitting in intervals. Many times the radio will turn on during sleep mode only to find that it has no data addressed to it. This process uses energy which is undesirable for longer battery life. To solve this issue this project proposes to use ZigBee during the sleep mode and also when the network's bandwidth is experiencing minimal capacity. The Proxim RangeLAN2 2.4 GHz 1.6 Mbps PCMCIA (Wi-Fi) card consumes 1500 mW in transmit, 750 mW in receive, and 10 mW in sleep mode and power consumption for Lucent's 15 dBm 2.4 GHz 2 Mbps Wavelan PCMCIA card is 1820 mW in transmit mode, 1800 mW in receive mode, and 180 mW in standby mode [36]. Current draw from ZigBee radios tells us that the average transmit current draw for a ZigBee radio is 15.8 mA, if Wi-Fi has a current draw of about 300 mA according to Table 5 that's 5% of the transmit power of Wi-Fi. ZigBee's transmit power is even lower than the sleep mode power consumption values of Wi-Fi. This project proposes to take advantage of this aspect of ZigBee [38].

Table 5: Current draw from ZigBee radios

ZigBee Radio Model	Transmit (mA)	Receive (mA)
mica2	15	9
mica2dot	15	9
micaz	17.4	18.8
AVERAGE	15.8	12.26666667

Table 6: Typical Current Draw Values of Wi-Fi [36], [10], [6], [39], [35]

Wi-Fi Card	Low-Power Idle (mA)	Transmit (mA)	Receive (mA)
CX53111	N/A	219	215
Prism I	50	488	287
Prism II	43	325	215
ORiNOCO PC Gold	161	280	190
Cisco AIR-PCM350	216	375	260
Linksys WUSB54GC[1]	267	369	304
Experimental Values obtained from this project.[1]			

Figure 13 shows the power consumption of each protocol during transmit and receiving. From Figure 13 we can see that ZigBee is the better protocol in terms of power consumption. Most notable is that Wi-Fi consumes roughly seven times the power of ZigBee during both modes of operation. However, Wi-Fi has certain advantages that are important to consider. One advantage of Wi-Fi is the energy consumption, with units of Joules/Megabyte. This measurement illustrates how many Joules of energy are used to transfer a single Megabyte of data. If the user wishes to download an extremely large file, or perhaps stream video it would take a long time to do so with ZigBee, meaning that ZigBee would need to stay in active transfer mode for a great amount of time eventually consuming significant amounts of energy. While if the user used Wi-Fi for the same application it would download much quicker and therefore the radio would be active for a much shorter period of time and in turn use less energy. The ideal solution for this problem would be to develop an algorithm that would be able to sense when an application needed the high capacity and speed of Wi-Fi and switch between that and ZigBee when appropriate.

Table 7: Typical Power Consumption of Wi-Fi Radios

Wi-Fi Card	Idle (mW)	Transmit (mW)	Receive (mW)
Cisco PCM-350	390	1600	N/A
Netgear MA701	264	990	N/A
Linksys WCF12	256	890	N/A
CX53111[1]	N/A	723	710
Linksys WUSB54GC[2]	1308	1775	1470
802.11b Wavelan	1319	1675	1425
Experimental Values obtained from this project.[1]			

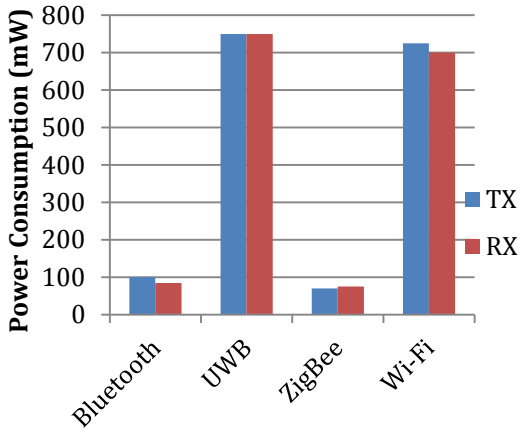


Figure 13: Comparison of the power consumed by the transmit (TX) and receive (RX) states for Bluetooth, Ultra-wideband, ZigBee, and Wi-Fi protocols [38].

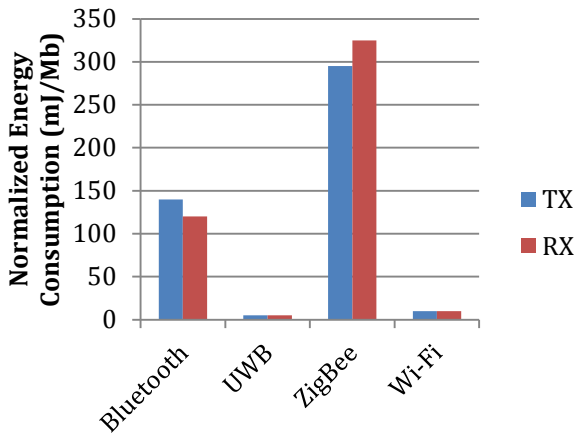


Figure 14: Comparison of the normalized energy consumption for each protocol [38].

2.4 Green Communications

There are two areas of current research into reducing wireless communications power consumption. One area of research is utilizing multiple wireless protocols to achieve the lowest possible power consumption while still maintaining an acceptable bandwidth for the current tasks. The other area of research is intelligently powering off the wireless radios of the device based on user demand and history. Programs monitor user behavior to determine at what times the radio is not likely to be used, and therefore power it down. Several of these approaches are outlined and analyzed below.

CoolSpots, a research project by researchers from Intel and UC San Diego, aimed to reduce power consumption by intelligently switching between Wi-Fi and Bluetooth wireless protocols. When available, Bluetooth became the default transmission mode. A program on the user's computer monitors the network usage. Intelligent algorithms with weighted parameters were put into place to decide when to switch from Bluetooth to Wi-Fi, and vice-versa. The results of the project were very encouraging – devices running CoolSpots had over a 50% reduction in energy consumption of the wireless subsystem [10]. This research was valuable because it utilized actual measured results from physical testing. This project proved it was physically possible to reduce power by using a multi-protocol system. Unfortunately the protocols used greatly hindered the mobility of the mobile user. Since Bluetooth itself is a "cable replacement" protocol, it has a very limited range when compared to Wi-Fi.

A similar project, "A Switch Agent for Wireless Sensor Nodes with Dual Interfaces: Implementation and Evaluation", explored the advantages of a dual-protocol network. This study only utilized simulations instead of a full physical implementation. There results indicated that, the end-to-end

delay and throughput achieved by the proposed interface switch agent was comparable to those achieved in a network of sensor nodes equipped only with IEEE 802.11 radios. Secondly, that energy consumed in the network using their interface switch agent was a fraction of that consumed in a network of the IEEE 802.11 sensor nodes and is comparable to that of sensors using only IEEE 802.15.4 radios [11]. This project is important because it proves that by using a multi-protocol system, significant gains can be achieved without sacrificing performance. This single handedly proves the feasibility of the team's concept for a multi-protocol system. The only downside to this project was the implementation. It was purely simulation based, relying on mathematical models rather than actual measurements. This is the major division between the team's approach, and the approach outlined in this paper.

Another area of research in wireless communications power reduction deals primarily with the application layer. In particular a paper called *Managing Battery Lifetime with Energy-Aware Adaptation* proposes a program called PowerScope, that monitors the energy consumption of a computer and pin points what process is the top cause of power consumption. Using this program they were able to modify Linux to yield battery lifetimes of user specified duration. They found that they were able to extend the life of the battery by as much as 30% [40].

Another program, called JuiceDefender, runs on Android smartphones. It increases battery life by turning off many features of the device, such as the Wi-Fi radio, 3G radio, display, and GPS depending on user customizable parameters. Parameters include time of day, time the device has been inactive, location, and battery life remaining. The Wi-Fi or 3G radios can be scheduled to turn on routinely to check for updates. Users can select from different pre-configured profiles, or create their own. Depending on the device used and how aggressive the settings are, battery life can increase anywhere from 25% to sometimes as much as 400% [41]. The most significant knowledge gained from this research was the software management system utilized. It outlined an approach to maximize battery life puely through software by eliminating unimportant processes and functions. However, this report was lacking guidance for power management of communication devices, primarily the Wi-Fi interface, and how to reduce its power consumption.

2.5 Summary

This chapter provides a background for the topics relating to the project. Section 2.1 Cognitive Radio discusses cognitive radios for their abilities to analyze and react to the wireless environment surrounding them. Both the software and hardware means of monitoring transmission parameters of radios were researched in this section. Section 2.2 Commercial Wireless Standards covers the multiple commercial wireless standards were considered for use in this project. In order to determine which protocols best suited the goal of the project team research into Wi-Fi, Bluetooth, and ZigBee was

necessary. Understanding energy management is also essential to develop a "green" network. As such, a thorough understanding of the power consumption of modern mobile devices was needed, as seen in section 2.3 Mobile Device Power Management. Finally, in section 2.4 Green Communications were studied to develop an understanding of the current work being put into reducing energy in the communications industry.

Chapter 3: Proposed Design and Project Logistics

This section discusses the main goal and objectives of the project. It goes into detail about the general design for the proposed wireless network management system and the hardware and software used to implement it.

3.1 Main Goal

Wi-Fi radios consume a significant amount of energy listening to beacons sent from access points as well as remaining in an inactive standby mode. Currently, almost all wireless protocols include a low power state in their standards, Wi-Fi has its own low power mode called Power Saving Mode (PSM). In this mode the wireless network interface card goes into longer sleep cycles. Every few milliseconds the card needs to wake up and look for beacons being sent by the access point. The beacons contain information that tells the computer whether or not it has any incoming data. When the wireless card isn't checking beacons it's in a sleep state. In the sleep state the radio uses as little energy as possible while still staying powered on. PSM also reduces transmission energies by going into a sleep phase for an extremely short period of time during transmissions. The time it is asleep during the transmission is extremely short, on the scale of milliseconds, however it does result in noticeable power savings. In low power mode the time between checking for beacons (the sleep state) is extended and in effect reduces power consumption. The problem with Wi-Fi is that even in sleep state it consumes a relatively large amount of power when compared to other protocols such as Bluetooth or ZigBee. Even when ZigBee and Bluetooth are in active transfer mode they consume less power than Wi-Fi in its sleep state. However, this is at the tradeoff of lower bandwidth and transmission range experienced by Bluetooth and ZigBee.

The goal of this project was to develop a wireless network that combines the energy efficiency of the ZigBee protocol IEEE 802.15.4 and the speed and high bandwidth of the Wi-Fi protocol IEEE 802.11 in order to improve upon the energy efficiency of current Wi-Fi only networks. The ZigBee radios would be used for low bandwidth applications, for which Wi-Fi would be in a power save or low traffic state. Then when additional bandwidth is needed the Wi-Fi connection would be initiated. Such a network would be ideal for any wireless device that relies on battery power. The benefits of this network would be extended battery life as well as a decreased impact on the environment through the conservation of electrical energy.

This design has many technical challenges to overcome. These challenges range from implementing basic communication features of the two protocols, to the overall data and power management system. These technical challenges include:

- **Standardization of data transfer methods into each protocol:** Currently, there exists no program that can interface between the ZigBee and Wi-Fi networks. So the project team needed to construct and program this interface themselves.
- **Implementing power control of Wi-Fi hardware:** In standard OS's the Wi-Fi hardware is controlled by the network manager that is imbedded in the OS. For this project the team needed to develop a technique to turn off the OS's network manager and allow a different network manager, developed by the team, to take control.
- **Implementing protocol/radio switching control:** A program is needed to interface between the two radios and also to control the switching aspect of them.
- **Implementing ZigBee, relatively unused protocol with little software and documentation available for developing a data transfer network:** ZigBee is a relatively new technology, and at the time that this project was being developed no prebuilt software was available to control any sort of data transfer between two ZigBee nodes. So the team programmed their own software to allow data transfer.
- **Testing, monitoring, and measuring the power efficiency of the proposed network:** In order to evaluate and monitor the wireless network and devices the team needed to develop a method to measure and observe the electrical consumption of the devices.

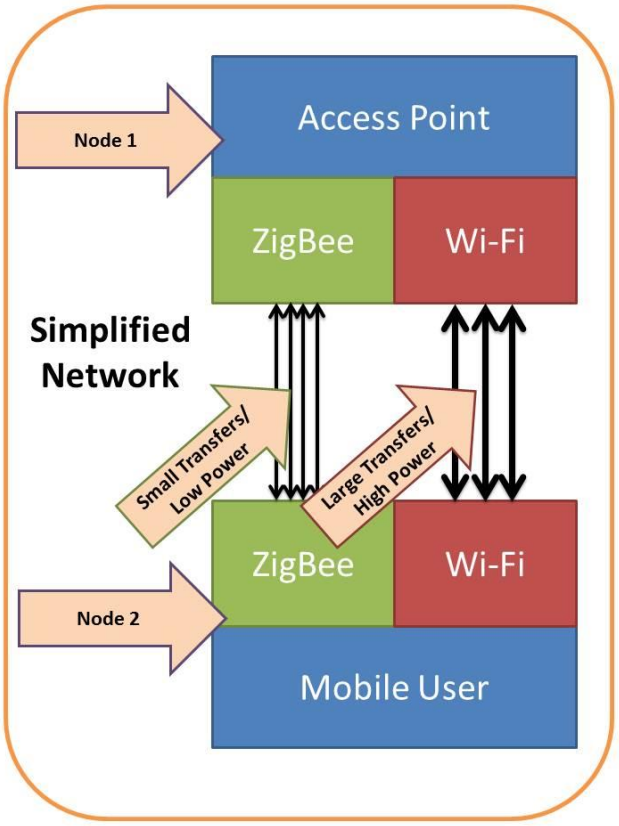


Figure 15: Simplified Dual Node Network, utilizing a single access point and mobile user.

The diagram above outlines the basic physical system the team is to develop. The concept will utilize only two nodes, minimizing the complexity of the network. Since this project's purpose is to demonstrate the power advantages of a multiprotocol network a larger network will not be constructed. This complexity can be expanded in future designs if necessary. As seen above the nodes to be constructed are an access point and mobiles user. As seen each node is equip will both a ZigBee and Wi-Fi radios, but will use these radio differently upon role in the network. For example, since the access point, in theory, will be constantly connected to multiple users with both protocols, it will maintain both interface at all times unlike the user nodes.

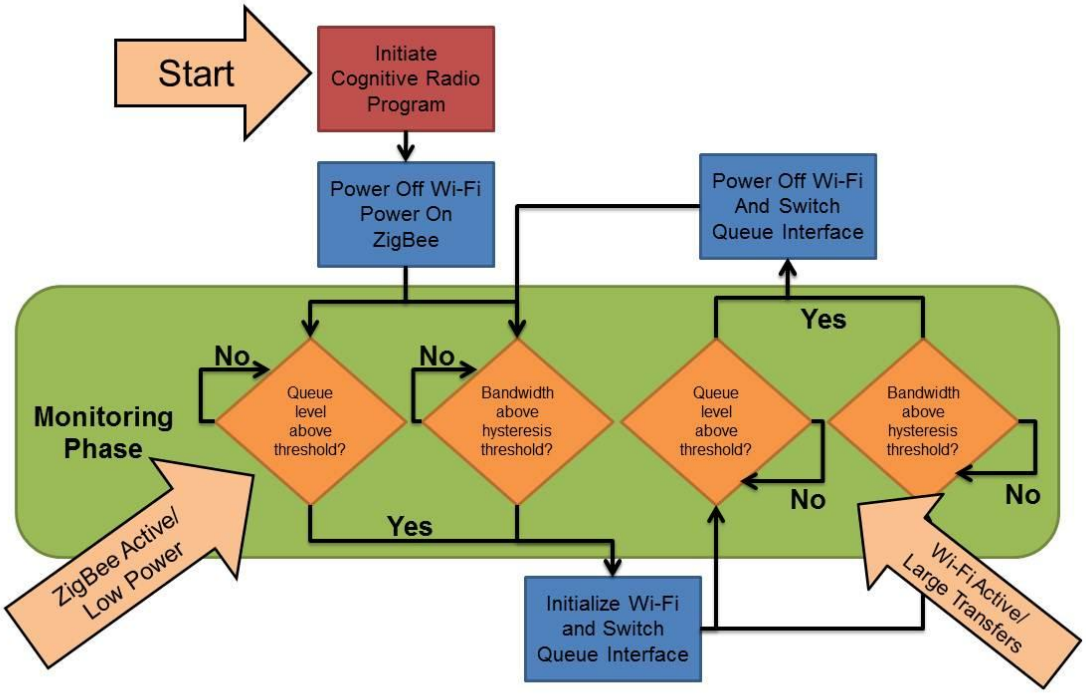


Figure 16: Flow-Chart of cognitive radio logic to gain the best power performance.

Above is a flowchart of the internal intelligence of the mobile user. This flow chart illustrates the heart of the mobile user's cognitive process. It controls which interface will be used to transfer data and when switching will occur between these interfaces. The cognitive process implemented utilizes both the amount of data in the queue and the current bandwidthbeing used by the two wireless radios of the mobile user. The system decides when to switch between the two protocols based on precalculated thresholds. These thresholds were constructed using the transmission power usage and the bandwidth capabilities of each protocol. When the threshold of ZigBee is exceeded the system switches to Wi-Fi, and when the lower bound threshold of Wi-Fi is passed, meaning that Wi-Fi is not using enough of its bandwidth to warrant the high power consumption, the system switches Wi-Fi off and ZigBee takes control. The overall concept of this system is that Wi-Fi is used for data transfer because of its speed and ZigBee is used as the idle state of operation due to its extremely low power consumption.

3.2 Project Objectives

The main objective of the proposed design is to reduce the energy consumption of the communications system through the use of multiple wireless protocols controlled by cognitive methods. To achieve this objective the system must meet several objectives:

- The system needs to have an algorithm to determine which protocol (ZigBee or Wi-Fi) is the most efficient for the task or situation.
- The system needs to seamlessly transition between the two protocols.
- The system needs to be able to transmit and receive data between two computers using the two protocols.
- The system, when implemented, should not have a significant negative impact on the data rate and performance of the system when compared to a Wi-Fi only network.
- The system is required to be more energy efficient than the Wi-Fi only network.

These objectives will be completed through the development of individual subsystems and then be combined to create a single fluid system. The subsystems consist of the ZigBee Communication Subsystem, the Wi-Fi and ZigBee Algorithm Subsystem, and the Power Management Subsystem. These subsystems are explained in section 3.4 Design Decisions. When completed this system will enhance the battery efficiency of the mobile system for which it is installed upon, while at the same time not hindering the performance of the system compared with current Wi-Fi only solutions.

3.3 Project Management and Tasks

To complete the project on time the team divided the work into three sections and split up the sections among the members. The project was broken up into three section for implementation; section 4.1 Setup and Installation of Equipment, 4.2 Software Radio Controller, and 4.3 Power Measurement System. The project was broken up in this manner so that each member could specialize in one of the three main areas of study of the project.

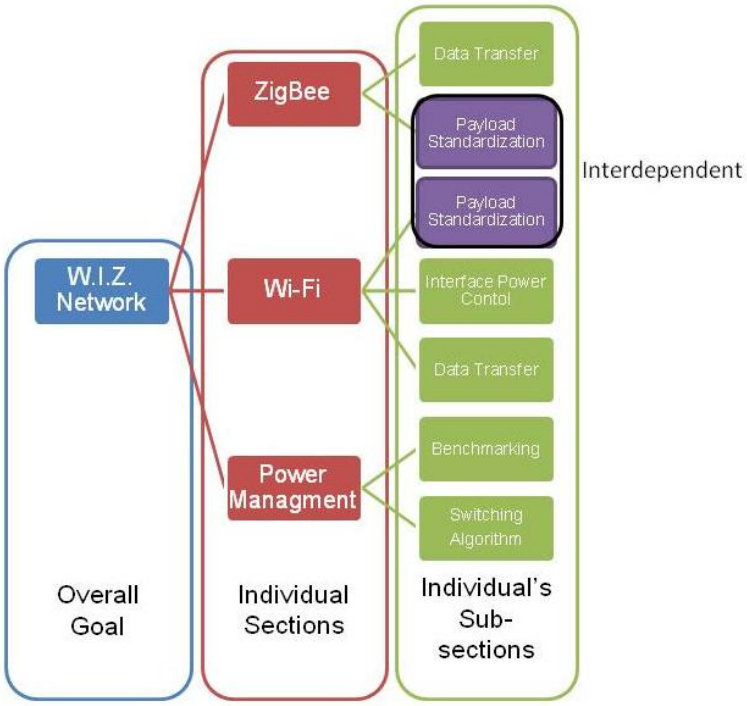


Figure 17: Breakdown of project among the team.

The team required a ZigBee communication system that was able to transmit and receive files, as such it was necessary for one member to acquire extensive knowledge strictly regarding ZigBee. The algorithm for switching dealt with interfacing between the two protocols in one program; this required one member to acquire extensive knowledge into the API's of both Wi-Fi and ZigBee. The final member was responsible for understanding the power modes and how to monitor these modes of the Wi-Fi, and ZigBee, card in an intelligent way to determine the efficiency of the overall design. Wi-Fi networks are already developed and used all over the world, and as such there was no need to focus any member on setting up a Wi-Fi network in the same manner as ZigBee. As seen from the diagram above these three main tasks are broken down into several subtasks. The lines in Figure 17 indicate tasks that are dependent on the completion of other tasks. These interdependences are the first steps in the overall system unification, which forced the team early on to work on components together. The natural progression, as seen above, was the construction of these smaller blocks in green and purple, then a system level conjunction of the red blocks, and finally an overall system construction in the blue block. These system level combinations proved to be the most difficult and the most time intensive to troubleshoot.

The project team also created a Gantt chart. The Gantt chart displays each task that needed to be completed and the time allotted to complete the said task. The planned Gantt chart for this project is displayed below in Figure 18.

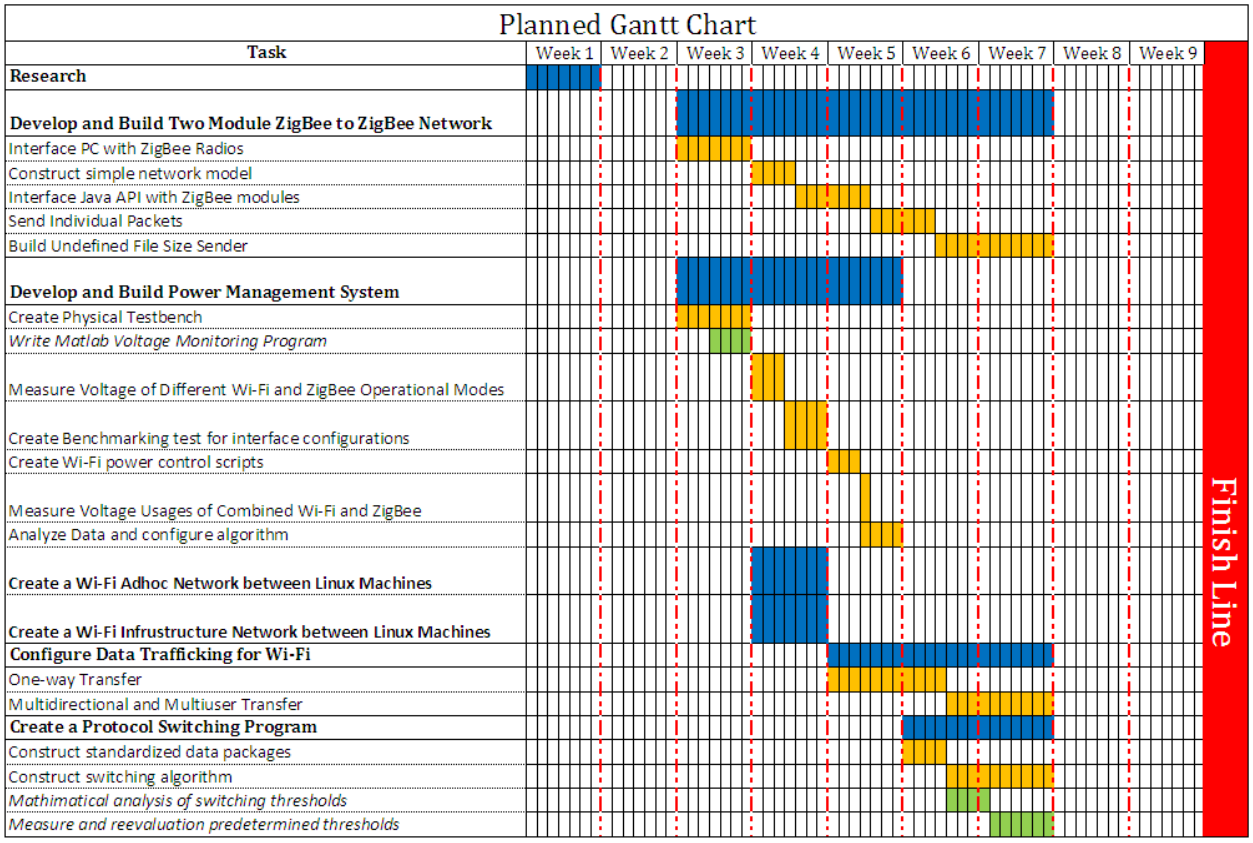


Figure 18: Planned Gantt Chart.

However, in reality things don't always go as they plan and as a result Figure 19 shows the Gantt chart for the actual progress through each task.

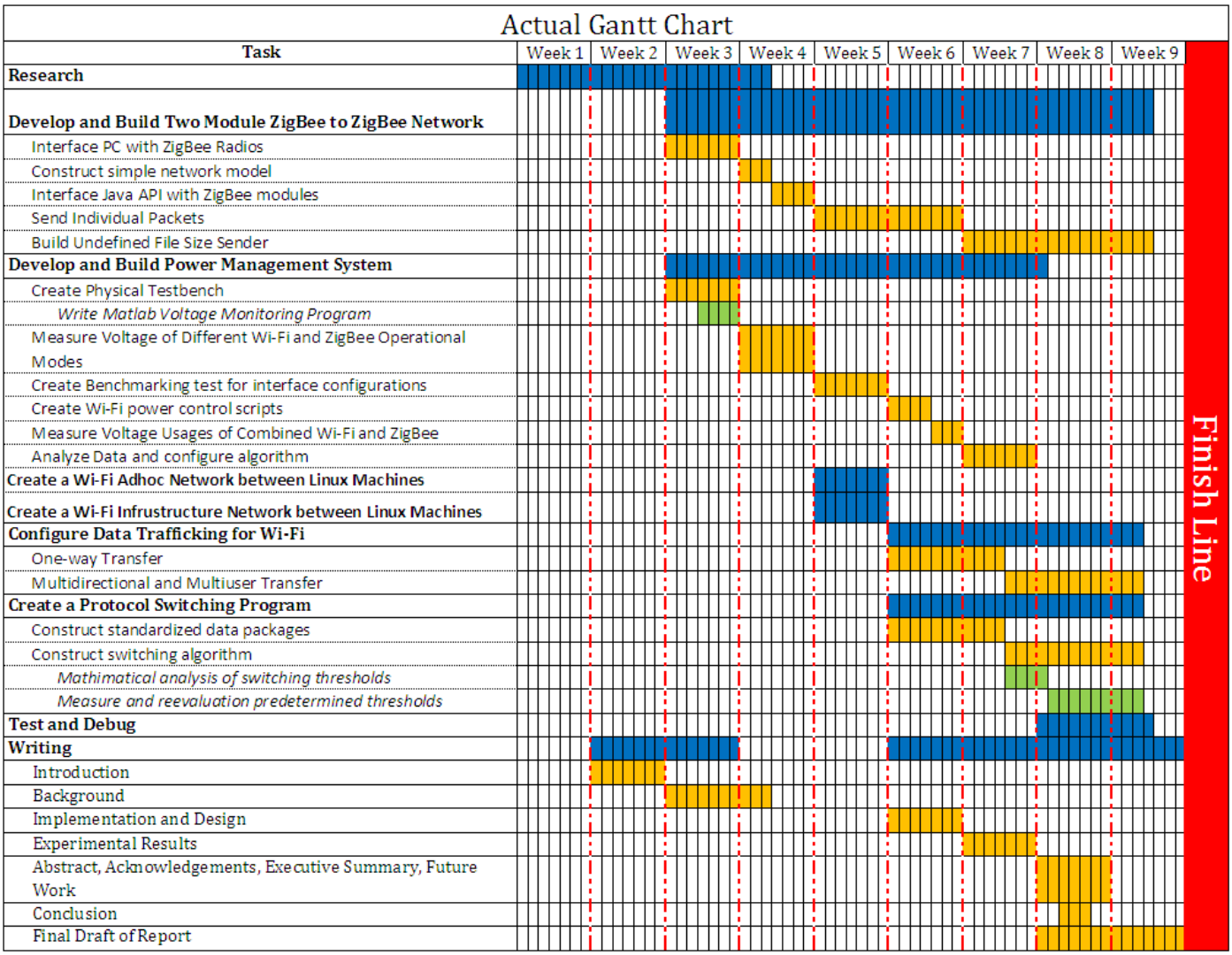


Figure 19: Actual Gantt Chart.

3.4 Design Decisions

Design Decision Methodology

The overall goal for this project was to create a wireless network that used multiple protocols to reduce energy consumption in mobile devices. The plan would require that both the access point node and the mobile device would have both protocols. If the user of the mobile device needed to run an application that required a large bandwidth, such as video streaming, the device would switch to Wi-Fi. When the user didn't required high bandwidth, such as when sending texts or being idle, the mobile device would switch to ZigBee to conserve power. The two major benefits of this idea are a reduced carbon footprint on the environment and longer battery life, while not hindering performance.

To create an energy efficient system the network needed two protocols that had complementary attributes. To determine which protocols to use the team compared the aspects of four protocols; Wi-Fi, ZigBee, Bluetooth, and 3G. 3G was immediately thrown out due to its complexity, scope, and lack of readily available development hardware and software. This left Wi-Fi as the only high bandwidth means of data transfer. This left Wi-Fi as the only high data rate and high bandwidth means of data transfer. Wi-Fi is also highly ubiquitous and many software methods exist for its protocol employment, making it one of the easier protocols to implement. So the team chose Wi-Fi as the first protocol. The team next moved towards selecting a second protocol which would complement Wi-Fi. To determine what protocol would complement Wi-Fi the team constructed Table 8, to evaluate its strengths and weaknesses.

Table 8: Wi-Fi pros and cons table. To find a suitable protocol to complement Wi-Fi it was necessary to evaluate the strengths and weaknesses of Wi-Fi first. Overall Wi-Fi is a quality wireless network. To improve Wi-Fi the team decided that the weakness of Wi-Fi was its high power consumption. Leading them to search for a low power protocol. Data taken from [38].

Wi-Fi (IEEE 802.11 a/b/g): Pros and Cons	
Pros	**Cons**
Max Signal Rate of 54 Mb/s	Nominal TX Power of 32-100mW
Range of 100 m inside, 300 m outside	
Channel Bandwidth 22 MHz	

The table claims that the main con of Wi-Fi is its costly power consumption. To compensate for this the team looked at two lower powered protocols; Bluetooth and ZigBee. In Table 9 several aspects of these protocols are compared. The bandwidth and signal rate of Bluetooth is much higher than that of ZigBee; however the limited range of Bluetooth compared to Wi-Fi would force the group to construct several Bluetooth nodes to compensate for the it's limited range. ZigBee on the other hand has a nominal range that is very close to that of Wi-Fi. With ZigBee the group could simply install ZigBee radios into Wi-Fi nodes, creating a one to one Wi-Fi to ZigBee node ratio. By constructing fewer nodes the group would decrease cost and complexity of the overall network. Therefore, the project team chose ZigBee for its low power consumption and superior range capabilities. The only downside being the limited documentation that exists for ZigBee, compared with that of Bluetooth.

Table 9: Bluetooth vs. ZigBee vs. Wi-Fi Comparison Table. Data acquired from [38].

	Wi-Fi (IEEE 802.11 a/b/g)	Bluetooth (IEEE 802.15.1)	ZigBee (IEEE 802.15.4)
Max Signal Rate	54 Mb/s	1 Mb/s	250 Kb/s
Nominal Range	100 m (inside), 300 m (outside)	10 m	10-100m
Channel Bandwidth	22 MHz	1 MHz	0.3/0.6 MHz; 2 MHz
Nominal TX Power	32 - 100mW	1-10 mW	1-0.003mW

With the necessary protocols chosen the team moved forward to create an actual wireless network that used these two protocols, Wi-Fi and ZigBee. For simplicity the network would be simplified to a two node system, which included an access point and one satellite mobile node. The access point and mobile device would be constructed with both protocols. On the access point both wireless radios would be constantly running, while the user would be running initially on ZigBee by default. At first ZigBee would only listen for packets and act as the standby mode for the mobile user. If the user wanted to stream video or any other bandwidth intensive task the mobile device would switch to Wi-Fi. After getting this working correctly the team would then begin to utilize ZigBee for minor transfer tasks such as text messaging. To switch between the two protocols the team would need to implement some sort of cognitive switching alogrithm. Before deciding how to implement this switching algorithm the team moved towards selecting a platform for the two chosen protocols.

The first platform the team considered was a cellular device, specifically the Android cell phone. Androids were readily available and have detailed documentation in there development. Unfortunately the team decided that developing this technology on a prebuilt cellular device would prove too difficult due to the restraints places on such devices. It would almost be impossible to add secondary hardware radios to the device. So the team decided to develop the project on laptop PCs for a more flexible development platform. This development would primarily take place in Linux due to its level of hardware control.

With the hardware selected the team reevaulated the goal of the project and considered what was possible with this hardware in a nine week period of time. After much deliberation, the group chose to simplify the project further. Instead of switching between protocols depending on the application's bandwidth requirements, the group instead would use ZigBee for low power mode and for sending Wi-Fi authentication and handshake information. Energy would be conserved primarily by eliminating nonessential Wi-Fi transmissions including handshakes and other link layer information. The team suspected that sending handshake information would be the most difficult step in the process due to the

complexity of the Wi-Fi standard. However after much research and hardware evaluations, sending authentication data for Wi-Fi through ZigBee proved to be both very difficult and very ineffective for conserving energy. The idea was dropped for two reasons:

1) The layers that makeup the Wi-Fi protocol are incredibly complex, and primarily the lower layers which are responsible for network association are inaccessible with the hardware available to the project team. Just to get access to those Wi-Fi layers the team would need to reverse engineer the networking cards. This would take much more time and expertise than the group could spare.
2) ZigBee and Wi-Fi use different levels of abstraction from the operating system and in order to make them work together we needed an even plane, and working at the MAC and PHY layers doesn't allow this freedom.

Another problem observed through experimentation was that the energy savings saved from using ZigBee for the authentication processes would be very small. This is due to the narrow periods for which authentications takes place in Wi-Fi networks. To produce a greater power savings and simplify the project the team decided on a new project concept. The new concept still involved a wireless network that used two protocols, ZigBee and Wi-Fi, and the utilization of the same hardware. The access point will be a laptop with both Wi-Fi and ZigBee radios. The access point will emit both protocols continuously. The mobile user, a laptop, will also be able to receive and transmit both protocols. The network will only be used to send data files between the two nodes. The team chose to only send files to keep the project simple, the project still intends to prove that using ZigBee for low bandwidth and data rate intensive tasks is more energy efficient than Wi-Fi Power Saving Mode (PSM). The files will be different sizes and will be sent at different data rates. In standby mode the laptop will use ZigBee to receive beacons from the ZigBee access point. In this standby mode the Wi-Fi will be off, only turning on when ZigBee could not handle a high bandwidth demand.

When the user sends a file the file will be sent through ZigBee automatically. However, there will be an algorithm that monitors the bandwidth and data rate of the radio channels. This algorithm will be responsible for switching between Wi-Fi and ZigBee. While the user sends a file the algorithm is making sure the bandwidth is not overcrowded. To determine when it is appropriate to power on Wi-Fi and use it for data transfer, the team would develop an intelligent algorithm to make this decision. The algorithm would monitor bandwidth and data rate of the combined interface. This algorithm would have precalculated thresholds based on the power characteristics of the physical device, and the network throughputs of the devices. Once the interface was selected a software interface was contructed to communicate between the different radios. Instead of physically monitoring data transmitted the team devised a queuing system. The queue would be a first in first out list and depending on its size and rate of

growth would determine which protocol its output would be transmitted from. For example, if the queue reached a certain level or was growing as a relatively fast rate the algorithm would initialize Wi-Fi and switch the queue's output to use Wi-Fi. The system would also have to be smart enough to not switch to an interface that was unavailable.

To prove that the network created in this project is more power efficient a number of tests will be done and compared to the control. The control will be Wi-Fi (PSM) running the same processes but without ZigBee. The project will monitor the voltage drop across a resistor that will be placed in series with the USB power line attached to the Wi-Fi USB adapter and another to the ZigBee USB adapter, displayed in Figure 24. With the voltage it is possible to calculate the current, and with the current it is possible to calculate the power, measured in Watts. This will measure the communications power but not the total power of the system. To measure the total power the team will use PowerTOP. PowerTOP is a Linux program that monitors the total system power consumption, as well as individual component power usage. The project team will also measure the Joules/Mbit. Watts allow the group to examine the rate at which energy is consumed and how much is consumed overall during the specific process. However, Joules/Mbit will allow the group to observe how much energy the computer uses per bit of data with the proposed multi-protocol network. Both of these values are important for evaluating the proposed network.

3.5 Design Summary

The design approach implemented has gone through many changes and revisions. These revisions were necessary to create a multi-protocol wireless network that was both energy efficent and without compromised performance. The final design consists of two netbooks running Linux and software to control their specialized dual communication interfaces, this design is both stable and easily configurable.

Chapter 4: Implemetation

This section discusses the specifics of the execution of the project. First, it describes the setup of the equipment, including laptops, drivers and operating systems. Then, it details the algorithm and code development process of the software controlled radio. Finally, it explains the setup and utilization of the power measurement equipment.

4.1 Setup and Installation of Equipment

The first step was to decide which hardware platform to use as a testbed for the software controlled radio. While the future goal of this project would be to implement the SCR on all mobile devices, a single platform was deemed sufficient for this proof-of-concept implementation. Laptop computers were used because of their portability over desktop computers. Asus EEE netbook PCs were chosen because they were readily available to the team in the lab. The next step was to decide which operating system to install on the laptops. The Ubuntu Linux operating system was selected over Microsoft Windows and OSX because they offer more hardware and software control to the user.

Afterwards, the team had to do decide on radios to use. The first question was whether to use internal or external radios. External radios were chosen because they allowed the communications power to be measured directly and independently of the total system power. To measure the power consumption and efficiency of internal network cards, the team would have to physically open the laptop and connect digital multimeter to it. However, external USB network adapters allow much easier access to the power pins, making power measurements much easier. The ZigBee radio used was the XBee Pro USB radio. The Wi-Fi adapter used in this experiment was the Linksys Cisco WUSB54GC 802.11 b/g. These unit was chosen because of their availability. The team was given these Wi-Fi and ZigBee adapters upon the start of the project.

Now that the radios were chosen, the next step was to determine how to control them to optimize power. Since Wi-Fi was going to be turned on an off as needed, the team needed a way to reduce its power usage when it was not being used. This control was implemented through the Unix commands "iwconfig" and "ifconfig". Additional control was also implemented by softblocking the radio to eliminate its power usage on a closer hardware level. The Wi-Fi interface would first be softblocked and then hardblocked. This was accomplished by "rfkill" switch packages. Softblock is a method used to simply disable a radio network interface. "Softblocking an interface" is blocking the use of that interface through software. Hardblocking, however, physically powers down or disconnects a device. This is usually done with a button or by physically ejecting a device. Another way to hardblock a device is to power down its root power supply, which can be done through software, primarily on USB devices. Normally hardblocking can cause a device to disassociated from the host, meaning it must be

reassociated. This can cause considerable delay. Fortunately, since the team was able to accomplish hardblocking through software, they were able to overcome this obstacle.

The team was able to control the power modes of the root USB hub for which this device was connected to. With this new ability to control the Wi-Fi card came a number of concerns for the team. They included latency issues associated with hard-blocking because the card must re-associate with the operating system once it has been hard-blocked. The second issue was the uncertainty of the power saving gained or lost from this approach. The team was unsure of the actual power savings gained by physically turning off the USB root hub verses just soft-blocking or powering down the radio. To answer these questions the team conducted a series of tests.

The first test was to measure the latency of power up and re-association when powering down the USB Wi-Fi dongle fully. The tests used a constantly pinging sources to confirm when disconnection and when re-association occurred. The pings are sent in one second intervals from the node to and external server, and can be seen in the right half of the screen shot below. On the left is the output of the teams script which performed several operations. It basically would disconnect from the network, power down the Wi-Fi card, wait 1 second, power the card back up, then finally re-associate to the managed network. As can be seen from the screenshot it took roughly 5 seconds to complete the test. This test was repeated several times with times ranging from 3 seconds to 5 seconds to complete. The reason this ranged was because of the associated that needed to be done by the DHCP server within the router itself. Additional tests were done using an statically maintained internet procotol addresses instead of dynamic ones. As long as the system connected within its lease period, which commonly last 24 hours, the card could associate instantly. In future versions static internet protocol addresses could be managed through the ZigBee interface to predetermine tables before connections. This would guarantee instant connections with Wi-Fi. In this project the team decide to focus primilary on dynamic internet protocol address because it is generally more common in the market place. Overall, these results were extremely acceptable to the team, and the powering aspect appeared almost transparent from cycling the card without dropping from full power.

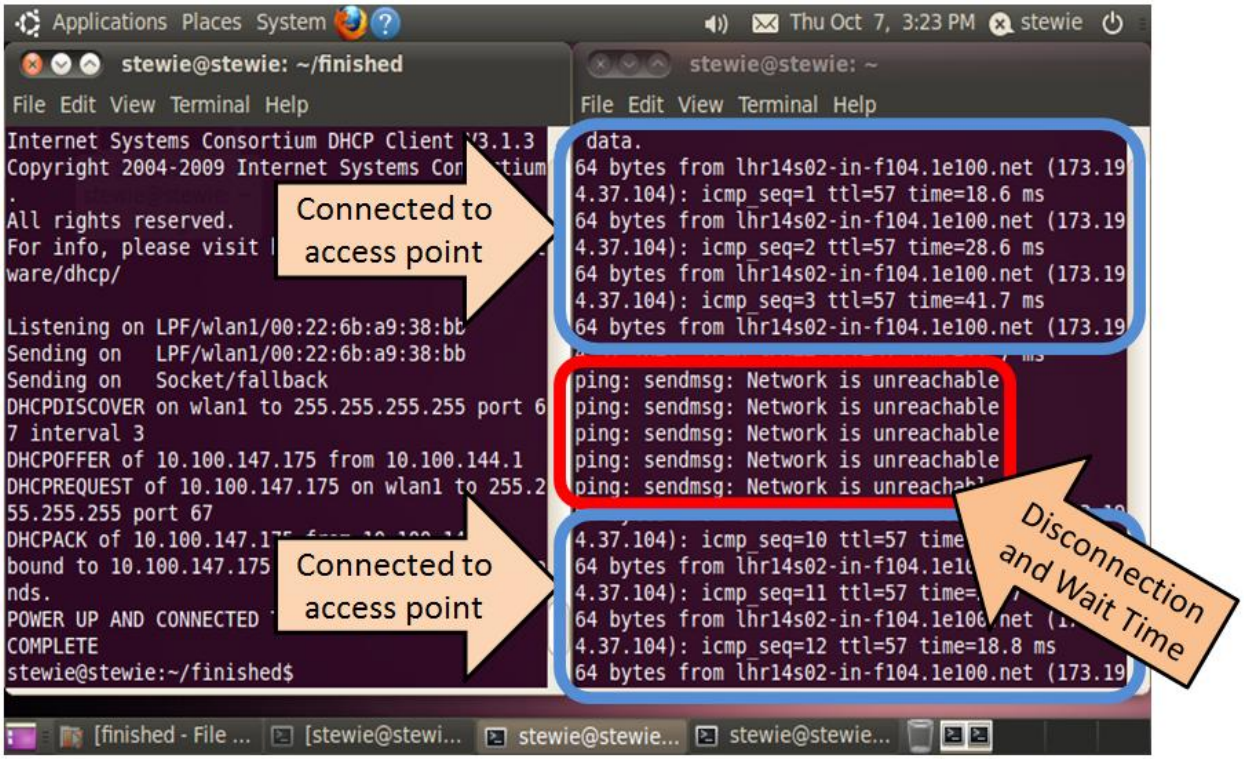


Figure 20: Wi-Fi USB connectivity scripts testing. The terminal window on the left shows the Wi-Fi power down and power up scripts running. The terminal on the right shows when packets can reach the internet and when they cannot.

The second test the team conduct utilized the same script as the previous test, but with an extended waiting window between powering up and powering down. The purpose of this test was to determine the difference between the card's power usage when being hard-blocked and being unhard-blocked. To perform this test the used a Hewlet Packard 34401A Multi-meter connected to Matlab for data acquisition. Matlab would take measurements every 0.5 seconds of the voltage leading into the Linksys WUSB54GC dongle through a predefined resistance of 500mΩ. Below is a picture of the teams test bench setup using the Eee PC 4G to control the networking card, running Ubuntu 10.04 LTS.

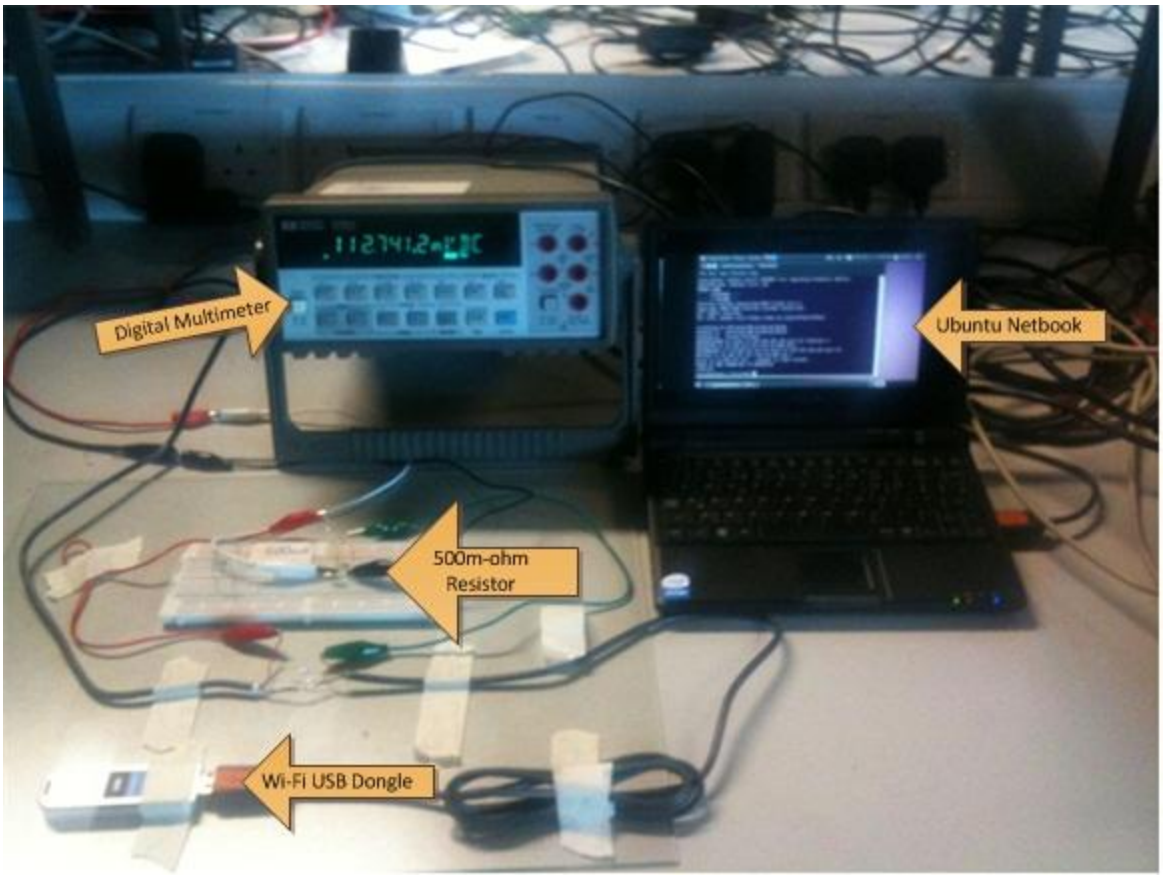


Figure 21: Test bench setup with Eee PC to monitor latency and power usage. The Netbook is running the communications system and controlling the radios. The USB cable is spliced open and connected to the resistor. The DMM, which is also connected to the resistor, measures and records the voltage drop across the resistor.

The team's data can be seen in the chart below, which is a representation of the voltage change over a period of twenty seconds. From the chart you can obviously see the power dip and sustained low voltage for a period of three seconds then a sharp rise. This very noticeable voltage drop was extremely attractive to the team. By powering off the networking card's root USB hub, they were able to drop the power consumption to roughly 2% of full power. Even in low power mode (PSM) the card will operate at 60% full power when idle. Therefore this drop in power to 2% provided the team a large power savings.

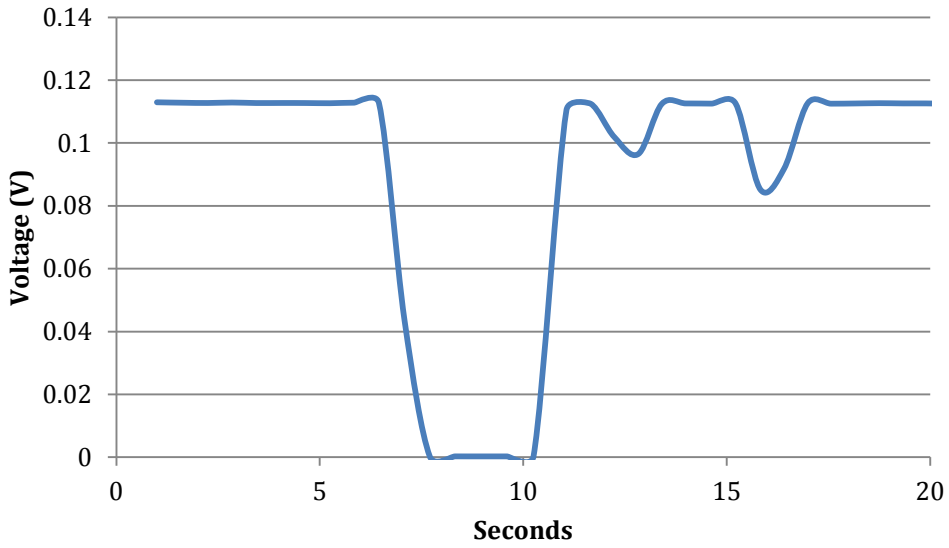


Figure 22: Voltage of Linksys WUSB54GC dongle being hard-blocked and unhard-blocked.

4.2 Software Radio Controller

The software controlled radio (SCR) handles all communications and data transfers. It monitors the amount of bandwidth used by the device and uses an algorithm to determine whether to use ZigBee or Wi-Fi to transmit data. The first step in developing the SCR was to develop this algorithm. This was done by considering the power consumption and data rate of both protocols. ZigBee is roughly 250Kbps and Wi-Fi is 54Mbps, but Wi-Fi can operate in a range of bandwidths. Therefore, as the bandwidth grows above a certain threshold it would be power efficient to switch from ZigBee to Wi-Fi. In other words, the team needed to determine at what bandwidth Wi-Fi becomes more efficient than ZigBee.

$$WiFi\ Data\ Rate = \frac{WiFi\ Instanteous\ Power \times ZigBee\ Data\ Rate}{ZigBee\ Instanteous\ Power}$$

Based on the above equation the bandwidth at which Wi-Fi becomes more efficient is 1.955Mbit/sec. With this calculated thresholds the team then moved the values to the device, were they were tuned even more. When moved to the actual hardware, significant transmission speed losses were seen. This occurred because of the increase overhead of the system, and level of controlled needed for the prototype. This was designed purely for proof of concept, with future work this speed could be regained. Since the bandwidth threshold where Wi-Fi becomes more efficient than ZigBee is relatively low, it was decided to simply consider the size of the queue when determining which protocol to use. All data to be transmitted and received was represented within a single queue. If the queue data size ever became larger than a certain size, then Wi-Fi would switch on. Once Wi-Fi is on and transmitting, it stays on until the queue is empty, since most of the inefficiency in this system is due to Wi-Fi initialization. It will remain on until the queue has been empty for a certain period of time, then turn off.

Now that the theoretical background behind the SCR was determined, the next step in developing the SCR was to select a programming language. Many languages were considered, including C++ and C#, but Java was ultimately decided upon for multiple reasons. The Java standard libraries include support for networking with TCP/IP sockets, creating graphical user interfaces, and multithreading – all features which would greatly aid in the programming of the SCR. Additionally, an open-source API for the XBee ZigBee radio was found written in Java. Lastly, Java is designed to be able to run on any platform that can run a Java Virtual Machine, including all major desktop operating systems (Windows, Macintosh, and Linux) as well as portable mobile operating systems such as Google Android.

Now that the programming language for the SCR were selected, the actual method of implementing and testing the system had to be decided upon. As mentioned before, the Wi-Fi protocol has support for networking through TCP/IP sockets whereas ZigBee does not. Since most networking applications on a personal computer are designed to use sockets, ZigBee could not be used with existing programs without modifying the source code of the program or operating system. Since this would take a long period of time to implement, it was decided that rather than monitor network traffic of typical user activities, such as viewing web pages, voice-over-IP calls, streaming videos, and video conferences, these activities could be simulated by creating a program to send and receive customizable amounts of data. In other words, a file transfer program would be created to send and receive file data through both Wi-Fi and ZigBee.

The first functionality built into the software controller were methods to interface with the XBee radio. Xbee-api, an open source Java application programming interface (API) was obtained to facilitate working with the XBee radio. It provides a wrapper to handle the low level functionality of common radio tasks, such as building and sending packets, waiting for acknowledgments, and listening for incoming packets. Xbee-api was used as a starting point to build the rest of the ZigBee radio software. A custom class was then created on top of the Xbee-api containing methods to perform the necessary tasks required of the radio.

Before sending and receiving file data from one laptop to another, certain information has to be sent between the two computers. Since, in this implementation, ZigBee would always remain on, it was decided to send all control information for the network through the ZigBee protocol. Since the control information of the network is sent over ZigBee, a common format for the data had to be created to ensure received packets were handled correctly. Five different types of ZigBee packets were identified. To differentiate them from each other, the first byte of each type of packet is given a unique number, or "status byte". The five different types of packets, along with their corresponding status byte, are listed below:

0 (Wi-Fi Request): Sent from a user to the access point when the user requests Wi-Fi bandwidth. It does not contain any data payload, since no other information is necessary in the request.

1 (Wi-Fi Info): Sent from the access point back to the user in response to a Wi-Fi request. It contains all the information necessary to connect to the Wi-Fi network (the ESSID, mode, and access point MAC address).

2 (Wi-Fi Stop): Sent from the user to the access point informing the access point that the Wi-Fi connection is closing.

3 (File Transfer Request): Sent either from a user to the access point or vice versa. It contains the file name and size in bytes of the file to be sent.

4 (File Data): Sent either from a user to the access point or vice versa. It contains bytes of the file being sent, with a maximum of 70 bytes per packet.

With the control information defined, the subsequeny step was to determine how to transfer data wirelessly. Java standard libraries include support for sockets to facilitate the transfer of data through any internet protocol (IP) based connection, such as Wi-Fi. To set up a connection between two computers, one computer first creates a socket on a specific port number and listens for connections. The client then opens that socket. Once the connection has been established, data transfer can take place. The computer sending the file reads the file into an input stream wrapper and sends that to the socket connection. The receiving computer receives this output stream and then sends that information into a file object to be saved on the hard drive.

Unlike Wi-Fi, ZigBee does not yet support the TCP/IP stack, meaning socket connections could not be used. However, ZigBee packets can be configured with up to seventy-two bytes. Therefore, rather than sending files through socket connections, it is still possible to send file data in ZigBee packets. Files were read seventy bytes at a time and stored in a ZigBee "File Data" packet. The packet is then sent to the receiver. The receiver has a packet listener, which runs in its own thread and waits until a ZigBee packet is received. When a packet is finally received, it strips out the status byte (a leading "4") and stores the rest of the data in a file object. A variable, `amountByteWritten`, is then incremented by the amount of bytes that were just written. If the amount of bytes written is equal to the total size of the file, then the file is saved and closed. If an acknowledgement is not received by the sender before the default timeout, the packet is resent to ensure data integrity. This continues until the entire file is sent.

Since the software controller could decides to switch between from ZigBee to Wi-Fi at any time, the ability to switch transmission protocols in the middle of a file transfer is imperative. The system was developed in a way that the transmission protocol is independent from the main program. In other words, the main program calls the `sendFile(file)` method, which internally handles sending the file through the currently selected transmission protocol.

Now that methods for sending and receiving files were completed, the next step was to develop the data queue that contains the stack of files to be transmitted and the algorithm that determines when to switch protocols. There are two objects that comprise the data queue. The first, `queueFiles`, contains information about all the files to be transferred. It stores the name of the file, the size in bytes, and whether the file is to be transmitted or received. The second, `queueBytes`, stores the sum of all the bytes left to be transferred for all the files in the queue. As files are being transferred, queueBytes is decremented in accordance with how many bytes have been sent. For every new file transfer request, a new `FileInfo` object is added to `queueFiles` and `queueBytes` is incremented by the size of the new file.

The bandwidth monitor contains all the logic that determines whether the mobile device should transmit using ZigBee or Wi-Fi. It runs in its own separate thread so it can constantly monitor the queue of files and the bandwidth consumption. Once started, the bandwidth monitor thread sleeps for 100 milliseconds, then runs its check of the data queue, executes its functions, and then loops back up to the beginning to sleep for another 100 milliseconds. The thread sleeps in order to not waste CPU cycles checking the data queue constantly. 100 milliseconds was chosen because it still checks the data queue often but does not overwhelm the CPU. The first check the monitor performs is to determine whether or not Wi-Fi is enabled. If it is not, it then checks to see if Wi-Fi is turning on and being initialized. If it is, then it loops to the top. If not, it checks to see if the data queue is greather than the Wi-Fi turn-on threshold. If the queue size is greater than the threshold, then there is too much data for ZigBee to send efficiently. A Wi-Fi request packet is sent to the access point and Wi-Fi will be turned on to empty the queue. If the queue size is not greater than the threshold, then the monitor loops back to the start.

If Wi-Fi is enabled, then the monitor checks whether or not the data queue is empty. If it is not empty, then there is still data for Wi-Fi to send. The queue empty time counter (*Wi-Fi Queue Empty Counter*) is reset to zero and the monitor loops back to start. If the data queue is empty, then the queue empty time counter is incremented. The thread then checks to see if the queue empty time counter is equal to its max threshold. If it is not, then the data queue has not been empty for too long a period of time and Wi-Fi should remain on in case there is network activity. If the counter is equal to its threshold, then the queue has been empty for a long period of time and Wi-Fi should be turned off to save power. In this case, a Wi-Fi stop packet is sent to the access point, Wi-Fi is turned off, and the queue empty time counter is reset to zero before the monitor loops back up to the start. A flowchart of the bandwidth monitor is shown below in Figure 23.

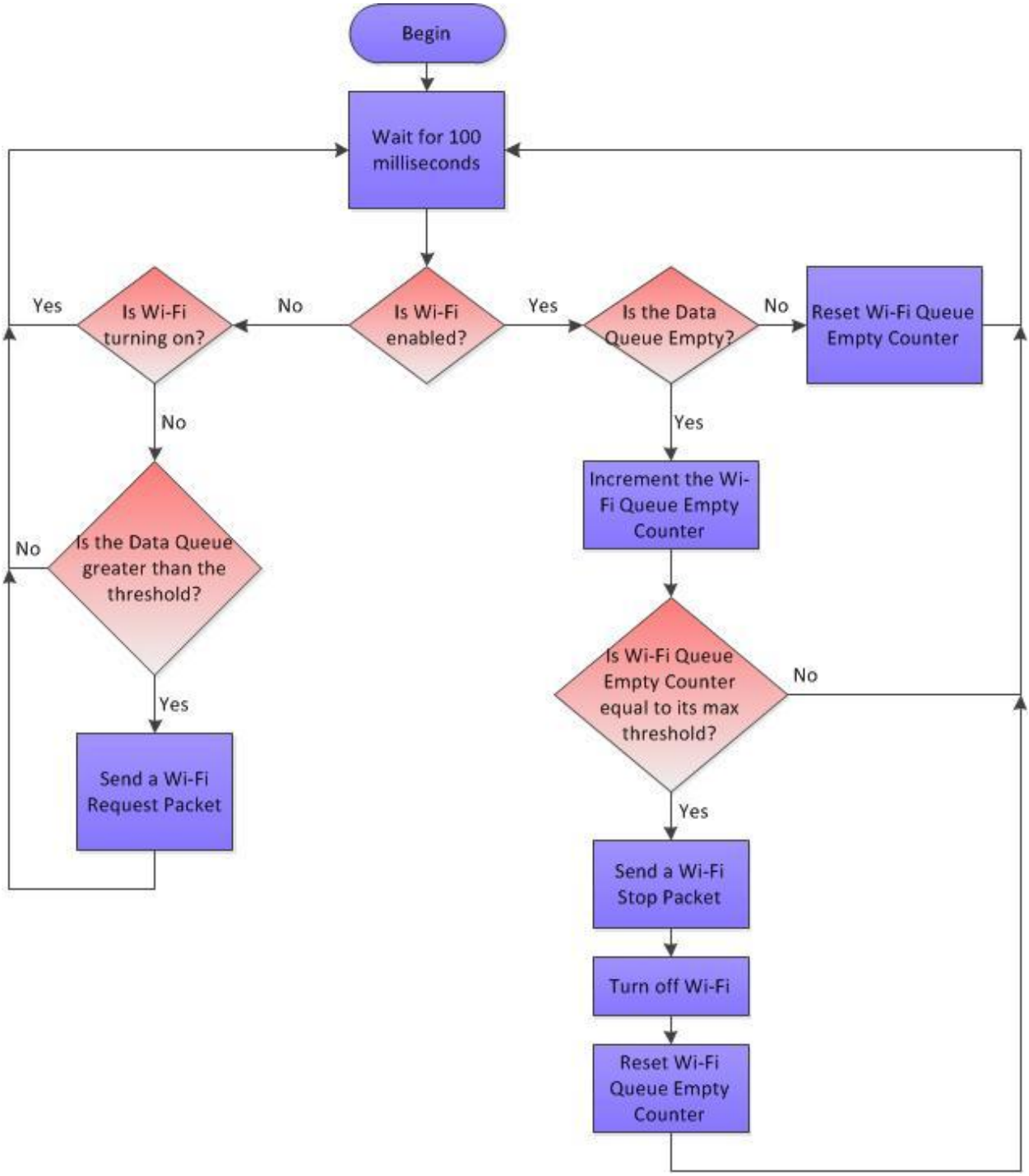


Figure 23: Bandwidth Monitor Flowchart.

4.3 Power Measurement System

In order to evaluate the software controlled radio system created in this project it was necessary to develop a method to measure and analyze the system's power consumption. The purpose of the power evaluation system was to measure and record the communications power consumption.

The hardware for the power evaluation system was to be integrated within the existing hardware of the computer system to monitor and measure the power and energy of the radio devices and the total system. For the first step the team decided to implement the system on a less complex level for trial and experimental purposes. The idea was to measure the power consumption and energy efficiency of a USB Wi-Fi adapter. Once the experiment was tested and concluded to be an accurate and verifiable method for measuring power consumption and energy efficiency it was documented and repeated for the ZigBee network interface and finally repeated for the combined Wi-Fi and ZigBee network. To accomplish this first step the team constructed the setup shown in Figure 24: .

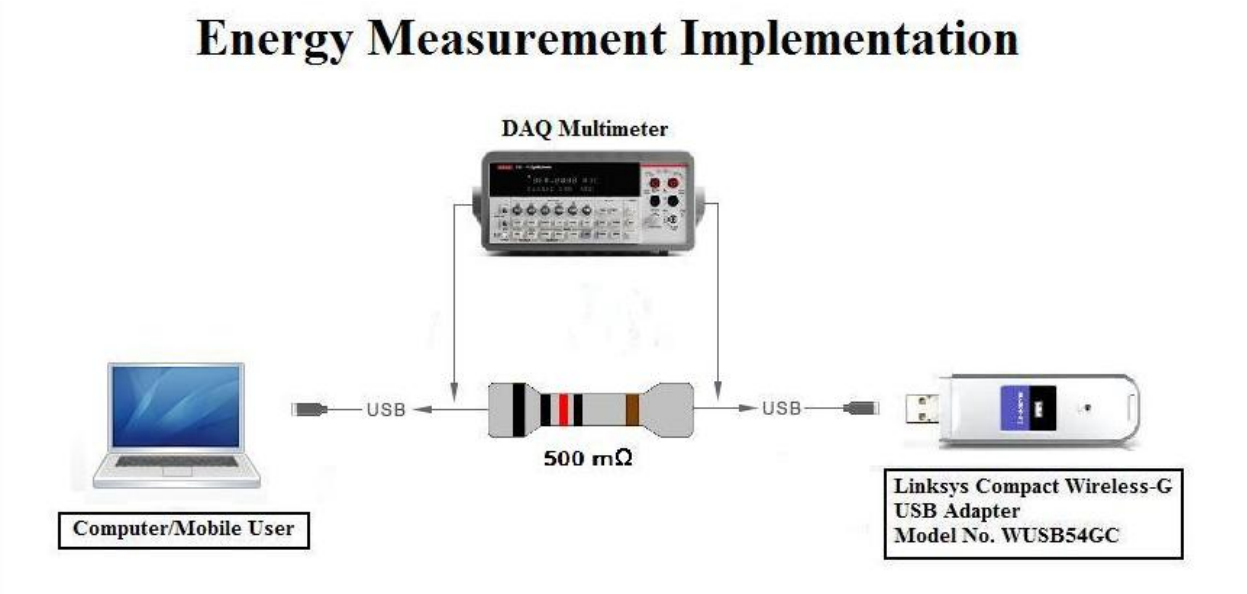
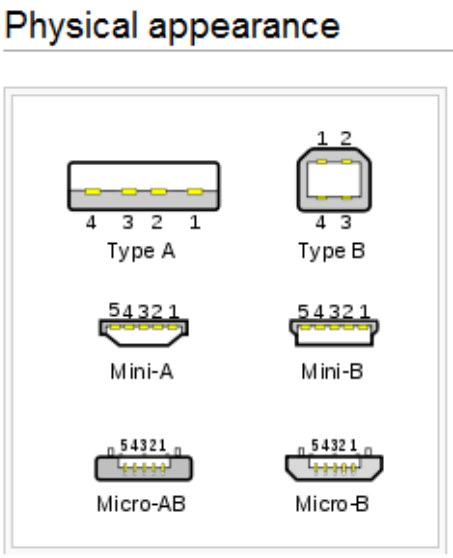


Figure 24: Energy Measurement Design. To measure the power being consumed by the wireless radios (Wi-Fi and ZigBee) a resistor is placed in series between the wireless USB adapter and the computer. By measuring the voltage difference across this resistor it is possible to calculate the power consumed by the wireless radio. The DAQ Multimeter is used to measure and record the voltage differences across the resistor as the radios run.

Many approaches were taken to measure the current, power, and energy consumed by the energy efficient mobile radio device discussed in this paper. This section goes over the processes taken to measure the communications' current, power, and energy consumed. It also explains the technique used to measure the total computer system power consumption and the technique use to calculate energy efficiency.

Communications' Current Draw and Power Consumption Measurement Techniques

Two techniques were considered to measure the current draw (mA) and power consumption (mW) of the network cards and the ZigBee-Wi-Fi network. Both techniques considered involved inserting an object in series with the power supply line (VCC) of a USB cord. Standard type A USB ports consist of four pins; VCC (+5 V), D- (Data -), D+ (Data +), and GND (Ground), as shown in Figure 25. To avoid damaging the USB cords of the network devices a USB extender was chosen to be cut in half.



USB 1.x/2.0 standard pinning

Pin	Name	Cable color	Description
1	VCC	Red	+5 V
2	D−	White	Data −
3	D+	Green	Data +
4	GND	Black	Ground

Figure 25: Standard Type "A" USB pinning diagram (Wikipedia.org).

The first method considered was to solder a resistor in series with the power supply line (VCC) of a USB extender. The voltage across the resistor is measured by a DMM and using Equation 1it is possible to calculate the Current (mA) draw of the USB device. Then using Equation 2 the power (mW) can be calculated.

Equation 1: Ohm's Law

$$I = \frac{V}{R}$$

Equation 2: The Power Equation

$$P = IV$$

The second method considered was to place an ammeter in series with the power supply line (VCC) of the USB cord. This method would directly measure the current, avoiding the use of Equation 1: Ohm's Law. However, ideally ammeters have a resistance of zero, but in reality this is not the case. When the ammeter was connected in series with the USB power supply its resistance was too much and produced a much larger voltage drop than intended. This caused the USB powered device to perform

poorly and not at its optimum performance levels, resulting in lower bit rates for the Wi-Fi adapter. For this reason the first method was chosen. By inserting a resistor the team could control exactly how much resistance was placed in series with the power supply and avoid the use of subpar lab equipment. A 500 mΩ resistor was used.

The Total Computer System Power Consumption Measurement Techniques

The techniques mentioned above to measure power and energy were concerned with the communications' energy consumed by the wireless radio cards in their various states. However, the additional algorithm used to switch between protocols presented in this project incurs a penalty to the overall power of the system due to the extra computations performed by the CPU. This project's purpose is to create an energy efficient radio that saves battery power. This means that the energy consumed by the algorithm must not be so great that it reduces battery life comparatively. To prove that this project's battery life is improved the team needed to not only measure the communications' energy, but also the total system's energy and compare this to a mobile PC without the multi-protocol radio.

Several approaches were considered to measure the computer/mobile user's total power and energy consumption. The most direct method would be to insert a resistor between the battery and the computer, as shown in Figure 26, similar to the method used to determine the energy consumption of the USB wireless network devices.

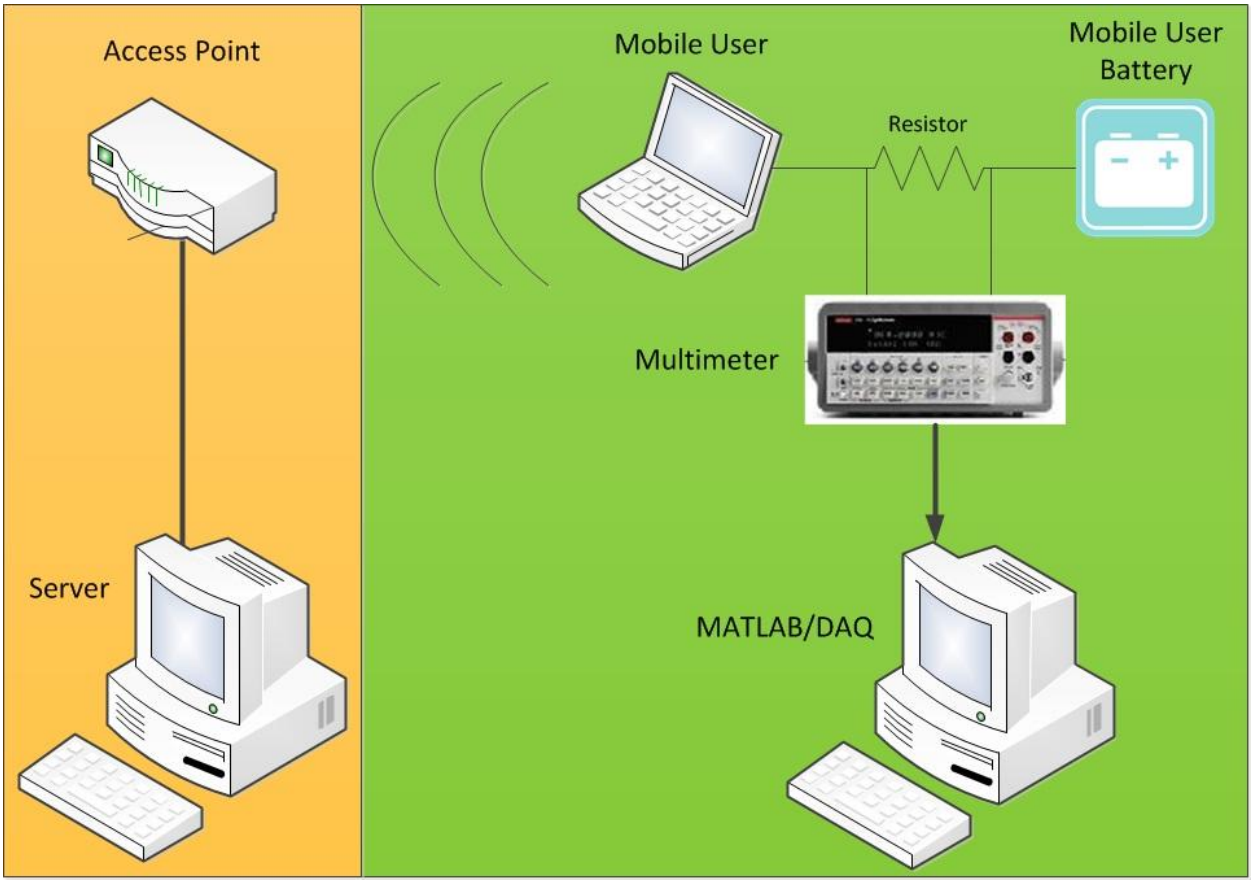


Figure 26: Technique for measuring the voltage difference across a resistor for calculating power consumption (W).

However, the battery of a laptop is much more difficult to connect a resistor in series with than a USB cord, as shown in Figure 27. One obstacle is the high current that is generated by the laptop battery. A carbon resistor, like the one pictured in Figure 24, can take up to 5W of power before it fails. An Acer Aspire 5520-5142 consumes about 20W when Idle. To measure the power consumption of the entire laptop a wirewound resistor or a heat sink would be required. Furthermore, the resistor could not be simply attached to the laptop battery. Figure 27 contains a picture of an Asus Eee PC battery. The contacts for the battery would need to be rigged in a unique and creative manner in order to place a resistor between it and the laptop. Due to this difficulty other methods were looked into.

Figure 27: Asus Eee PC 701 Asus Eee PC 2G Laptop Batteries (http://www.laptopbatteryinc.co.uk/).

To avoid these complications the project team looked into software means of measuring the power and energy consumption of a laptop. Eventually the group found that the Linux program dstats would be best suited for this purpose. Dstats is a versatile tool for generating system resource statistics. Dstats performs multiple actions that lend themselves well to scientific documentation and this project. Dstats records the exact time and date of the measurement taken. It can take measurements in increments of down to 1 second. It measures the power consumed (W) and also the percentage of the battery capacity remaining as well as the milliampere-hour **(**mAh) remaining. Another attractive attribute of dstats is its ability to write the recorded data to a .CSV file, which can be imported to OpenOffice Spreadsheet or Microsoft Excel. Using this tool the team was able to graph the battery life and power consumption of a mobile device/laptop.

Energy Efficiency and Performance (J/MB) Measurement Techniques

Energy efficiency is measured in Joules per Megabyte (J/MB). This measurement reflects that a more efficient system should consume the least amount of joules to transmit each byte. This value cannot be measured directly, so to calculate this value the group used data acquired from the other techniques. When performing the performance measurements tests the duration of the test, the average voltage difference across the resistor, and the total bytes transmitted or received were recorded during experiments. Using the calculated power (Watts), the duration of the test (seconds), and the total amount bytes transmitted and/or received it is possible to calculate the energy efficiency of the system, in Joules per Megabyte. The equations used for the calculation are shown below.

If, $$Watts = \frac{Joules}{second}$$

Then, $$\frac{Joules}{second} * \frac{seconds}{Megabytes} = \frac{Joules}{Megabyte}$$

Then the energy per megabyte is calculated by the equation…

$$\frac{Joules}{Megabyte} = \frac{Power * Duration\ of\ Test}{Megabytes\ Transmitted\ and/or\ Received\ during\ the\ test}$$

This value is calculated for both the total system and the communications system.

4.4 Implementation Summary

This chapter provides the detailed description of the implementation of this project. It documents the hardware and software decisions and problems experienced during this project's development. The project was split into three sections. The wireless network ultimately created consists of a ZigBee radio and the program written to control the communication features of said radio, a Wi-Fi radio that transmits files through the same program as ZigBee. Another program was developed to control the switching aspect between these two radios. Finally, a power measurement system was constructed to measure and record the power consumption of both these radios to prove the efficiency of the overall wireless network. The next chapter, Chapter 5: Results, discusses the results of the network implemented.

Chapter 5: Results

In the previous chapter the team discussed the methods used to develop the sections of the project. In order to determine if the multi-protocol Wi-Fi ZigBee network would in fact save energy the team needed to analyze it in pracice. The methods by which the team decided to measure the different criteria such as the current draw, power consumption, and energy consumption were described in detail in the prior section. In this chapter the results obtained from each of the different networks (Wi-Fi Only (CAM), Wi-Fi Only (PSM), ZigBee Only, and Wi-Fi and ZigBee combined (WiZ)) are discussed. The general test bed used to obtain these results is shown in Figure 28.

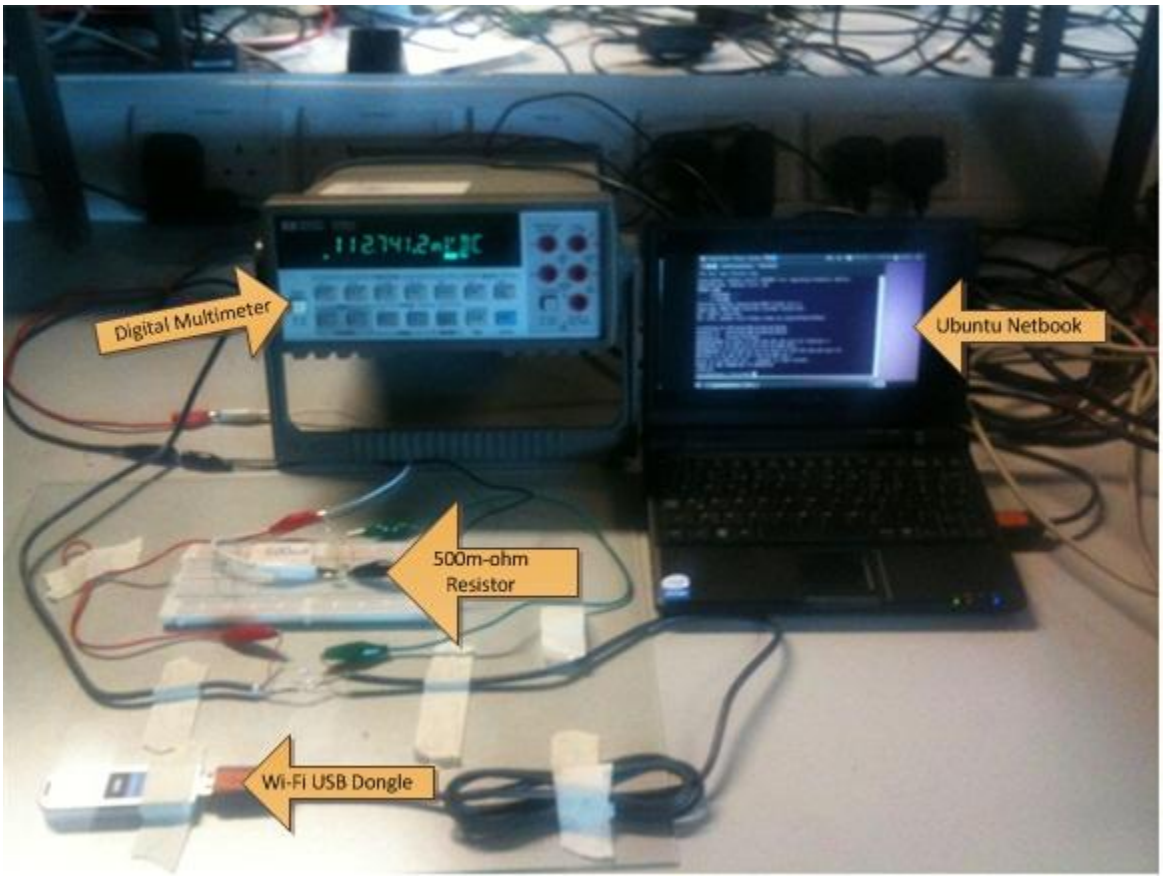


Figure 28: Energy Measurement Setup: The object labeled "resistor" is the breadboard that holds the resistors that form the 500 mΩ resistance in series with the Vcc of the USB cord that was cut. The power-supply-in (Vcc) was the only wire that was modified. Alligator clips connected the power-supply-in (Vcc) with the breadboard resistor setup then connected back to the power-supply-in, resulting in the series resistance of 500 mΩ. The Digital multimeter was used to record the voltage across the resistors. This multimeter was connected to a computer with MATLAB via a RS232 serial port. The Ubuntu Netbook was used as the mobile user in the experiment. The USB wireless network interface adapter used in the experiment is connected to the end of the USB cable.

The Wi-Fi network adapter is shown in this picture however for other experiments such as the ZigBee configuration the Wi-Fi adapter would be replaced. The network interface device is ultimately connected to the Asus Eee PC. Not included in Figure 28 is the computer configured to be the AP. This computer was also an Eee PC. The role of the AP computer, in this network, was to be the recipient during transmitting and the donor during receiving.

5.2 Procedure

Table 10: Sample Test Template for wireless network testing was created to provide the tester with a documented procedure to perform each test. Five different states of radio operation were considered in the test: Receiving, Transmission, Idle and Associated, Off, and a Webpage Simulation test. Four different wireless networks were tested: Wi-Fi Only (CAM), Wi-Fi Only (PSM), ZigBee, and the combined ZigBee Wi-Fi (CAM) network named WiZ.

Table 10: Sample Test Template for wireless network testing.

Sample Test Template Conduct these tests on the <***Network Name***>:	
Step	Description
1	Measure the average amount of energy, current, and power consumed by the wireless interface to **download (Receive)** files from the AP. Download the following files twenty times each and record the *voltage difference*, *total bytes*, and *duration*: a. 100 KB b. 1 MB c. 10 MB d. 100 MB
2	Measure the average amount of energy, current, and power consumed by the wireless interface to **upload (Transmit)** files from the AP. Send the following files twenty times each and record the *voltage difference*, *total bytes*, and *duration*: a. 100 KB b. 1 MB c. 10 MB d. 100 MB
3	Measure the average amount of energy, current, and power consumed by the wireless interface when **Idle and Associated** with the AP. Send and download no files and record the *voltage difference*, *total bytes*, and *duration*.
4	Measure the average amount of energy, current, and power consumed by the wireless interface when **Off**. Put the radio in a software controlled shut down and record the *voltage difference*, *total bytes*, and *duration*.
5	Measure the average amount of energy, current, and power consumed by the wireless interface during the www.wikipedia.org test case **(Intermittent data download)**. Record the *voltage difference*, *total bytes*, and *duration.*

In the first step, testing the receiving operating mode, the same four files (100KB, 1MB, 10MB, and 100MB) were now received. The AP was responsible for sending the files to the mobile computer. The mobile computer referred to was the item labeled "Ubuntu Netbook" in Figure 28.

In the second step, testing the transmission operating mode, four different files of different sizes were sent to the AP. The sizes chosen were 100KB, 1MB, 10MB, and 100MB. Each file was sent four to twenty times depending on the size of the file. Smaller files were sent more times and larger files were sent fewer times. Since larger files took longer to send it was easier to obtain many data points and so fewer files needed to be sent.

In the third step, testing idle and associated, the radio was associated with the AP but no noticeable data transfer was taking place. During this test no files are transferred. However, in typical networks beacons are sent out from the AP that contain important information for the user such as whether or not the user has an incoming transmission. As such, in idle mode the user experiences negligible network activity. In Adhoc networks beacons checking to see if the user is connected to the AP do not occur since there is technically no AP. In an Adhoc network both the computers are peers and merely transfer data between one another when necessary.

In the fourth step, testing the off state, the radio was put in a software controlled off. This was done through bash scripts that would echo instructions to the root USB hub of the Wi-Fi dongle. This would turn off the power to specific devices connected to the hub. Essentially through the use of these commands it was possible to achieve zero power consumption from the network interface.

The purpose of the fifth test was to monitor the radio during a typical internet webpage download. This is accomplished by running the browsing simulation script in Appendix A. The script was designed to simulate network activity that would be present during a typical webpage download. The data to create the script was obtained through various sources. The first source used was IMIX [42]. IMIX contains statistics about network activity such as common sizes of packets sent through the internet to create the various web pages, videos, and pictures that reside within the internet. IMIX is specifically used to simulate network activity for experimental purposes. To create the webpage simulation packets of these sizes were sent through our network. The team also recorded the network activity present upon visiting a website using bwm-ng to monitor and record the bytes going in and out of the network card as the website was accessed. Figure 29 contains the recorded network activity.

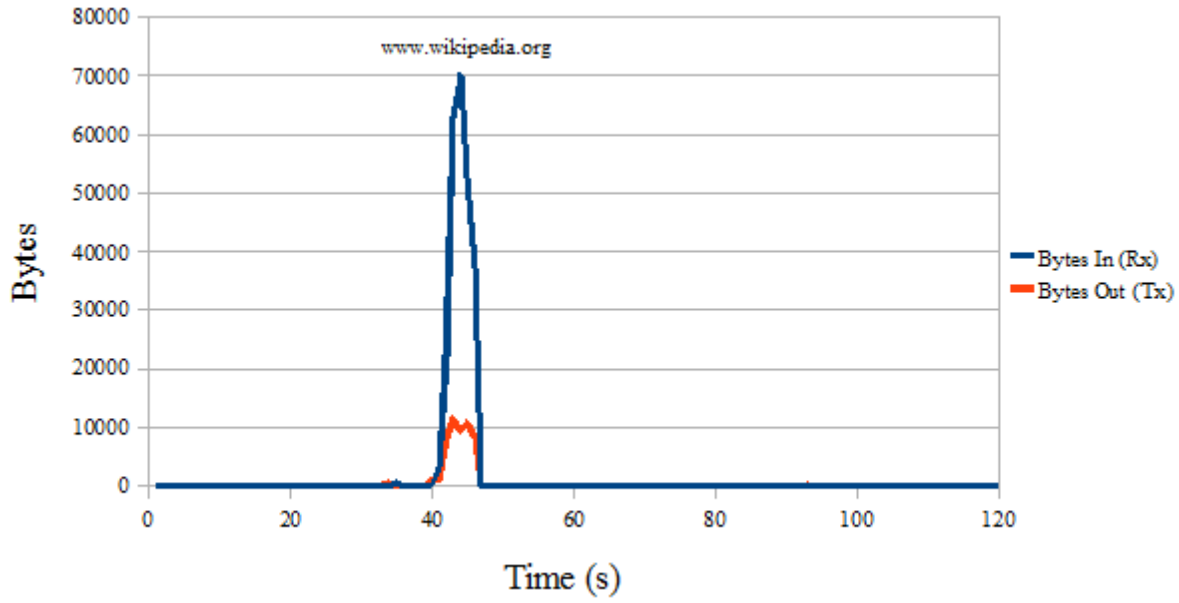


Figure 29: Recorded network activity during the www.wikipedia.org test case. The spike is the user accessing www.wikipedia.org.

The area under the curve is the total amount of data sent in order to produce the Wikipedia homepage. Calculating the area under each curve results in Table 11 below:

Table 11: Measured Website Sizes.

Website	Size (Kilobytes)
www.wikipedia.org	242.94

According to a study done at Binghamton University New York the average size of a web page is 130 Kilobytes [43]. With this information a script was written that simulated visiting a common internet website. The size of 130 KB was chosen to simulate, but due to network overhead the actual observed amount of data transferred is slightly higher. Using the average size of a web page and IMIX a script was created to simulate the network activity. This script would send files of common packet sizes, obtained from IMIX. The sum of the files was intended to accumulate to 130 KB. The script can be viewed in Browsing Simulation Script.

5.3 Testing

With the procedures outlined and documented the team was ready to begin the testing of each network. Four networks are tested in this report; Wi-Fi (CAM), Wi-Fi (PSM), ZigBee, and WiZ. Each of these networks underwent the outlined procedures and the results and observed of each were recorded. In section 5.4 Overall Results the results of all the networks are compared.

Wi-Fi Only Network in Continuously Active Mode (CAM)

The first network that was tested was the Wi-Fi only network in CAM mode. CAM stands for continuously active mode; this mode is being replaced by the newer, more power efficient Power Saving Mode (PSM). However, it provided a basic baseline network to serve as a benchmark to compare values to. The setup for this test is identical to the setup described in Figure 28. In order to create a Wi-Fi only CAM network two factors need to be addressed. The first factor is the wireless network card's capabilities. The network card can be CAM or PSM ready, any Wi-Fi network card can operate in CAM. The second factor is that if the Access Point (AP) is not able to handle PSM then the wireless card will perform in CAM mode regardless if it is PSM ready or not. So, to create a CAM network an Adhoc network was set up. An Adhoc network was chosen because they cannot facilitate PSM. The results from the Wi-Fi CAM network tests are displayed below.

Table 12: Wi-Fi CAM Current (A), Power (W), and J/MB observed values.

Wi-Fi (CAM)				
		Current (A)	**Power (W)**	**J/MB**
Tx	Average	0.30026	1.45407	1.79367
	Std Dev	0.00101	0.00566	0.21334
Rx	Average	0.28776	1.40273	1.91553
	Std Dev	0.00088	0.00605	0.23249
Idle	Average	0.26139	1.27956	MB ≈ 0
	Std Dev	0.00447	0.01416	MB ≈ 0

One purpose of this table is to verify that the values obtained in the experiment were consistent with typical values observed in other experiments and specification sheets. The other purpose is that the values here were also used to develop graphs, plots, equations, and other means of displaying the data.

Table 13: Measured Summary Statistics of Wi-Fi (CAM). The data transferred includes Rx and Tx as measured through the network interface and includes protocol overhead.

Wi-Fi (CAM) Summary Statistics						
Test Cases	**Time over Wi-Fi (s)**	**Data Transferred (MB)**	**Data Pattern**	**Data Rate (Mbits/s)**	**Total Energy Consumed (J)**	**Energy Efficiency J/MB**
File Transfer-1	14.8	11	Bulk Data	5.4	21.520	2.152
File Transfer-2	110	104	Bulk Data	7.3	159.948	1.599
www.wikipedia.org	120	.1435	Intermittent	6.6	153.883	1070.357
Idle	20	0	None	0	25.642	N/A

These results will serve as a benchmark to compare other measured values to from the other network interface configurations. The test cases represent different scenarios that could be encountered in a network. These cases will serve to illustrate and compare the strengths and weaknesses of each network configuration discussed in this report.

Wi-Fi Only Network in Power Saving Mode (PSM)

For this network the mobile computer needed to be setup to operate in PSM. To operate in PSM a network interface device and the AP must be able to support this function. A personal computer cannot perform this which is why Adhoc was not used for this test. In this experiment the project team used the existing wireless network at the University of Limerick as the wireless network connection through which the files would be transferred. Aside from this modification the test bed and tests remained the same. The results from the Wi-Fi Only (PSM) network tests are displayed below.

Table 14: (PSM) Current (A) and Power (W) observed values.

Wi-Fi (PSM)				
		Current (A)	**Power (W)**	**J/MB**
Tx	Average	0.29175	1.41549	6.12511
	Std Dev	0.01222	0.05774	2.59267
Rx	Average	0.28106	1.36567	3.34356
	Std Dev	0.02060	0.09803	0.44481
Idle	Average	0.26209	1.27604	MB ≈ 0
	Std Dev	0.01137	0.05425	MB ≈ 0

In terms of power usage the PSM network exhibits lower power consumption (Watts) which would imply that it is also more energy efficient. However, the energy efficiency (J/MB) of PSM is much worse than the energy efficiency of CAM. The reason for this is that the network used to transmit data for PSM was the campus Wi-Fi network. As a result of this the data rates experienced during testing the PSM network are much lower. Table 15 displays the differences in data rates experienced by both networks.

Table 15: The average observed data rates for the Wi-Fi CAM and PSM networks.

Average Observed Data Rate (Mbits/sec)		
Network	**Average**	**Std. Dev.**
Wi-Fi (CAM)	6.599781453	0.763531169
Wi-Fi (PSM)	2.550100056	1.267728853

This could be due to multiple factors such as a weaker signal due to the distance between the mobile user and the wireless access point, or there could have been significant traffic while the test was taking place. It is also important to note that the standard deviation of the PSM network is high which means that the bit rate was not consistent throughout the tests. Because of these uncontrollable data rates the team decided to use the data rate acquired from the CAM network for the PSM network. The reason the team chose to use the CAM data rate for PSM is due to the effect the data rate has on the energy efficiency. If a 100MB file is transferred on the PSM network it will take longer than it would on the CAM network, since the CAM network has a higher data rate. So even though, in reality, PSM is more energy efficient (thus the name Power Saving Mode) the longer time spent transferring data will always results in a higher amount of energy consumed (Joules), even though the PSM network exhibits lower power usage (Watts).

So, the problem is that the lower data rate results in a longer transfer time and more joules consumed. This means that because of the less than ideal data rate of the uncontrollable campus network the PSM network will never be more energy efficient than the isolated CAM network. However, the poor data rate of the campus network is not intertwined with the project presented in this paper. The team witnessed reliable transfer power (Watts) values. These values are independent from the data rate and still provide a solid foundation for the energy efficiency properties of the PSM network. Ideally Wi-Fi PSM and CAM should have the same data rates so this is what is assumed in this project.

ZigBee Only Network

To prepare the test bed for the ZigBee networks tests the only modification needed is to replace the Wi-Fi wireless network adapter with a ZigBee wireless network adapter. After performing the test Table 16 was produced.

Table 16: ZigBee Power and Current Usage.

ZigBee			
		Current (A)	**Power (W)**
Tx	Average	0.07300	0.36500
	Std Dev	0.00070	0.00120
Rx	Average	0.07260	0.36000
	Std Dev	0.00078	0.00390

No idle power was measured because ZigBee does not have an idle state. The energy efficiency (J/MB) was measured, however the ZigBee network developed in this project performed data transfer so poorly that the energy efficiency was absurdly high and the deviation between different file sizes energy efficiency was in the 100's. To describe how poor the data transfer was Table 17 was constructed.

Table 17: Average data rates obtained from the ZigBee network.

Average Observed Data Rate (Mbits/sec)		
Network	**Average**	**Std Dev**
ZigBee	0.003906	0.00016

This value equates to 4 kilobits/s. Even though the ideal data rate is published to be 250 Kbits/s the actual value is inherently much lower as seen with the Wi-Fi as well. There are several possible causes for this dilemma such as the substantial network overhead due to the amount of bulky software needed to create the ZigBee network. Since ZigBee is not a mainstream wireless network there was little development put into the product before it was put on the shelves, unlike Wi-Fi. As a result the team needed to program the entire ZigBee interface and file transfer system. Fortunately ZigBee has other uses than data transfer.

The plan for ZigBee was that it would handle small data transfer then when its bandwidth became overwhelmed Wi-Fi would turn on and handle the heavy traffics. However, with the abhorrent data rate observed in the tests any data transfer through ZigBee was impractical. The solution to this was to use ZigBee as a sensor to communicate with the AP. When the AP needed to send files ZigBee would

attempt to send the file and quickly reach its max bandwidth, in which case Wi-Fi would turn on. And so ZigBee lost its role as a method for low data transfer and instead became a sensor that would switch Wi-Fi on when an incoming or outgoing transmission was needed.

Wi-Fi-ZigBee Power Saving Network, WiZ (Uses CAM for Wi-Fi)

To set up the test bench for this network both the Wi-Fi and ZigBee network adapters were plugged into the mobile user and the AP. 500mΩ resistors were placed between each network adapter and their respective power supply. An Adhoc network was and thus the WiZ uses Wi-Fi (CAM) for its Wi-Fi portion of functionality. The tests were then carried out in the same fashion as previously explained. The measured parameters of the network are displayed in Table 18.

Table 18: WiZ measured values. WiZ is a combination of two wireless networks: Wi-Fi (CAM) and ZigBee.

WiZ: Wi-Fi (CAM) in combination with ZigBee				
		Current (A)	**Power (W)**	**J/MB**
Tx	Average	0.36061	1.76171	2.2726
	Std Dev	0.00413	0.55	0.33472
Rx	Average	0.359	1.754	2.15636
	Std Dev	0.0027	0.014	0.02857
Idle	Average	0.0726	0.36	N/A
	Std Dev	0.00078	0.0039	N/A

The WiZ network experiences slightly higher power usage than the Wi-Fi only networks due to the addition of the ZigBee running all the time. This causes the WiZ network to consume more energy (Joules) when transferring data. The strength of this network is its idle state. WiZ's idle state is constituted of only ZigBee, because in this idle state the Wi-Fi is completely turned off. By this logic this network is beneficial to wireless network users that remain in idle state most of the time the device is running. For most personal wireless devices this is typically the case.

WiZ Browsing Simulation

A typical internet browsing simulation was created in order to test the functionality of the WiZ network. This was done similarly to the webpage simulation. Data packets of sizes acquired from IMIX were transferred to simulate the network activity during the recorded internet browsing session. A script was written that would send these packets in a fashion that mimics the network activity of the session. The script is located in Appendix A. The browsing session was based off an actual browsing simulation performed by the project team. The actual network activity of this session was recorded and is displayed in Figure 30.

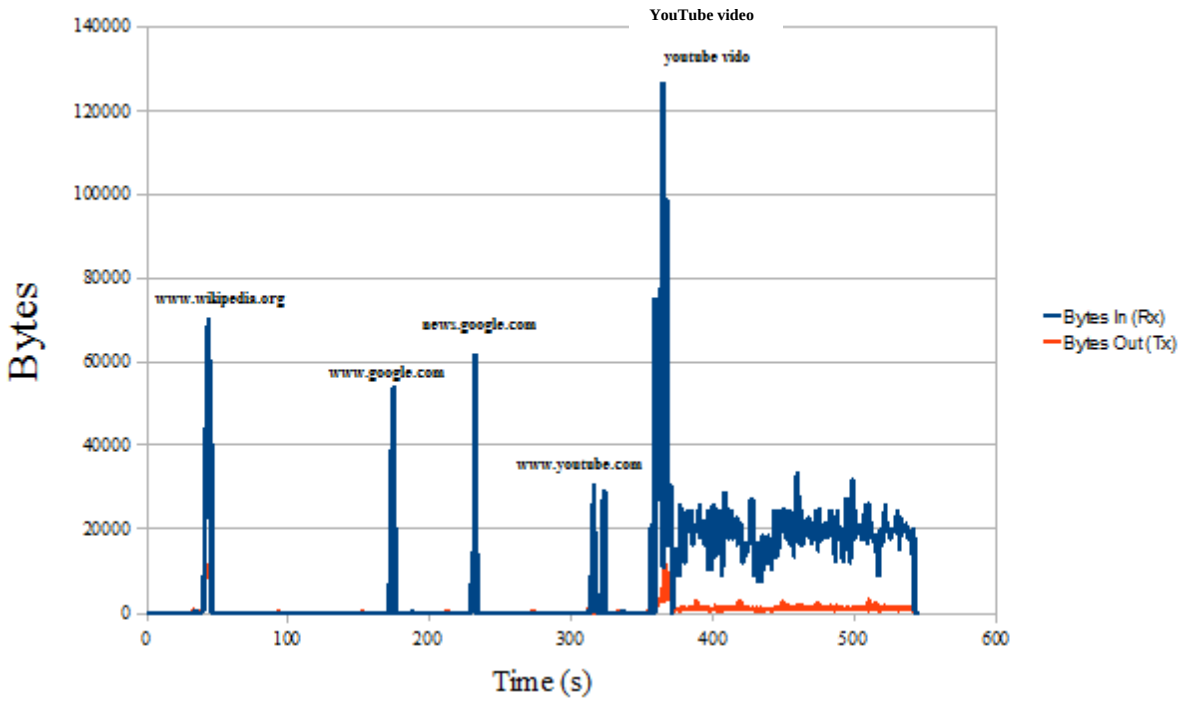


Figure 30: Recorded network activity during the internet browsing session. The notes indicate what each spike is a result from. The first spike is when the user accessed Wikipedia.org.

The power of each wireless component of WiZ was measured while the simulation was run and Figure 31 was produced. Figure 31 consists of the ZigBee and Wi-Fi (CAM) networks that cooperate to form the WiZ network. The idle value of the Wi-Fi (CAM) is zero due to the software introduced by WiZ. When the radio is idle it turns off Wi-Fi completely.

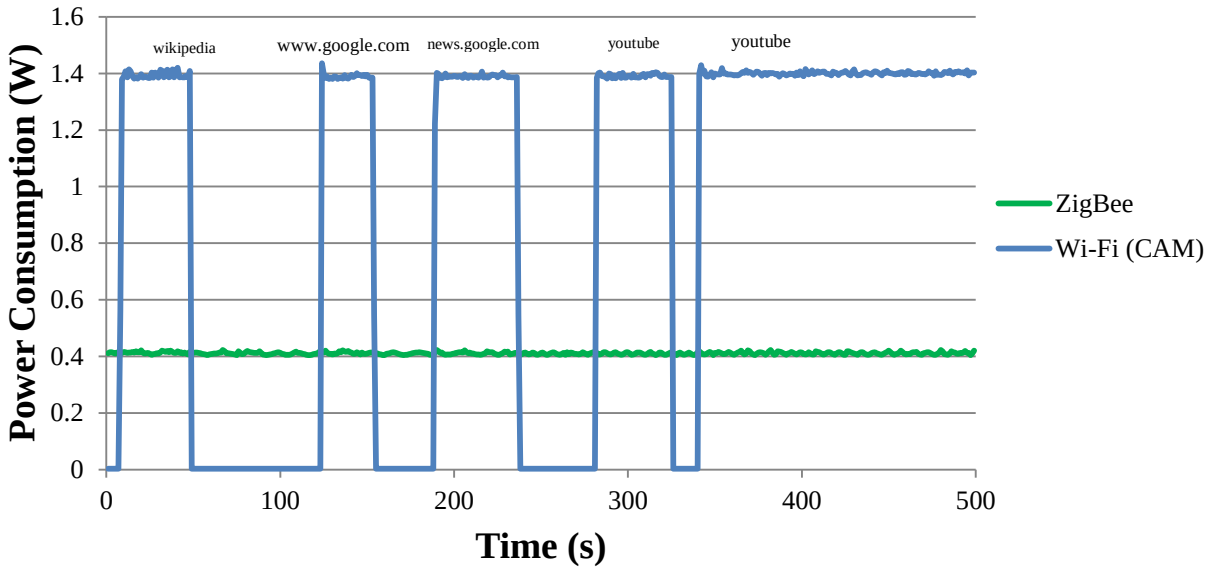


Figure 31: Recorded power consumption of the ZigBee and Wi-Fi (CAM) wireless devices that compose WiZ. Recorded during the browsing simulation test.

Since WiZ is not a single network device its power could be measured directly. So, to measure its power consumption the team added the values of the two wireless networks to produce the graph of the power consumption of WiZ, Figure 33. Figure 32 shows the two network components, ZigBee and Wi-Fi (CAM), alongside the WiZ, it is by adding these two components that WiZ is formed. The idle of CAM seems to be zero but that is only due to WiZ. In the WiZ network Wi-Fi (CAM) is turned completely off while the network is idle. ZigBee remains at the same power level the whole duration and so it becomes the idle state of WiZ and it is also present during the transfer state. Since it is present during the transfer state the WiZ transfer power is equal to the Wi-Fi (CAM) transfer power plus the ZigBee power.

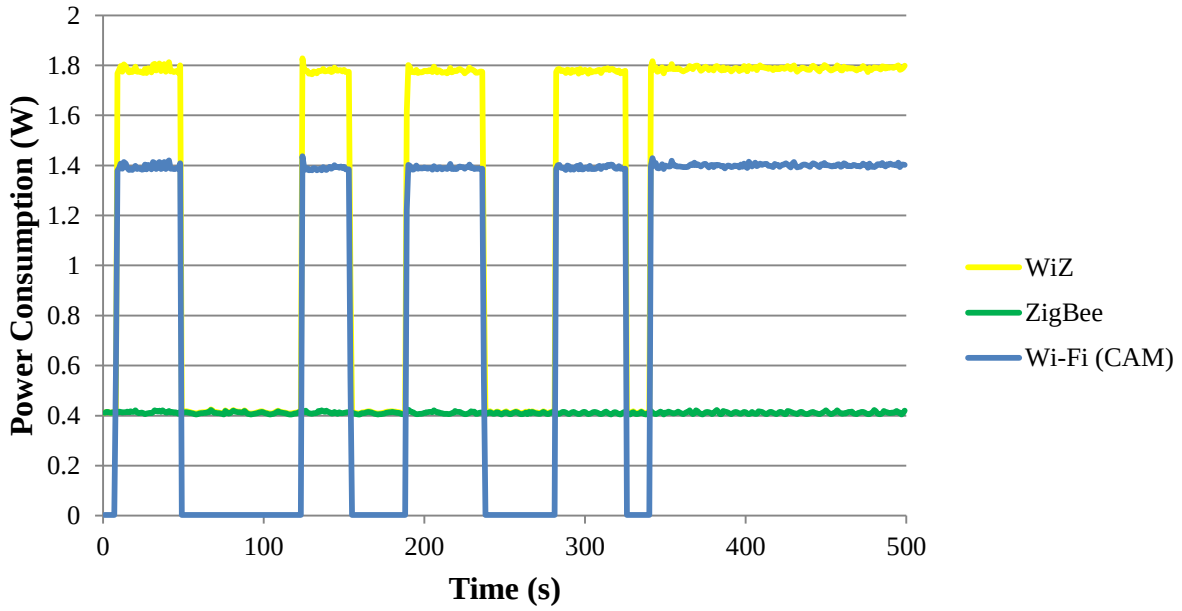


Figure 32: Power consumption of WiZ (yellow), ZigBee (green), and Wi-Fi(CAM) (blue) during the browsing simulation test. The WiZ values are equal to the CAM values added with the ZigBee values.

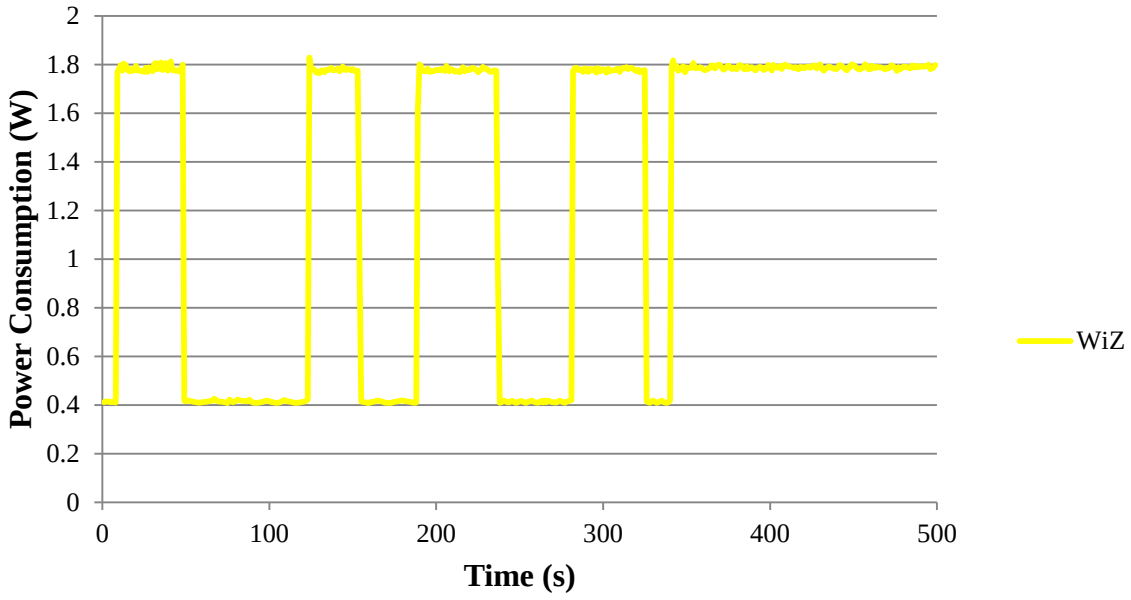


Figure 33: The power consumption of the WiZ network during the browsing simulation test. Its idle power is equal to that of ZigBee and its active transfer power is equal to the ZigBee power plus the Wi-Fi transfer power. It is this addition of the ZigBee power during transfer that causes WiZ to consume more power than the traditional CAM and PSM networks.

5.4 Overall Results

The goal of this project was to create an energy efficient wireless network using multiple wireless protocols to save energy on mobile platforms. To effectively compare each wireless network configuration each network had its parameters measured and a set of cases were developed to compare the networks in different scenarios encountered in common wireless networks. The parameters measured were transmit power, transmit current, receive power, receive current, idle power, and idle current. Table 19 provides a summary of these measured parameters.

Table 19: Summary table of the measured parameters for each network configuration.

Compiled Experimental Data						
	Tx Current (A)	**Tx Power (W)**	**Rx Current (A)**	**Rx Power (W)**	**Idle Current (A)**	**Idle Power (W)**
Wi-Fi (PSM)						
Average	0.2918	1.4155	0.2811	1.3657	0.2621	1.276
St Dev	0.0122	0.0577	0.0206	0.098	0.0114	0.0542
Wi-Fi (CAM)						
Average	0.3003	1.4541	0.2878	1.4027	0.2614	1.2796
St Dev	0.001	0.0057	0.0009	0.006	0.0045	0.0142
ZigBee						
Average	0.0829	0.388	0.0726	0.365	0.0726	0.36
St Dev	0.0008	0.004	0.0008	0.0039	0.0008	0.0039
WiZ						
Average	0.3606	1.7616	0.359	1.754	0.0726	0.36
St Dev	0.0041	0.55	0.0027	0.014	0.0008	0.0039

Test Cases

Multiple test cases were developed in order to evaluate the networks as well. The cases were determined analyzing common several scenarios that wireless networks encounter on a daily basis. The test cases consisted of two file transfers; one file transfer of 10MB and the other a 100MB file. These tests serve to simulate bulk data transfer. This could be seen as downloading a file from a website, such as a picture, document, or sort of large data transfer. Since downloading is much more common for personal wireless users these files were downloaded, not uploaded. The time over Wi-Fi was the average time it took for the network to download a file of that size

The next test case was called www.wikipedia.org. This case simulates accessing a website. To simulate this the test accesses Wikipedia.org by downloading .1435 Megabytes of data then idles for remaining time. The reason for the idle time is that in many cases, such as going to Wikipedia, the user pauses to read the website upon loading. It also introduces downloading and idling into a single test case

to demonstrate the effect of entering more than one mode of operation during a period of time. This case is labeled intermittent because data transfer in this test case in not constant such as in the previous two test cases; file transfer 1 and 2. The final test case is Idle. This case tests the idling of each network. Since wireless radios spend most of their time in the idle state it is important to observe the differences in this state among the networks. The results of these test cases are shown in Table 20 below.

Table 20: Table of test cases created to analyze each wireless network.

Test Cases	Time Over Protocol	Data Transmitted (MB)	Data Pattern	Max Data Rate (Mbits/s)
Wi-Fi (CAM)				
File Transfer 1	14.8	10	Bulk Data	5.40541
File Transfer 2	110	100	Bulk Data	7.27273
wikipedia.org	120	0.1435	Intermittent	6.6
Idle	20	0	None	0
Wi-Fi (PSM)				
File Transfer 1	12.12	10	Bulk Data	6.60066
File Transfer 2	121.21	100	Bulk Data	6.60012
wikipedia.org	120	0.1435	Intermittent	6.6
Idle	20	0	None	0
ZigBee				
File Transfer 1	20480	10	Bulk Data	0.00391
File Transfer 2	204800	100	Bulk Data	0.00391
wikipedia.org	293.888	0.1435	Intermittent	0.00391
Idle	20	0	None	0
WiZ (Wi-Fi (CAM) Combined with ZigBee)				
File Transfer 1	11.667	10	Bulk Data	6.85714
File Transfer 2	120	100	Bulk Data	6.6664
wikipedia.org	120	0.1435	Intermittent	6.6
Idle	20	0	None	0.36

Test Cases	Average Tx Power Usage (W)	Total Energy Consumed (J)	Energy Efficency (J/MB)
Wi-Fi (CAM)			
File Transfer 1	1.45407	21.52027	2.15203
File Transfer 2	1.45407	159.94797	1.59948
wikipedia.org	1.45407	153.88321	1070.23065
Idle	1.28211	25.64222	N/A
Wi-Fi (PSM)			
File Transfer 1	1.41549	17.15575	1.71557
File Transfer 2	1.41549	171.57164	1.71572
wikipedia.org	1.41549	153.14872	1067.23846
Idle	1.27604	25.52074	N/A

ZigBee			
File Transfer 1	0.388	7475.2	747.52
File Transfer 2	0.388	74752	747.52
wikipedia.org	0.388	107.26912	747.52
Idle	0.36	7.2	N/A
WiZ (Wi-Fi (CAM) Combined with ZigBee)			
File Transfer 1	1.81907	21.22251	2.12225
File Transfer 2	1.81907	218.2887	2.18289
wikipedia.org	1.81907	43.45379	302.81387
Idle	0.36	7.2	N/A

Evaluation of the Energy Consumption and Efficiency of the Wireless Networks

Energy efficiency measures how many joules are spent to transmit a megabyte of data. This metric illustrates how effective a system is at using its energy. The efficiency is also directly related to the amount of Joules spent during each case. The advantage of the energy efficiency metric (J/MB) is that it is independent from the variables of each test such as the duration and megabytes transferred. The following graphs display the energy efficiency and consumption experienced during each of the test cases mentioned above.

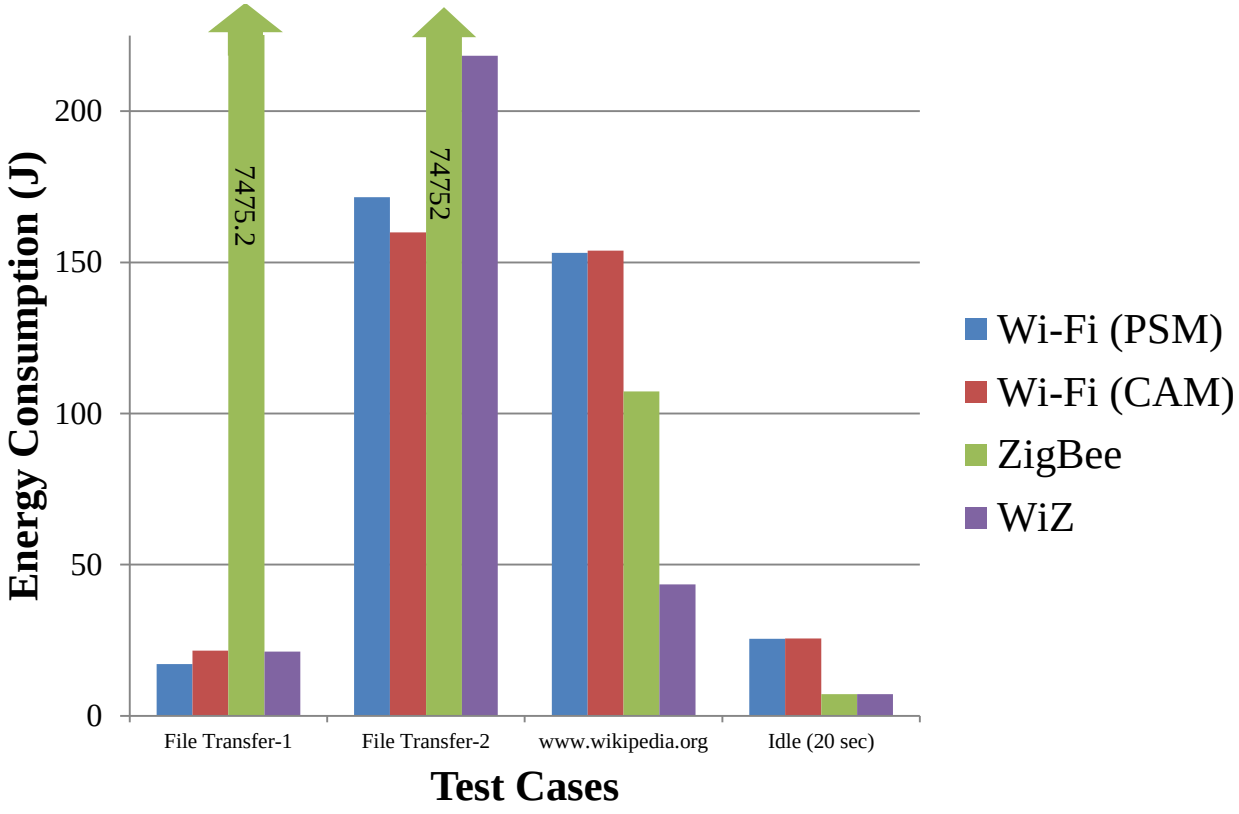


Figure 34: Energy consumption of wireless test cases. Since ZigBee is not meant for large file transfers such as 10-100MB its energy consumed was literally off the charts, so a green arrow was inserted to illustrate this. The value in the green bars is the amount of joules consumed by ZigBee in that test case.

Energy is equal to the power multiplied by time. The energy consumed by a wireless radio increases as its transfer time increases and time increases as data rate decreases. So the biggest influence on the energy is the data rate and the power usage. The first two test cases, File Transfer-1 and File Transfer-2, illustrate this property in the figure above. Wi-Fi (CAM), Wi-Fi (PSM), and WiZ all consume relatively similar amounts of energy. This is because their data rates, or bandwidths, are almost identical. The primary factor in their difference is energy consumption is their power consumption (W). Since WiZ consists of both Wi-Fi (CAM) and ZigBee running while transferring data it consumes the

most power in this mode of operation; making it the least appealing candidate for pure bulk data transfer, besides ZigBee. In spite of this, since the average data rate of WiZ was higher for 10MB files the energy efficiency of WiZ performed better than that of CAM in this case. This is how the data rate affects the energy consumption of the wireless networks. If we assumed them to experience the same data rates then CAM would achieve a more efficient energy consumption rate in any bulk data transfer. Figure 36 is the same graph as Figure 35 except that the data rates of each network, except ZigBee, are all equal. In this graph we can see how the high transmission power of WiZ causes it to be less efficient than the other Wi-Fi networks. ZigBee fairs the worst for bulk data transfer reaching energy consumption values literally off the chart. In the case of Figure 37 the data rate of Wi-Fi (CAM) is much higher than in the previous file transfer. As a result CAM's energy efficiency has dropped (become more efficient).

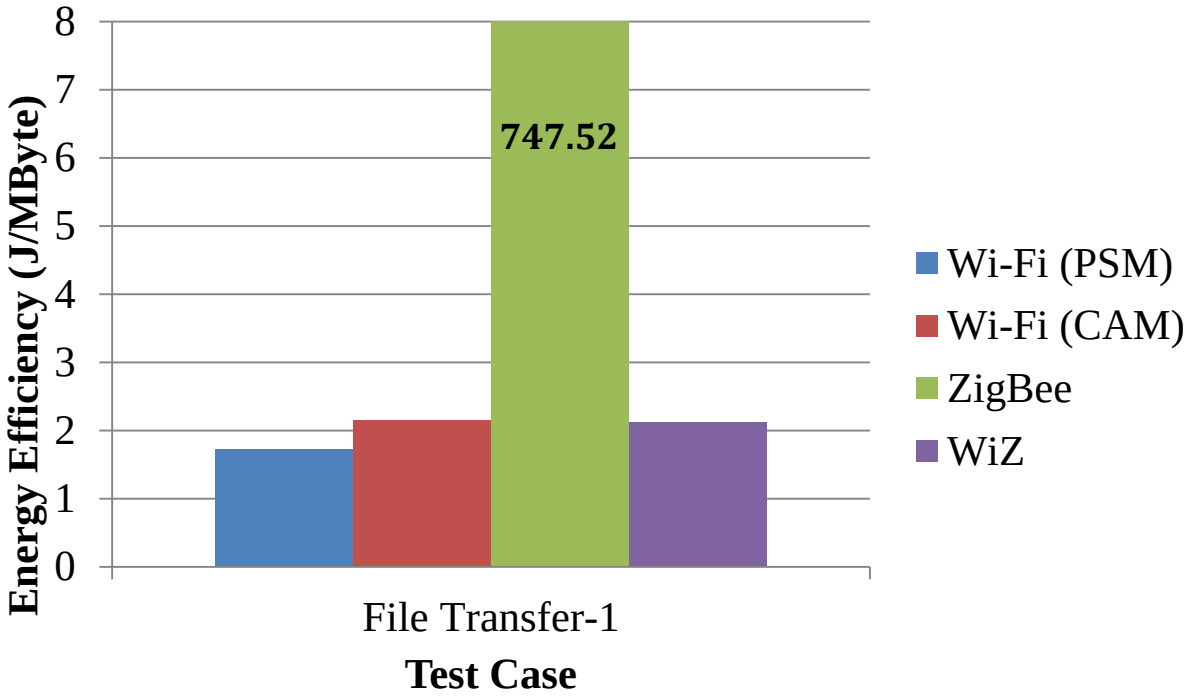


Figure 35: Energy Efficiency of Wireless Networks during the file transfer-1 test case. The number in the green ZigBee bar is the energy efficiency of the ZigBee network. The chart for the energy efficiency is very similar to the chart for energy consumption due to the similarity in test parameters, such as the time, megabytes transferred, and data rate.

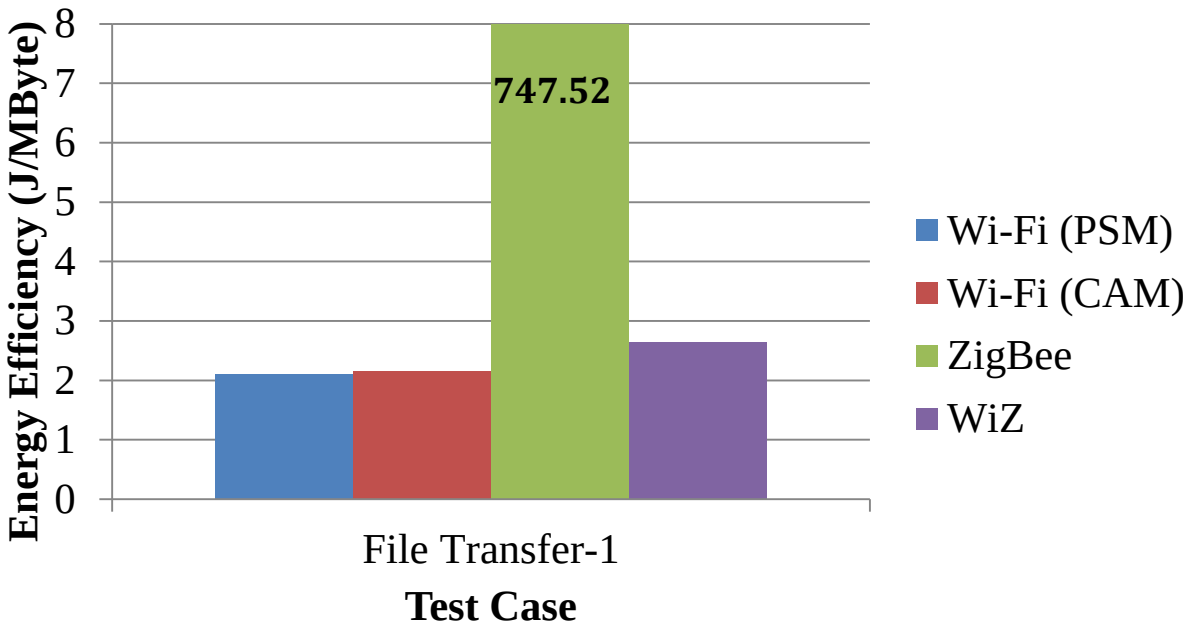


Figure 36: Energy efficiency of wireless networks with equal data rates. The data rates WiZ, Wi-Fi (CAM), and Wi-Fi (PSM) are all equal to 6.6Mbits/s in this graph to make the data rate appear constant so that the effect of transmission power can be more easily seen. ZigBee remains the same as the previous graph.

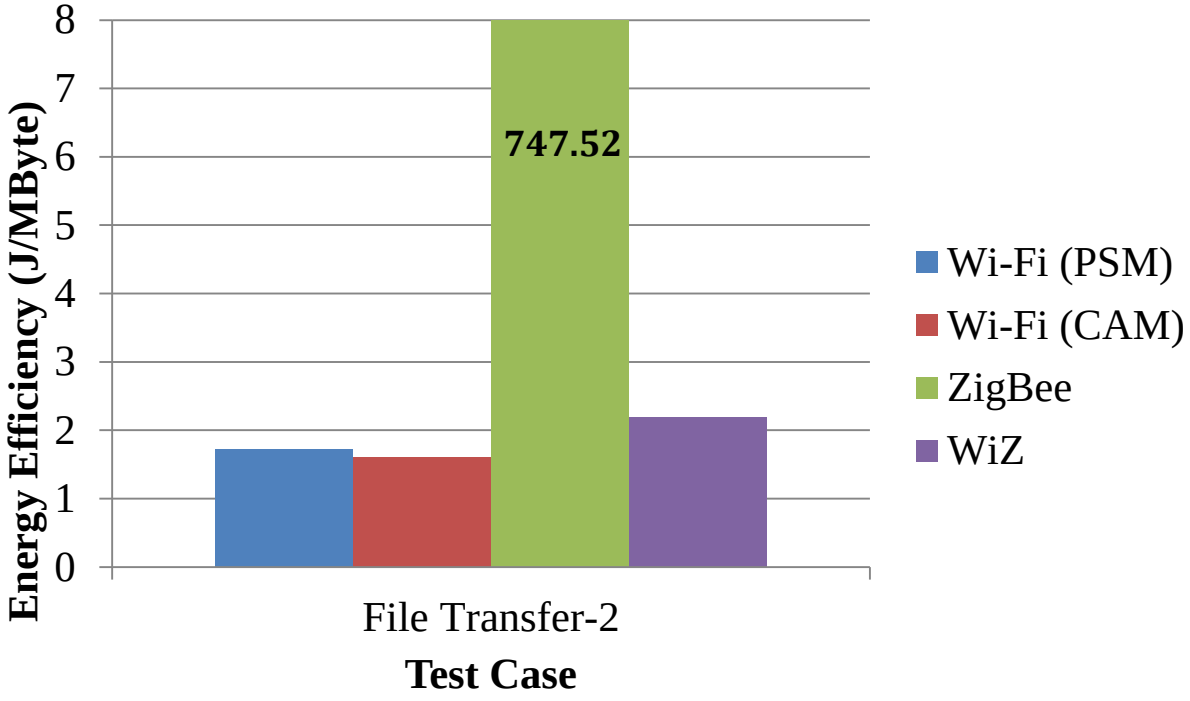


Figure 37: Energy efficiency of the wireless networks during the file transfer-2 test case.

The www.wikipedia.org test case consists of a small transfer of bulk data, then the rest of the time is spent idling. This is the type of scenario where WiZ excels. When the user travels to www.wikipedia.org they receive a small amount of data, in the WiZ network the Wi-Fi (CAM) is activated at this point and quickly downloads the data required to produce the webpage. When the downloading is done WiZ returns to idle, turns off Wi-Fi (CAM), and runs ZigBee alone. When the network is idle ZigBee consumes the least amount of energy, and so WiZ consumes the same. So, even though WiZ has a slightly higher power usage than PSM or CAM its idle power consumption is a fraction of PSM or CAM.

According to this chart ZigBee does quite well in terms of energy consumption, however, it takes ZigBee 204800 seconds (56.9 hours!) to download that small amount of data to produce the webpage. This does not possibly meet any Quality of Service requirements and is thus not a viable method to download a webpage.

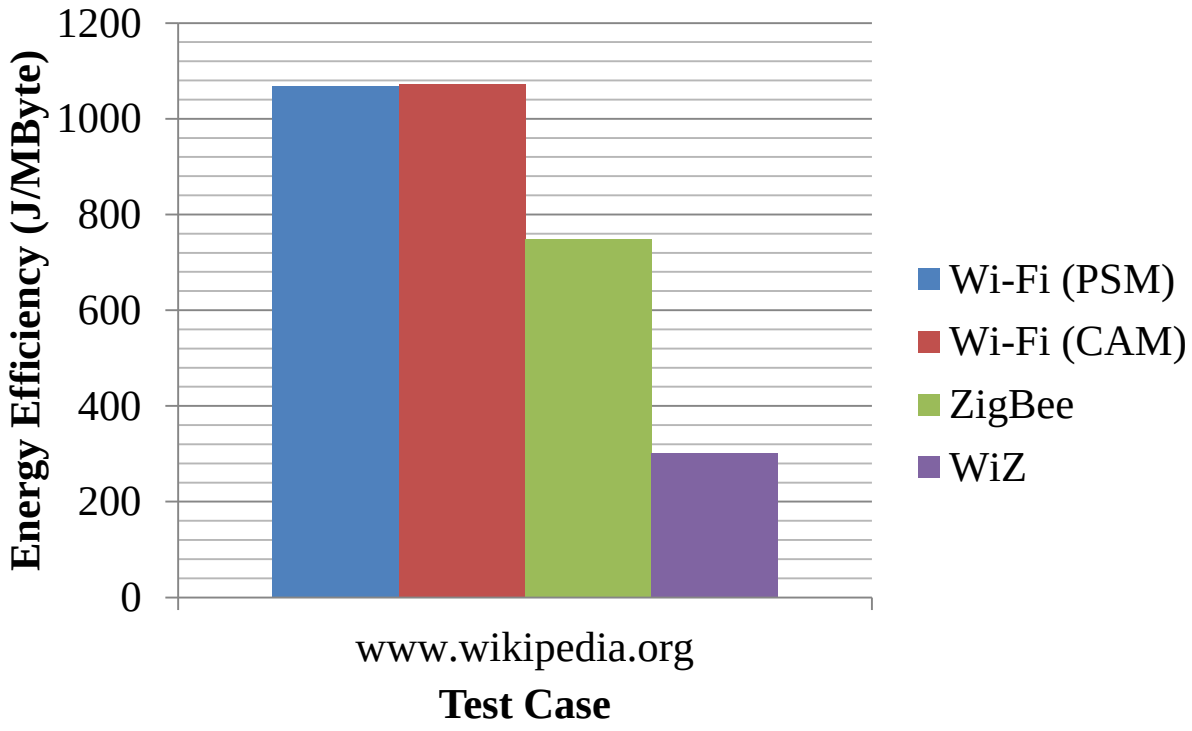


Figure 38: This is the chart for the www.wikipedia.org test case.

The energy efficiency is much higher, though less efficient, in Figure 38. Since the test transmitted so few megabytes much more joules were used while the system was idle and because it was idle for an extended period of time both the WiZ and ZigBee performed well. Wi-Fi (CAM and PSM) performs poorly in idle and thus consumed much more energy than the other networks. WiZ and ZigBee have the same idle properties. WiZ takes advantage of ZigBee's excellent idling capabilities allowing it

to achieve considerably lower power and energy consumption in idle than either Wi-Fi (CAM) or Wi-Fi (PSM). The idle test case does not have a energy efficiency chart because little to no data is transferred while in the idle state. Only the Wi-Fi (PSM) network would receive any network traffic in idle from the beacons sent by the AP. Table 21 summarizes these test cases below with a chart of the strengths and weaknesses of each network.

Table 21: A strengths and weaknesses chart of the networks. Strong represents that the network performs the task the best of the four networks, and is a viable means of executing the task. Moderate represents that the network performs the task fairly well compared to the other networks. It's neither a strength nor a weakness, and other networks may be better for suited for such a task. Weak means that the network performs the task poorly and the network should not be used for such a situation.

Network	Bulk Data Transfer	Intermittent Data Transfer	Idle
Wi-Fi (CAM)	Strong	Weak	Weak
Wi-Fi (PSM)	Strong	Weak	Weak
ZigBee	Weak	Weak	Strong
WiZ	Moderate	Strong	Strong

Conclusion of the Results

WiZ is a multiprotocol wireless network that uses Wi-Fi (CAM) and ZigBee in cooperation to reduce the energy consumption of mobile wireless devices. Although WiZ's transmitting parameters consume more power than that of conventional wireless protocols such as Wi-Fi (CAM) and Wi-Fi (PSM), the idle state of WiZ is vastly superior. By using ZigBee as its idle radio state and Wi-Fi (CAM) as its active state WiZ is able to have the equivalent data rate/bandwidth of conventional protocols and be more energy efficient. The plot in Figure 39 illustrates this the most effectively.

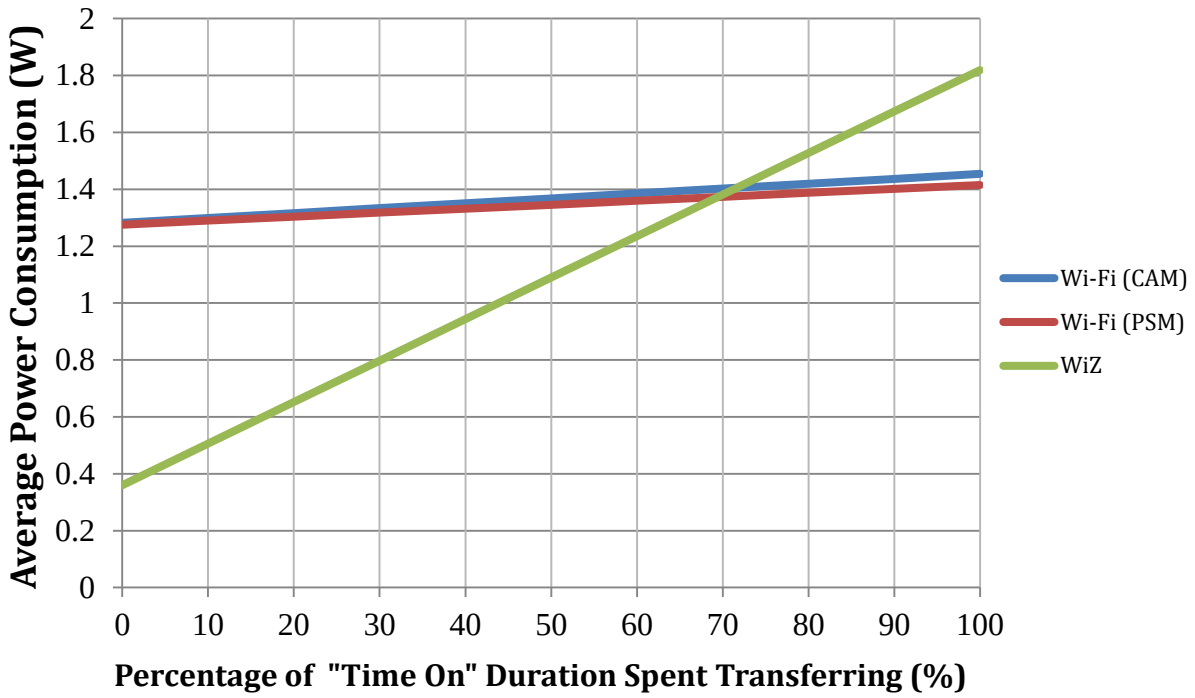


Figure 39: Power Usage as a Function of "Time On" Percentage. Plot of the percentage of time a mobile device spends transferring while on vs the average power consumed during that setting. If a device is on and transferring for less than 70% of the time it remains on then this graph claims that WiZ will save more energy than CAM or PSM.

This plot was created by an equation developed from the measured parameters of the four network protocols discussed in this report. The X value is the percent of time the network spends transferring data, and 100-X is the remaining time and in this remaining time the radio is idle. The Y value is the average power consumption for the situation described by the X-axis. For example at the X value of 30 the network is transmitting for 30% of time it is running, and it is idle for the remaining time, 70% of the time duration. To describe it in a more tangible manner let's say the radio begins in its "off" state. The radio is then turned "on" and immediately begins to transmit for 30 seconds. After the 30 seconds the radio changes its state to "idle" and stops transmitting. It continues transmitting for 70 seconds then the radio is turned "off". The Y value corresponding to the x value of 30 is .79 W for WiZ, 1.32 W for PSM, and 1.33 W for CAM. These values are the average power usages for that specific situation (30% Tx, 70% Idle).

The actual value of the time duration is irrelevant. What is important is the ratio between idling and transmitting. The use of a ratio is employed so that the plot can be applied to various wireless situations to determine how each wireless network will perform. For example, if a laptop is left on all day and the wireless remains idle until the owner turns it off at the end of day this plot claims that if the user

is using the WiZ network the average power consumption of the laptop's wireless radio will be .36 W throughout the day. If a the time between when the laptop is turned on in the morning and the time it is shut off at the end of the day is for example 15 hours, it possible to calculate that the wireless radio has consumed 324 Joules of energy that day. This calculation is shown below in Equation 3.

Equation 3: $$15\ hours * 60\ sec *.36\ Watts = 324\ Joules$$

Overall this plot states that if a wireless radio is transferring data for less than 70% of a duration of time the WiZ network will be more energy efficient and consume less energy (Joules) than Wi-Fi (CAM) or Wi-Fi (PSM). Since mobile wireless devices are idle most of the time the WiZ has the potential to save considerable amounts of energy.

Summary

With such focus in our world today being on mobility, greater efficencies in communications of these mobile device must be realized. For this project was a stepping stone in a new direction in efficencies of wireless technologies. The fact alone that off the shelf parts were used with such substantial results only cements the expectation for rapid growth of such a concept. Combined with the ubiqity of wireless devices and the evergrowing market place, multiprotocol systems show tremenous promise for the future. The applications for multiprotocol systems are far ranging. From large scale communication networks, to house hold appliances and networks. The greatest advantage is the simplicity and omnipotence of such a concept. The concept of this multiprotocol approach was proven by the achieved goals of this project.

The implementation described in this report has successfully achieved the following goals:

- Multi-protocol Network: The system was able to utilize dual radio protocols to transmit data synchronously
- Power Effency: The system was able to considerably reduce the communications power consummed compared with a single protocol network
- Performance Transparency: The multiprotocol network performed equalily or greater than the a single protocol network as interms of throughput
- Standard Equipment: This project only utlized off the shelf part in both nodes of the network

By demonstrating the practical feasibility of this network, a foundation is being established for research and exploration into multiprotocol systems. As standards like WiMax and ZigBee become more common, the practicality for these types of multi-protocol networks will grow. As this project was

completed with common equipment, it is only a matter of time before networks like the one described in this report are implemented. The technology already exists in the market place, it only needs to be recognized and combined.

Chapter 6: Conclusions and Future Work

The goal of this project was to develop a multi-protocol wireless network, that combines the energy efficiency of the ZigBee protocol IEEE 802.15.4 and the speed and high bandwidth of the Wi-Fi protocol IEEE 802.11 in order to improve the energy efficiency of current Wi-Fi only networks. The network also possesses cognitive radio attritubes that analyze and react to the surrounding radio environment. From the results of this project, it is clear that if ZigBee were to be used in place of Wi-Fi in idle mode, significant power savings will be seen. However, this implementation is far from being a completely functional multiprotocol "green" energy communication system for commercial use. Listed below are some improvements that could be made to this system in order to make it a product ready to by sold.

The first, and biggest, improvement to this system is integrating it with operating systems. Without integrating the software that controls WiZ into operating systems, programs would have to be modified to be able to take advantage of the multi-protocol network. By integrating the software of WiZ with operating systems, application programmers would not need to modify their programs at all. The network also lacks the ability to communicate with commercial software. In this project files are transferred using a programming software called Eclipse. However, no work has been done to stream video or make use of any other transfer of media aside from single files. Extensive work would need to be conducted to integrate the wireless network presented in this project with standard commericial software such as internet browsers, email software, and other computer programs that require internet access.

Another improvement to this system would be adding other wireless protocols. Bluetooth is an obvious choice, since it is currently installed in many laptops and cellular phones. Bluetooth could perform data transfer faster than ZigBee, and Bluetooth would also consume less power than Wi-Fi. Bluetooth could be the wireless radio used for data rates between Wi-Fi and ZigBee. Also, many research projects have proven that using a multiprotocol Wi-Fi and Bluetooth network can make wireless communications more energy efficient. Another protocol to consider is IMT-2000 (commonly known as "3G"). 3G is currently installed on many cellular phones and smartphones, as well as some laptops. Using 3G would greatly extend the wireless ubiquity of the network. The concept for this project was developed assuming that a mobile device would be the platform used. Moving the WiZ system to a

mobile device, such as a smart phone, would be a possible subject for future work. However, due to difficulties explained earlier in this report the project team had to abandon the mobile phone platform. By applying WiZ to a mobile phone the battery life could be greatly increased. An important consideration about 3G is that it is often not free to use. Most 3G customers have data plans and must either pay per megabit or pay based on a tiered data usage structure. Adding any extra wireless protocol will increase the versatility of the wireless device. The WiZ network developed in this project uses Wi-Fi (CAM) in conjunction with ZigBee. Currently, almost all modern Wi-Fi devices use Wi-Fi (PSM). A possible addition to the WiZ network would be to use Wi-Fi (PSM) with ZigBee. Wi-Fi (PSM) is more power efficient (thus the name Power Saving Mode) and would most certainly result in some power savings. By adding more protocols the cognitive engine will have the freedom to choose the best radio for the given circumstance.

One of the most appealing aspects of the ZigBee protocol is that it supports mesh networking. Mesh networking offers several advantages including a wider range, faster and more efficient routing of messages, and a more flexible network. A mesh network can also potentially withstand the unexpected disappearance of a router by using other nearby nodes to route traffic. Such a network could be implemented in future work to create a more robust network. Support for mesh networking was partially implemented in the program, but abandoned due to time constraints and because creating a routing network was not the focus of the project. The software controller was programmed with some support for mesh networks. It contains a network table with the name, 16-bit address, and 64-bit address of each XBee in the current network. At any point during the execution of the program, the XBee could send out a Node Discovery packet. Any radio that receives this packet sends back a reply containing its name, 16-bit, and 64-bit address. Therefore, any radio can update its network table with all of the radios within its range by sending out a Node Discovery packet, listening for the replies, and then adding any node's reply it gets to the network table.

Employing more common cognitive abilities is another option for future work. Learning from previous user behavior is one method employed in cognitive radios. Instead of just enabling and disabling Wi-Fi based on the size of the data queue, the system could also monitor parameters such as time of day, battery life, the location of the device, and what applications are being used. It would create different application profiles to ensure the necessary bandwidth for each application. For example, the system could learn that the user of the mobile device rarely turns on the radios at night time. With the ability to learn from the user the system could turn off the radios during the night to reduce battery usage, among other possibilities.

Bibliography

[1] (2008, September) 1st Internation Workshop on Green Wireless 2008 (W-GREEN). [Online]. http://www.cwc.oulu.fi/workshops/W-Green2008.pdf

[2] (1, January) Cellular-News. [Online]. http://www.cellular-news.com/story/36315.php?s=h

[3] Ian Mansfield. (2010, September) Cellular-News. [Online]. http://www.cellular-news.com/story/45599.php

[4] David Lagesse. (2009, April) US News: Money. [Online]. http://money.usnews.com/money/blogs/daves-download/2009/04/09/batteries-cant-keep-up-with-smartphones

[5] Joseph Mitola III and Gerald Q. Macguire, Jr., "Cognitive Radio: Making Software Radios More Peronsal," *IEEE Personal Communications*, pp. 13-18, August 1999.

[6] G. Anastasi, M. Conti, E. Gregori, and A. Passarella, "802.11 Power-Saving Mode for Mobile Computing in Wi-Fi hotspots: Limitations, Enhancements, and Open Issues," Dept. of Information Engineering, University of Pisa, Pisa, Italy, 2005.

[7] Daintree Networks, "What's so good about mesh networks?," Mountain View, 2007.

[8] Jeff Sharkey, "Coding For Life - Battery Life, That Is," in *Google IO*, San Francisco, 2009, pp. 1-24.

[9] Liviu Iftode, Cristian Borcea, Nishkam Ravi, Porlin Kang, and Peng Zhou, "Smart Phone: An Embedded System for Universal Interactions," in *10th IEEE International Workshop on Future Trends of Distributed Computing Systems*, Piscataway, NJ, 2004.

[10] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want, "CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces," Intel Research, UC San Diego, 2006.

[11] Tao Zheng and Sridhar Radhakrishnan, "A Switch Agent for Wireless Sensor Nodes with Dual Interfaces: Implementation and Evaluation," School of Computer Science, University of Oklahoma & Oklahoma State University, Norman & Stillwater, Oklahoma, USA, November 10, 2009.

[12] Simon Haykin, "Cogntive Radio: Brain-Empowered Wireless Communications," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201-220, February 2005.

[13] Ian F. Akyildiz, Won-Yeol Lee, Mehmet C. Vuran, and Shantidev Mohanty. (2008, April) A Survey on Spectrum Management in Cognitive Radio Networks.

[14] Timothy R. Newman, "Multiple Objective Fitness Functions for Cognitive Radio Adaptation,"

University of Kansas, PhD Thesis 2008.

[15] Charles Clancy, Erich Stuntebeck, Joe Hecker, and Tim O'Shea, "Applications of Machine Learning to Cognitive Radio Networks," *IEEE Wireless Communications*, pp. 47-52, August 2007.

[16] Joseph Mitola and Gerald MaGuire. (1999) IEEE. [Online]. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00788210

[17] Thomas W. Rondeau, Bin Le, Christian Rieser, and Charles W. Bostian, "Cognitive Radios with Genetic Algorithms: Intelligent Control of Software Defined Radios," in *SDR 04 Techincal Conference and Product Exposition*, 2004, p. 6.

[18] Mark Norton. (2005, November) Spectrum and Its Influence on 3G and Wi-Fi Architectures. Document.

[19] (2011) ETSI World Class Standards. [Online]. http://portal.etsi.org/portal/server.pt/community/BRAN/299

[20] Helping Define 802.11n and other Wireless LAN Standards. [Online]. http://www.intel.com/standards/case/case_802_11.htm

[21] Ryan Kim. (2011) GigaOM. [Online]. http://gigaom.com/2010/12/30/wi-fi-hotspots-only-going-to-get-hotter/

[22] (2011) Skyhook, Inc. [Online]. http://www.skyhookwireless.com/howitworks/coverage.php

[23] The Economist Print Edition, "A Brief History of Wi-Fi," 2004.

[24] Wi-Fi Alliance. (2009, September) Wi-Fi Alliance. [Online]. http://www.wi-fi.org/register.php?file=wp_Wi-Fi_CERTIFIED_n_Industry.pdf

[25] Stephen McCann and Alex Ashley. (2011, February) Official IEEE 802.11 working group project timelines. [Online]. http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm

[26] Alexander Wyglinksi. (2011, January) Wireless Innovation Laboratory. [Online]. http://www.wireless.wpi.edu/courses/ece4305c11/lectures/ece4305_L02.pdf

[27] Raja Jurdak, Antonio G. Ruzzelliz, and G.M.P. O'Harez. (2009, September) Texas Instruments. [Online]. http://focus.ti.com/docs/prod/folders/print/cc2420.html

[28] Prof. Godred Fairhurst. (2008, November) University of Aberdeen. [Online]. http://www.erg.abdn.ac.uk/users/gorry/course/road-map.html

[29] Patrick Kinney, "ZigBee Technology: Wireless Control that Simply Works," Lake Zurich, 2003.

[30] Gary Legg. (2004) EETimes. [Online]. http://www.eetimes.com/design/communications-design/4017853/ZigBee-Wireless-Technology-for-Low-Power-Sensor-Networks

[31] William C. Craig. (2006, January) InTech. [Online]. http://www.isa.org/InTechTemplate.cfm?Section=Control_Fundamentals1&template=/ContentManagement/ContentDisplay.cfm&ContentID=51164

[32] Digi. (2010, July) Waspmote ZigBee Networking Guide.

[33] Jack Shandle. (2008, February) EETimes. [Online]. http://www.eetimes.com/electronics-news/4182200/ZigBee-Alliance-charters-new-group-to-explore-Internet-solutions

[34] Texas Instruments. (2009, April) Texas Instruments. [Online]. http://focus.ti.com/pr/docs/preldetail.tsp?sectionId=594&prelId=sc09054

[35] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. (2001) A Survey of Energy Efficient Network Protocols forWireless.

[36] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen, "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi," in *The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON)*, Taipei, Taiwan, 2007, pp. 46-51.

[37] Paolo Baronti et al., "Wireless senor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Computer Communications*, vol. 30, no. 7, pp. 1655-1695, May 2007.

[38] Erina Ferro and Francesco Potorti, "Bluetooth and Wi-Fi wireless protocols: a survey and a comparison," *Wireless Communications, IEEE*, vol. 12, no. 1, pp. 12-16, February 2005.

[39] Jason Flinn and M. Satyanarayan, "Managing Battery Lifetime with Energy-Aware Adaptation," *ACM Transactions on Computer Systems*, vol. 22, no. 2, May 2004.

[40] Late Droid. (2011) http://latedroid.com/juicedefender.

[41] Spirent Communications, Test Methodology Journal: IMIX (Internet Mix) Journal, March 2006.

[42] R. Levering and M. Cutler, "The Portrait of a Common HTML Web Page," Binghamton University SUNY, 2006.

[43] Digi International, Inc. (2007, June) XBee Series 2 OEM RF Modules Product Manual.

[44] Honggang Zhang. (2009, January) Zhejiang University. [Online]. http://sites.google.com/site/honggangzhanglabs/Home/green-communications--green-networking-and-green-spectrum

[45] Michael Ghizzoni, Mathew Kelley, and Conor Rochford, "Cognitive Radio Using Radio Resource Management," Worcester, Major Qualifying Project 2009.

[46] Fernando Company Serra, Javier González López, David Baqués Ibañez, and Javier López Rubio, "Design and Implementation of a Cognitive Node for Heterogeneous Wireless Ad-Hoc Network,"

Limerick, Bachelor of Engineering Thesis 2010.

[47] Gadi Shor, "How Bluetooth, UWB, and 802.11 stack up on power consumption," *EE Times Design*, p. 4, April 2008.

[48] Daniel Indiviglio. (2009, December) The Atlantic. [Online]. http://www.theatlantic.com/business/archive/2009/12/at-t-to-discourage-mobile-data-usage/31538/

[49] Raymond J. Lackey and Donald W. Upmal, "Speakeasy: The Military Software Radio," *IEEE Communications Magazine*, pp. 56-61, May 1995.

[50] Walter HW Tuttlebee, "Advances in Software Defined Radio," *Ann. Telecommun.*, pp. 314-337, 2002.

[51] Matthew N. O. Sadiku and Cajetan M. Akujuobi. (2004, October/November) Software Defined Radio: A Brief Overview.

[52] J Mitola, "The Software Radio," *IEEE National Telesystems Conference*, 1992.

[53] Federal Communications Commission, "Authorization of Spread Spectrum Systems Under Parts 15 and 90 of the FCC Rules," 1985.

[54] Sudhir B Pendse, US4707829, 1987.

[55] (2000, May) Intersil. [Online]. http://www.qsl.net/n9zia/pdf/AN9820.pdf

[56] Louis Litwin. (2001) IEEE Potentials. [Online]. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00969594

[57] John Hoadley. (2005, September) Building future networks with MIMO and OFDM. [Online]. http://connectedplanetonline.com/wireless/technology/mimo_ofdm_091905/

[58] IEEE Computer Society. (2007, June) Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.

[59] R. Venkatesha Prasad, Przemyslaw Pawelczak, H. Steven Berger, and James A. Hoffmeyer, "Cognitive Functionality in Next Generation Wireless Networks: Standardization Efforts," *IEEE Communications Magazine*, pp. 72-78, April 2008.

[60] Jose Gutierrez et al., "IEEE 802.15.4: A Developing Standard for Low-Power Low-Cost Wireless Personal Area Networks," *IEEE Network*, vol. 15, no. 5, pp. 12-19, 2001.

[61] (2010, October) UnbeatableSale.ocom. [Online]. http://site.unbeatablesale.com/img055/dhwusb100.jpg

[62] IEEE. (2002) IEEExplore. [Online]. http://ieeexplore.ieee.org/servlet/opac?punumber=7932

[63] Hoovers. Hoovers. [Online]. http://www.hoovers.com/business-information/--pageid__13751--

/global-hoov-index.xhtml

[64] Patrice Oehen. ZigBee: An Overview of the Upcoming Standard.

[65] Tao Zheng and Sridhar Radhakrishnan. (2009, November) A Switch Agent for Wireless Sensor Nodes with Dual Interfaces: Implementation and Evaluation.

[66] (2004, June) Marcus Spectrum Solutions LLC. [Online]. http://www.marcus-spectrum.com/documents/economist.pdf

[67] Stephen McCann and Alex Ashley. (2011) OFFICIAL IEEE 802.11 WORKING GROUP PROJECT TIMELINES - 2011-02-02. [Online]. http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm

Appendix A:

FileBytes.java

```
package lpcn.xbee;

import java.io.Serializable;

public class FileBytes implements Serializable
{
	public byte[] bytes;

	public FileBytes()
	{
	}

	public FileBytes(long byteSize)
	{
		this.bytes = new byte[(int)byteSize];
	}

	public FileBytes(byte[] bytes)
	{
		this.bytes = bytes;
	}

	public int[] getBytesAsIntArray()
	{
		int[] ints = new int[bytes.length];
		int i = 0;
		for(int oneInt : bytes)
		{
			ints[i] = oneInt;
			i++;
		}
		return ints;
	}
}
```

FileInfo.java

```
package lpcn.xbee;

import com.rapplogic.xbee.api.XBeeAddress16;
import com.rapplogic.xbee.api.XBeeAddress64;

public class FileInfo {
	String fileName;
	long fileSize;
```

```
    long bytesTouched;
    boolean toSend;

    public FileInfo(String fileName, long fileSize, boolean toSend)
    {
        this.fileName = fileName;
        this.fileSize = fileSize;
        this.bytesTouched = 0;
        this.toSend = toSend;
    }

    public FileInfo(String fileName, long fileSize, long bytesTouched, boolean toSend)
    {
        this.fileName = fileName;
        this.fileSize = fileSize;
        this.bytesTouched = bytesTouched;
        this.toSend = toSend;
    }

    public String getFileName() {
        return fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public long getFileSize() {
        return fileSize;
    }

    public void setFileSize(long fileSize) {
        this.fileSize = fileSize;
    }

    public long getbytesTouched() {
        return bytesTouched;
    }

    public void setbytesTouched(long bytesTouched) {
        this.bytesTouched = bytesTouched;
    }

    public boolean isToSend() {
        return toSend;
    }

    public void setToSend(boolean toSend) {
        this.toSend = toSend;
    }
}
```

GeneralGUI.java

```
package lpcn.xbee;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketException;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.LinkedBlockingQueue;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBeeException;
import com.rapplogic.xbee.api.zigbee.ZNetTxStatusResponse;

public class GeneralGUI extends JFrame implements ActionListener
{
	/* CONSTANTS */
	final String FILEPATH = "/home/pdesantis/Documents/testing/";   // Holds the default download
directory for this program

	/* THREADS */
	QueueManager tQueueManager = new QueueManager();
	WiFiDataListener tWiFiDataListener = new WiFiDataListener();

	/* CLASS VARIABLES */
	public XBeeInterface xbee;
	public PacketListener packetListener;

	Socket clientSocket;
	ObjectOutputStream clientObjectOutputStream;
```

```
ObjectInputStream clientObjectInputStream;

public volatile boolean connectedWiFi;
public volatile boolean transmissionInProgress;

long scriptStartTime = 0;

int port = 13267;

volatile LinkedBlockingQueue<int[]> queueDataReceived = new LinkedBlockingQueue<int[]>();
volatile LinkedBlockingQueue<FileInfo> queueFiles = new LinkedBlockingQueue<FileInfo>();
public long queueBytes;
public File file;
volatile boolean fileTransferComplete;

public FileInputStream fileInStream;
public BufferedInputStream buffInStream;

public FileOutputStream fileOutStream;
public BufferedOutputStream buffOutStream;

/* GUI VARIABLES */
JPanel pane;
JTextField devicePort = new JTextField("/dev/ttyUSB0", 15);
JLabel devicePortLabel = new JLabel("Enter Device Port:");
JButton startButton = new JButton("Enable SmartPower");
JButton sendPacketButton = new JButton("Send Packet");
JButton sendFileButton = new JButton("Send a File");
JTextField fileLocation = new JTextField(FILEPATH + "file1", 40);
JLabel fileLocationLabel = new JLabel("Enter File Path:");

boolean readyToSend;

boolean selfReadyToReceive;
boolean friendReadyToReceive;

JTextArea outputBox = new JTextArea();;

/* GUI METHODS */
public GeneralGUI()
{
	//FILEPATH = "/home/pdesantis/Documents/testing/";

	connectedWiFi = false;
	transmissionInProgress = false;
	readyToSend = true;
	selfReadyToReceive = false;
	friendReadyToReceive = false;

	setSize(600, 600);
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        /*
        MessageConsole mc = new MessageConsole(outputBox);
        mc.redirectOut(null, System.out);
        mc.setMessageLines(25);
         */
        startButton.addActionListener(this);
        sendPacketButton.addActionListener(this);
        sendFileButton.addActionListener(this);
        pane = new JPanel();

        pane.add(devicePortLabel);
        pane.add(devicePort);

        pane.add(startButton);
        pane.add(sendPacketButton);
        pane.add(sendFileButton);
        pane.add(outputBox);

        pane.add(fileLocation);
        pane.add(fileLocationLabel);

}

// This is basically an "abstract class" here
// It gets redefined in our subclasses
public void actionPerformed(ActionEvent event)
{
}

/* CLASS METHODS */
public void openXBeeInterface(String XBeePort, int XBeeBaud)
{
        XBeePort = "/dev/ttyUSB0";
        XBeeBaud = 9600;
        clientSocket = new Socket();
        try
        {
                if(xbee == null)
                {
                        xbee = new XBeeInterface(XBeePort, XBeeBaud);
                }
                else
                {
                        //xbee.xbee.close();
                        //xbee.xbee.open(XBeePort, XBeeBaud);
                }
                tQueueManager.start();  // Start the Queue Manager Thread
        }
        catch (XBeeException e)
        {
```

```
        }
    }

    /* UTILITY FUNCTIONS */
    public int[] removeFirstArrayElement(int[] arr)
    {
        int[] new_arr;
        new_arr = new int[arr.length - 1];
        for(int i = 1; i < arr.length; i++)
        {
            new_arr[i - 1] = arr[i];
        }
        return new_arr;
    }

    public String removeFirstChar(String str)
    {
        String new_str = str.substring(1);
        return new_str;
    }
    /* END UTILITY FUNCTIONS */


    /* SEND/RECEIVE FUNCTIONS */
    public void sendFileTransferRequest(FileInfo fileInfo)
    {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String fileName;
        char[] fileSizeChar;

        // TODO add file selection box thingy

        fileName = fileInfo.getFileName();          // Read the name of the file
        long fileSize = fileInfo.getFileSize(); // Read the size of the file in bytes
        fileSizeChar = Long.toString(fileSize).toCharArray();   // Convert the file size (long) to a string

        // CREATE THE ZIGBEE PACKET
        // Create the payload integer array with the array size of 2 (for the status byte and separator), the
length of the file name, and the length of the file size
        // TODO add a check to see if the payload is still less than the max payload size
        int[] payload = new int[2 + fileName.length() + fileSizeChar.length];

        // Set the status byte to 3
        payload[0] = '3';

        // Break apart the fileName string into chars and add them to the payload
        int i = 1;
```

```
            for(char oneChar : fileName.toCharArray())
            {
                payload[i++] = oneChar;
            }

            payload[i++] = ',';          // Add a comma to separate the file name from the file size

            // Break apart the fileSizeChar array and add it to the payload
            for(char oneChar : fileSizeChar)
            {
                payload[i++] = oneChar;
            }

            //System.out.println("Sending file transfer request for file: " + fileName);
            readyToSend = false;
            // Send the packet, check whether it was delivered successfully or not, and if not, repeat sending
it until it succeeds
            while(xbee.XBeeSendPacket(payload) != ZNetTxStatusResponse.DeliveryStatus.SUCCESS);
            //xbee.XBeeSendPacket(payload);
            //xbee.XBeeSendPacket(payload); // Send the packet and wait for an ACK
            readyToSend = true;
            //System.out.println("File transfer request ACK received for file: " + fileName);

            //System.out.println("Adding file to queue to be sent: " + fileName);
            // Add this file to the queue of files to be sent
            queueFiles.add(fileInfo);
            queueBytes = queueBytes + fileInfo.getFileSize();
        }

        public void sendFile(FileInfo fileInfo)
        {
            transmissionInProgress = true;

            ArrayList<Integer> payloadTemp;
            int currentAmtBytesRead;
            int oneByte;
            long start;
            long end;
            int packetCounter = 0;

            boolean firstLoopZigBee = true;
            boolean firstLoopWiFi = true;

            try
            {
                FileInputStream fileInStream = new FileInputStream(FILEPATH +
fileInfo.getFileName());
                BufferedInputStream buffInStream = new BufferedInputStream(fileInStream);

                //System.out.println("Waiting for friend to be ready to receive");
                int spacer = 0;
```

```
                //System.out.println("");
                while(!friendReadyToReceive)
                {
                        /*
                        System.out.print(".");
                        spacer++;
                        if(spacer == 100)
                        {
                                spacer = 0;
                                System.out.println("");
                        }
                         */
                        // sleep, so as not to kill CPU cycles
                        // is this even necessary?
                        try
                        {
                                //System.out.print(".");
                                Thread.sleep(10);
                        }
                        catch (InterruptedException e)
                        {
                                e.printStackTrace();
                        }
                }
                //System.out.println("");
                //System.out.println("Friend is ready to receive!");
                /*
                if(fileInfo.getbytesTouched() == fileInfo.getFileSize())
                {
                }
                */

                start = System.currentTimeMillis();
                System.out.println("Sending started " + (start - scriptStartTime) + " milliseconds after the
start of the script");

                while(fileInfo.getbytesTouched() != fileInfo.getFileSize())
                {
                        //System.out.println("trying to send? readyToSend is: " + readyToSend);
                        currentAmtBytesRead = 0;        // reset the current amount of bytes read for this loop
to 0
                        if(connectedWiFi)       // Wi-Fi
                        {
                                if(firstLoopWiFi)
                                {
                                        System.out.println("Sending the file through Wi-Fi: " +
fileInfo.getFileName());
                                        firstLoopWiFi = false;
                                }

                                long bytesLeft = fileInfo.getFileSize() - fileInfo.getbytesTouched();
```

```
                    FileBytes fileBytes = new FileBytes(bytesLeft);

                    buffInStream.read(fileBytes.bytes);

                    clientObjectOutputStream.writeObject(fileBytes);
                    clientObjectOutputStream.flush();
                    clientObjectOutputStream.reset();
                    fileInfo.setbytesTouched(fileInfo.getbytesTouched() + fileBytes.bytes.length);
                    currentAmtBytesRead = fileBytes.bytes.length;
                }
                else    // ZigBee
                {
                    if(readyToSend)
                    {
                        if(firstLoopZigBee)
                        {
                            System.out.println("Sending the file through ZigBee: " +
fileInfo.getFileName());
                            firstLoopZigBee = false;
                        }

                        payloadTemp = new ArrayList<Integer>(); // Reset the temporary payload
list
                        payloadTemp.add(new Integer(52));             // Set the first (status) byte to
52 (the decimal code for '4'), the number for FILE DATA
                        while(currentAmtBytesRead < 70)               // Read (up to) 70 bytes from
the file
                        {
                            oneByte = buffInStream.read();
                            if(oneByte == -1)       // If there are no more bytes left in the file (there
were less than 70 bytes left to read)
                            {
                                break;  // Done reading, break out of this loop!
                            }
                            else
                            {
                                payloadTemp.add(new Integer(oneByte));  // Add these bytes to
the temporary payload array
                                currentAmtBytesRead++;                // Increment the amount of
bytes read
                            }
                        }

                        // Convert the arraylist to an integer array
                        int[] payload = new int[payloadTemp.size()];
                        for(int i = 0; i < payload.length; i++)
                        {
                            payload[i]=((Integer)payloadTemp.get(i)).intValue();
                        }
```

```
                                //System.out.println("Sending packet with " + currentAmtBytesRead + "
bytes of data");
                                System.out.print(".");

                                packetCounter++;
                                if(packetCounter == 49)
                                {
                                        packetCounter = 0;
                                        System.out.println("");
                                }

                                fileInfo.setbytesTouched(fileInfo.getbytesTouched() +
currentAmtBytesRead);
                                // Send the packet, check whether it was delivered successfully or not, and
if not, repeat sending it until it succeeds
                                //while(xbee.XBeeSendPacket(payload) !=
ZNetTxStatusResponse.DeliveryStatus.SUCCESS);
                                xbee.XBeeSendPacket(payload);
                        }
                        else
                        {
                                try
                                {
                                        Thread.sleep(10);
                                }
                                catch (InterruptedException e)
                                {
                                        e.printStackTrace();
                                }
                        }
                    }
                    queueBytes = queueBytes - currentAmtBytesRead;          // Decrease queueBytes by
the amount of bytes just sent
                }
                friendReadyToReceive = false;
                //System.out.println("Sent file: " + fileInfo.fileName + " , bytes: " +
fileBytes.bytes.length);
                end = System.currentTimeMillis();
                //System.out.println("");
                long bandwidth = 0;
                if((end-start) != 0)
                        bandwidth = ((fileInfo.getFileSize() / (end-start)) * 1000);
                System.out.println("Sent file. File: "  + fileInfo.getFileName() + " , Size: " +
fileInfo.getFileSize() + " bytes, time: " + (end-start) + " milliseconds, bandwidth: " + bandwidth + "
bytes/second");
                System.out.println("Sending completed " + (end - scriptStartTime) + " milliseconds after
the start of the script" + " (" + new Date().toString() + ")");
                System.out.println("queueBytes = " + queueBytes);
                System.out.println("");
                buffInStream.close();   // Close the buffered input stream
```

```
                fileInStream.close();   // Close the file input stream
            }
            catch (FileNotFoundException e)
            {
                e.printStackTrace();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }

        public void receiveFile(FileInfo file)
        {
            int[] bytesToWrite;
            int bytesWritten = 0;
            long start;
            long end;
            int packetCounter = 0;

            try
            {
                fileOutStream = new FileOutputStream(FILEPATH + file.getFileName());    // set up the
Output Streams
                buffOutStream = new BufferedOutputStream(fileOutStream);

                System.out.println("Receiving file. File: "  + file.getFileName());
                //System.out.println("Output Streams set up");

                start = System.currentTimeMillis();
                System.out.println("Receiving started " + (start - scriptStartTime) + " milliseconds after
the start of the script");
                //System.out.println("Sending ready to receive packet!");
                selfReadyToReceive = true;

                int[] payload = new int[] { '5' };
                while(xbee.XBeeSendPacket(payload) !=
ZNetTxStatusResponse.DeliveryStatus.SUCCESS);     // Send the packet, check whether it was delivered
successfully or not, and if not, repeat sending it until it succeeds

                while(file.getFileSize() - file.getbytesTouched() > 0)  // While the file has not been
completely received
                {
                    bytesToWrite = queueDataReceived.take();        // Wait until data has been received,
then take it off the queue...
                    //System.out.println("Took data off the data received queue: " +
bytesToWrite.length);
                    for(int oneByte : bytesToWrite)
                    {
                        buffOutStream.write(oneByte);                // ...and write it to the output stream
                        //System.out.println("Writing one byte: " + oneByte);
```

```
                        }
                        //System.out.println("Wrote " + bytesToWrite.length + " bytes");
                        //System.out.print(".");
                        /*
                        packetCounter++;
                        if(packetCounter == 50)
                        {
                              packetCounter = 0;
                              //System.out.println("");
                        }
                        */
                        file.setbytesTouched(file.getbytesTouched() + bytesToWrite.length);     // Update the
amount of bytes written
                        bytesWritten = bytesWritten + bytesToWrite.length;
                        queueBytes = queueBytes - bytesToWrite.length;  // Subtract the number of bytes
written from the queue size
                  }
                  end = System.currentTimeMillis();
                  //System.out.println("");
                  long bandwidth = 0;
                  if((end-start) != 0)
                        bandwidth = ((file.getFileSize() / (end-start)) * 1000);
                  System.out.println("File received. File: "  + file.getFileName() + " , Size: " +
file.getFileSize() + " bytes, time: " + (end-start) + " milliseconds, bandwidth: " + bandwidth + "
bytes/second");
                  System.out.println("Sending completed " + (end - scriptStartTime) + " milliseconds after
the start of the script" + " (" + new Date().toString() + ")");
                  System.out.println("queueBytes = " + queueBytes);
                  System.out.println("");
                  selfReadyToReceive = false;
                  buffOutStream.flush();
                  buffOutStream.close();
                  fileOutStream.close();
            }
            catch (FileNotFoundException e)
            {
                  e.printStackTrace();
            }
            catch (IOException e)
            {
                  e.printStackTrace();
            }
            catch (InterruptedException e)
            {
                  e.printStackTrace();
            }
      }
      /* END SEND/RECEIVE FUNCTIONS */


      /* THREAD CLASSES */
```

```
// Takes files off the queue and sends them to the appropriate handler function
class QueueManager implements Runnable
{
	private volatile Thread threadQueueManager;
	FileInfo currentFile;

	void stop()
	{
		threadQueueManager = null;
	}

	void start()
	{
		threadQueueManager = new Thread(this);
		threadQueueManager.start();
	}

	public void run()
	{
		Thread thisThread = Thread.currentThread();
		//System.out.println("Started the queue manager thread");
		while(threadQueueManager == thisThread)
		{
			try
			{
				//System.out.println("Waiting to take a file off the queue");
				currentFile = queueFiles.take();
				if(currentFile.isToSend())
				{
					//System.out.println("File taken off the queue to be sent: " +
currentFile.getFileName());
					sendFile(currentFile);		// wait until file is sent
					//System.out.println("File sent!");
				}
				else
				{
					//System.out.println("File taken off the queue to be received: " +
currentFile.getFileName());
					receiveFile(currentFile);		// wait until file is received
					//System.out.println("File received!");
				}
			}
			catch (InterruptedException e)
			{
				e.printStackTrace();
			}
		}
	}
}

// Takes packet from received from socket and adds it to the data queue
```

```
class WiFiDataListener implements Runnable
{
	private volatile Thread threadWiFiDataListener;

	void stop()
	{
		threadWiFiDataListener = null;
	}

	void start()
	{
		threadWiFiDataListener = new Thread(this);
		threadWiFiDataListener.start();
	}

	public void run()
	{
		Thread thisThread = Thread.currentThread();
		try
		{
			while(threadWiFiDataListener == thisThread)
			{
				if(connectedWiFi)
				{
					try
					{
						//if(clientObjectInputStream.available() > 0)
						//{
							//DataInputStream testy;
							//testy.
							FileBytes fileBytes = (FileBytes)
clientObjectInputStream.readObject();
							//System.out.println("WiFi Listener received: " + inByte);
							queueDataReceived.add(fileBytes.getBytesAsIntArray());
						//}
							/*
						else
						{
							try
							{
								Thread.sleep(10);
							}
							catch (InterruptedException e)
							{
							}
						}
						*/
					}
					catch (ClassNotFoundException e)
					{
						e.printStackTrace();
```

```
                                }
                                catch(SocketException e)
                                {
                                        //e.printStackTrace();                                  }
                                }
                        }
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
        }
    }
}
```

SmartPowerAP.java

```
package lpcn.xbee;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;

import javax.swing.JButton;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import com.rapplogic.xbee.api.ApiId;
import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBeeAddress64;
import com.rapplogic.xbee.api.XBeeResponse;
import com.rapplogic.xbee.api.zigbee.ZNetRxResponse;
import com.rapplogic.xbee.api.zigbee.ZNetTxStatusResponse;
import com.rapplogic.xbee.util.ByteUtils;

public class SmartPowerAP
{
    public final static Logger log = Logger.getLogger(SmartPowerAP.class);

    /**
     * @param args
     */
    public static void main(String[] arguments)
    {
        PropertyConfigurator.configure("log4j.properties");
```

```
            APGUI bf = new APGUI();
        }
}

class APGUI extends GeneralGUI implements ActionListener
{
        // CLASS VARIABLES
        ServerSocket serverSocket;

        // THREADS
        AddFilesToQueue tAddFilesToQueue = new AddFilesToQueue();
        WiFiConnectionListener tWiFiConnectionListener = new WiFiConnectionListener();
        SendWiFiInfo tSendWiFiInfo = new SendWiFiInfo();

        // GUI VARIABLES
        JButton runAddFileScriptButton = new JButton("Run Add Files Script");

        // Constructor
        public APGUI()
        {
                setTitle("AP in the Hizzouce");

                runAddFileScriptButton.addActionListener(this);

                pane.add(runAddFileScriptButton);

                add(pane);
                setVisible(true);

                try
                {
                        serverSocket = new ServerSocket(port);
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }

                packetListener = new PacketListener()
                {
                        public void processResponse(XBeeResponse response)
                        {
                                //System.out.println("Received packet: " + response);
                                if (response.getApiId() == ApiId.ZNET_RX_RESPONSE)
                                {
                                        int[] payload;          // Stores the payload of the received packet
                                        ZNetRxResponse ZNetResponse = new ZNetRxResponse();             // Stores the
ZNetRxResonse type of the received packet

                                        ZNetResponse = (ZNetRxResponse) response;       // Cast received packet to the
correct type (ZNetRxResonse)
```

```
                        payload = ZNetResponse.getData();                   // Store the data from the
packet into the payload array

                        switch(payload[0])      // Look at the status byte (first byte) of the payload
                        {
                        case '0':       // WIFI_REQUEST : Requests a Wi-Fi access point to connect to
                              System.out.println("Wi-Fi request received");
                              tWiFiConnectionListener.start();
                              tSendWiFiInfo.start();
                              break;

                        case '1':       // WIFI_INFO : Contains Wi-Fi access point information
                              // not needed on the access point
                              break;

                        case '2':       // WIFI_STOP : Informs Wi-Fi AP that the connection is closing
                              try
                              {
                                    //clientObjectOutputStream.close();
                                    connectedWiFi = false;
                                    tWiFiDataListener.stop();
                                    clientSocket.close();
                              }
                              catch (IOException e)
                              {
                                    e.printStackTrace();
                              }
                              break;

                        case '3':       // FILE_TRANSFER_REQUEST : Contains file information
                              String fileInfoString = removeFirstChar(ByteUtils.toString(payload));   //
Strip the status (first) byte from the payload
                              String[] fileInfo = fileInfoString.split(",");          // Split the data into
separate strings for file name and size
                              String fileName = fileInfo[0];                                // Store the file
name
                              long fileSize = Long.parseLong(fileInfo[1]);            // Convert the file size
string into a long and store it
                              //System.out.println("File Transfer Request received: " + fileName + " of
size " + fileSize + " bytes");
                              queueFiles.add(new FileInfo(fileName, fileSize, false));        // Add the file
to the write queue
                              queueBytes = queueBytes + fileSize;
                              //receiveFileTransferRequest(fileName, fileSize);               // Handle the file
transfer request
                              break;

                        case '4':       // FILE_DATA : Contains 70 bytes of file data
                              int[] dataInt = removeFirstArrayElement(payload);       // Strip the status
(first) byte from the payload
```

```
                                queueDataReceived.add(dataInt);         // Add the byte array to the received
data queue
                                break;

                        case '5':       // INCOMING_FRIEND_TO_RECEIVE : Contains file information
                                friendReadyToReceive = true;
                                //System.out.println("received a ready to receive packet from friend");
                                break;

                        default:
                                //System.out.println("Received random ass packet: " + response);
                                //System.out.println("Packet length is this shit: " +
ZNetResponse.getLength());
                                //System.out.println("Payload length is fuckin: " +
ZNetResponse.getData().length);
                                break;
                        }
                    }
                }
            };
        }

        // Event handler
        public void actionPerformed(ActionEvent event)
        {
            Object source = event.getSource();
            if(source == startButton)
            {
                openXBeeInterface(devicePort.getText(), 9600);
                xbee.setDestinationAddress64(new XBeeAddress64(0, 0x13, 0xa2, 0, 0x40, 0x30, 0xc1,
0x3e));
                while(!xbee.xbee.isConnected())
                {
                    // Do nothing until the xbee has been connected to
                }
                xbee.xbee.addPacketListener(packetListener);
            }
            else if(source == sendPacketButton)
            {
                sendWiFiInfo();

                //int[] payload = new int[] { '6' };

            }
            else if(source == sendFileButton)
            {
                File fileToSend = new File(fileLocation.getText());             // Look up the file on the hard
drive
                FileInfo fileInfo = new FileInfo(fileToSend.getName(), fileToSend.length(), true);      //
Create a FileInfo object
                sendFileTransferRequest(fileInfo);              // Send a file transfer request
```

```
            }
            else if(source == runAddFileScriptButton)
            {
                  scriptStartTime = System.currentTimeMillis();
                  tAddFilesToQueue.start();
            }
      }

      public void sendWiFiInfo()
      {
            int[] payload = new int[] { '1', 'u', 'l', 'w', 'i', 'r', 'e', 'l', 'e', 's', 's', ',', 'M', 'a', 'n', 'a', 'g', 'e', 'd', ',', 'A',
'n', 'y' };
            System.out.println("Sending packet: " + xbee.XBeeSendPacket(payload));
      }

      // THREADS
      // Waits for a connection on the socket, and then sets the WiFi connection status to true
      class WiFiConnectionListener implements Runnable
      {
            private volatile Thread threadWiFiConnectionListener;

            void start()
            {
                  threadWiFiConnectionListener = new Thread(this);
                  threadWiFiConnectionListener.start();
            }

            public void run()
            {
                  try
                  {
                        clientSocket = serverSocket.accept();   // Wait for the client to connect on the socket
                        connectedWiFi = true;                           // Let the program know WiFi is
connected
                        clientObjectOutputStream = new
ObjectOutputStream(clientSocket.getOutputStream());
                        clientObjectInputStream = new ObjectInputStream(clientSocket.getInputStream());
                        tWiFiDataListener.start();                      // Start the WiFi data listener
                        threadWiFiConnectionListener = null;    // Kill this thread?
                  }
                  catch (IOException e)
                  {
                        e.printStackTrace();
                  }
            }
      }

      class SendWiFiInfo implements Runnable
      {
            private volatile Thread threadSendWiFiInfo;
```

```
        public void stop()
        {
            threadSendWiFiInfo = null;
        }

        public void start()
        {
            threadSendWiFiInfo = new Thread(this);
            threadSendWiFiInfo.start();
        }

        public void run()
        {
            int[] payload = new int[] { '1', 'u', 'l', 'w', 'i', 'r', 'e', 'l', 'e', 's', 's', ',', 'M', 'a', 'n', 'a', 'g', 'e', 'd',
',', 'A', 'n', 'y' };
            System.out.println("sending connection info packet");
            while(xbee.XBeeSendPacket(payload) !=
ZNetTxStatusResponse.DeliveryStatus.SUCCESS);
            //System.out.println("Sending packet: " + xbee.XBeeSendPacket(payload));
        }
    }

    class AddFilesToQueue implements Runnable
    {
        private volatile Thread threadAddFilesToQueue;

        long ass;
        long file1;
        long file2;
        long file3;
        long file4;
        long file5;
        long file6;
        long file7;
        long file8;
        long file9;

        public void stop()
        {
            threadAddFilesToQueue = null;
        }

        public void start()
        {
            File asstemp = new File(FILEPATH + "ass.jpg");

            File file1temp = new File(FILEPATH + "ap1");
            File file2temp = new File(FILEPATH + "ap2");
            File file3temp = new File(FILEPATH + "ap3");

            /*
```

```
            File file1temp = new File(FILEPATH + "file1");
            File file2temp = new File(FILEPATH + "file2");
            File file3temp = new File(FILEPATH + "file3");
             */
            File file4temp = new File(FILEPATH + "file4");
            File file5temp = new File(FILEPATH + "file5");
            File file6temp = new File(FILEPATH + "file6");
            File file7temp = new File(FILEPATH + "file7");
            File file8temp = new File(FILEPATH + "file8");
            File file9temp = new File(FILEPATH + "file9");

            ass = asstemp.length();
            file1 = file1temp.length();
            file2 = file2temp.length();
            file3 = file3temp.length();
            file4 = file4temp.length();
            file5 = file5temp.length();
            file6 = file6temp.length();
            file7 = file7temp.length();
            file8 = file8temp.length();
            file9 = file9temp.length();

            threadAddFilesToQueue = new Thread(this);
            threadAddFilesToQueue.start();
        }

        public void run()
        {
            try
            {
                System.out.println("Starting add file thread");

                Thread.sleep(5000);
                sendFileTransferRequest(new FileInfo("ap1", file1, true));
                sendFileTransferRequest(new FileInfo("ap2", file2, true));
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
                Thread.sleep(4000);
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
                sendFileTransferRequest(new FileInfo("ap3", file3, true));
            }
            catch (InterruptedException e)
            {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

SmartPowerUser.java

```
package lpcn.xbee;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.Date;

import javax.swing.JButton;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import com.rapplogic.xbee.api.ApiId;
import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBeeAddress64;
import com.rapplogic.xbee.api.XBeeResponse;
import com.rapplogic.xbee.api.zigbee.ZNetRxResponse;
import com.rapplogic.xbee.api.zigbee.ZNetTxStatusResponse;
import com.rapplogic.xbee.util.ByteUtils;

public class SmartPowerUser
{
	public final static Logger log = Logger.getLogger(SmartPowerUser.class);

	/**
	 * @param args
	 */
	public static void main(String[] arguments)
	{
		PropertyConfigurator.configure("log4j.properties");
		UserGUI bf = new UserGUI();
	}
}

class UserGUI extends GeneralGUI implements ActionListener
{
	// CLASS VARIABLES
	String ip = "10.100.147.98";
	boolean connectingWiFi = false;
	boolean firstFileEver = true;

	// THREADS
	AddFilesToQueue tAddFilesToQueue = new AddFilesToQueue();
```

```
SwitchingAlgorithm tSwitchingAlgorithm = new SwitchingAlgorithm();
ShitPackets tShitPackets = new ShitPackets();

// THREAD VARIABLES
long counter_WiFi_QueueEmpty;
long max_WiFi_QueueEmpty;
long zigbeeQueueThreshold;
boolean shittingPackets = false;

// GUI Variables
JButton requestBandwidthButton;
JButton shitPacketsButton;
boolean shitPackets;
long amountPacketsShit;
public Thread shitPacketsThread;
JButton turnOffWiFiButton;
JButton turnOnWiFiButton;
JButton runAddFileScriptButton = new JButton("Run Add Files Script");

// Constructor
public UserGUI()
{
	setTitle("Paty's Disco Jungl");

	requestBandwidthButton = new JButton("Request Bandwidth");
	shitPacketsButton = new JButton("Repeatedly Send Packets");
	turnOffWiFiButton = new JButton("turn off wifi");
	turnOnWiFiButton = new JButton("turn on wifi");

	requestBandwidthButton.addActionListener(this);
	shitPacketsButton.addActionListener(this);
	turnOffWiFiButton.addActionListener(this);
	turnOnWiFiButton.addActionListener(this);
	runAddFileScriptButton.addActionListener(this);

	pane.add(requestBandwidthButton);
	pane.add(shitPacketsButton);
	pane.add(turnOffWiFiButton);
	pane.add(turnOnWiFiButton);
	pane.add(runAddFileScriptButton);

	add(pane);
	setVisible(true);

	amountPacketsShit = 0;
	packetListener = new PacketListener()
	{
		public void processResponse(XBeeResponse response)
		{
			//System.out.println("Received packet: " + response);
			if (response.getApiId() == ApiId.ZNET_RX_RESPONSE)
```

```
                {
                        int[] payload;          // Stores the payload of the received packet
                        ZNetRxResponse ZNetResponse = new ZNetRxResponse();             // Stores the
ZNetRxResonse type of the received packet

                        ZNetResponse = (ZNetRxResponse) response;       // Cast received packet to the
correct type (ZNetRxResonse)
                        payload = ZNetResponse.getData();                       // Store the data from the
packet into the payload array

                        switch(payload[0])      // Look at the status byte (first byte) of the payload
                        {
                        case '0':       // WIFI_REQUEST : Requests a Wi-Fi access point to connect to
                              // not needed on the user node
                              break;

                        case '1':       // WIFI_INFO : Contains Wi-Fi access point information
                              System.out.println("received connection info packet");
                              String apInfoString = removeFirstChar(ByteUtils.toString(payload));     //
Strip the status (first) byte from the payload
                              String[] apInfo = apInfoString.split(",");      // Split the data into separate
strings for file name and size
                              String essid = apInfo[0];                                 // Store the essid
                              String mode = apInfo[1];                                  // Store the mode
                              String ap = apInfo[2];                                    // Store the ap
                              connectWiFi(essid, mode, ap);           // Connect to the WiFi network
                              break;

                        case '2':       // WIFI_STOP : Informs Wi-Fi AP that the connection is closing
                              // not needed on the user node
                              break;

                        case '3':       // FILE_TRANSFER_REQUEST : Contains file information
                              String fileInfoString = removeFirstChar(ByteUtils.toString(payload));   //
Strip the status (first) byte from the payload
                              String[] fileInfo = fileInfoString.split(",");          // Split the data into
separate strings for file name and size
                              String fileName = fileInfo[0];                                    // Store the file
name
                              long fileSize = Long.parseLong(fileInfo[1]);            // Convert the file size
string into a long and store it
                              //System.out.println("Received file transfer request. Adding to queue: " +
fileName);
                              queueFiles.add(new FileInfo(fileName, fileSize, false));
                              queueBytes = queueBytes + fileSize;
                              //receiveFileTransferRequest(fileName, fileSize);               // Handle the file
transfer request
                              if(firstFileEver)
                              {
                                    firstFileEver = false;
                                    scriptStartTime = System.currentTimeMillis();
```

```
                                        System.out.println("Starting test simulation at time: " +
scriptStartTime + " (" + new Date().toString() + ")");
                                }
                                break;

                        case '4':       // FILE_DATA : Contains 70 bytes of file data
                                int[] dataInt = removeFirstArrayElement(payload);       // Strip the status
(first) byte from the payload
                                queueDataReceived.add(dataInt);         // Add the byte array to the received
data queue
                                break;

                        case '5':       // INCOMING_FRIEND_TO_RECEIVE : Contains file information
                                friendReadyToReceive = true;
                                break;

                        default:
                                //System.out.println("Ignoring unexpected response: " + response);
                                break;
                        }
                    }
                }
            };
        }

        // Event handler
        public void actionPerformed(ActionEvent event)
        {
            Object source = event.getSource();
            if(source == startButton)
            {
                //System.out.println("Opening xbee interface");
                openXBeeInterface(devicePort.getText(), 9600);
                //System.out.println("Xbee interface opened");
                xbee.setDestinationAddress64(new XBeeAddress64(0, 0x13, 0xa2, 0, 0x40, 0x30, 0xc1,
0x47));
                while(!xbee.xbee.isConnected())
                {
                    // Do nothing until the xbee has been connected to
                }
                xbee.xbee.addPacketListener(packetListener);
                //System.out.println("Added a packet listener");
                stopNetworkManager();
                tWiFiDataListener.stop();
                turnOffWiFi();
                tSwitchingAlgorithm.start();
            }
            else if(source == sendPacketButton)
            {
                //ZNetTxStatusResponse response;
                //ZNetTxStatusResponse response;
```

```
                //int[] payload = new int[] { 'P', 'a', 't' };
                //response = (ZNetTxStatusResponse) xbee.XBeeSendPacket(payload);
                //System.out.println("Packet sent. Delivery status is: " + response.getDeliveryStatus());
            }
            else if(source == sendFileButton)
            {
                File fileToSend = new File(fileLocation.getText());           // Look up the file on the hard
drive
                FileInfo fileInfo = new FileInfo(fileToSend.getName(), fileToSend.length(), true);     //
Create a FileInfo object
                sendFileTransferRequest(fileInfo);            // Send a file transfer request
            }
            else if(source == requestBandwidthButton)
            {
                int[] payload = new int[] { '0' };
                readyToSend = false;
                xbee.XBeeSendPacket(payload);
            }
            else if(source == shitPacketsButton)
            {
                if(shittingPackets)
                {
                    tShitPackets.stop();
                    shittingPackets = false;
                }
                else
                {
                    tShitPackets.start();
                    shittingPackets = true;
                }
                /*
                shitPackets = true;
                if(shitPacketsThread == null)
                {
                    shitPacketsThread.start();
                }
                else
                    shitPackets = false;
                 */
            }
            else if(source == turnOffWiFiButton)
            {
                turnOffWiFi();
            }
            else if(source == turnOnWiFiButton)
            {
                connectWiFi("ulwireless", "Managed", "any");
            }
            else if(source == runAddFileScriptButton)
            {
                scriptStartTime = System.currentTimeMillis();
```

```
                tAddFilesToQueue.start();
            }
        }

        // Turn off the network manager
        public void stopNetworkManager()
        {
            try
            {
                Runtime rt = Runtime.getRuntime();
                Process proc = rt.exec("sudo service network-manager stop");
                int exitVal = proc.waitFor();
                //System.out.println("Attempting to stop network manager: " + exitVal);
            }
            catch (Throwable t)
            {
                t.printStackTrace();
            }
        }

        // Shut down the Wi-Fi card
        public void turnOffWiFi()
        {
            try
            {
                connectedWiFi = false;  // Let the rest of the program know that Wi-Fi is NOT connected

                //System.out.println("Is the socket connected? " + clientSocket.isConnected());
                if(clientSocket.isConnected())          // If the socket is connected to anything
                {
                    //clientObjectInputStream.close();
                    clientSocket.close();                // Close the socket
                    //System.out.println("Closed the socket");
                }

                Runtime rt = Runtime.getRuntime();
                //Process proc = rt.exec("sudo ifconfig wlan0 down");     // Shut down the Wi-Fi card
                Process proc = rt.exec("sudo sh /home/pdesantis/Documents/workspace/xbee-
api/src/lpcn/xbee/usbcontrol_OFF.sh");
                int exitVal = proc.waitFor();          // Wait until it has been brought down
            }
            catch (Throwable t)
            {
                t.printStackTrace();
            }
        }

        // Turn on the Wi-Fi card, wait until it associates, then open up a socket to the access point
        public void connectWiFi(String essid, String mode, String ap)
        {
            try
```

```
            {
                Runtime rt = Runtime.getRuntime();
                //Process proc = rt.exec("/home/pdesantis/Documents/workspace/xbee-
api/src/lpcn/xbee/wireless.sh " + essid + " " + mode + " " + ap);
                Process proc = rt.exec("sudo sh /home/pdesantis/Documents/workspace/xbee-
api/src/lpcn/xbee/usbcontrol_ON.sh " + essid + " " + mode + " " + ap);
                int exitVal = proc.waitFor();       // Wait until Wi-Fi has been brought up
                if(exitVal == 0)
                {
                    System.out.println("Attempting to connect to wifi: SUCCESS!");
                    System.out.println("Attempting to open socket...");
                    clientSocket = new Socket(ip, port);    // Connect to the access point
                    clientObjectOutputStream = new
ObjectOutputStream(clientSocket.getOutputStream());
                    clientObjectInputStream = new ObjectInputStream(clientSocket.getInputStream());
                    tWiFiDataListener.start();                      // Start the WiFi data listener
                    System.out.println("Socket connection is " + clientSocket.isConnected());
                    connectedWiFi = true;               // Let the rest of the program know that Wi-Fi is
connected
                    connectingWiFi = false;                     // TODO comment this
                    readyToSend = true;                     // Let the rest of the program know that it is
okay to send data again
                }
                else
                {
                    //System.out.println("attempting to connect to wifi: FAILURE!");
                }
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }

        class AddFilesToQueue implements Runnable
        {
            private volatile Thread threadAddFilesToQueue;

            long ass;
            long file1;
            long file2;
            long file3;
            long file4;
            long file5;
            long file6;
            long file7;
            long file8;
```

```
long file9;

public void stop()
{
	threadAddFilesToQueue = null;
}

public void start()
{
	File file1temp = new File(FILEPATH + "user1");
	File file2temp = new File(FILEPATH + "user2");
	File file3temp = new File(FILEPATH + "user3");

	File asstemp = new File(FILEPATH + "ass.jpg");
	/*
	File file1temp = new File(FILEPATH + "file1");
	File file2temp = new File(FILEPATH + "file2");
	File file3temp = new File(FILEPATH + "file3");
	 */
	File file4temp = new File(FILEPATH + "file4");
	File file5temp = new File(FILEPATH + "file5");
	File file6temp = new File(FILEPATH + "file6");
	File file7temp = new File(FILEPATH + "file7");
	File file8temp = new File(FILEPATH + "file8");
	File file9temp = new File(FILEPATH + "file9");

	ass = asstemp.length();
	file1 = file1temp.length();
	file2 = file2temp.length();
	file3 = file3temp.length();
	file4 = file4temp.length();
	file5 = file5temp.length();
	file6 = file6temp.length();
	file7 = file7temp.length();
	file8 = file8temp.length();
	file9 = file9temp.length();

	threadAddFilesToQueue = new Thread(this);
	threadAddFilesToQueue.start();
}

public void run()
{
	try
	{
		System.out.println("Starting add file thread");

		Thread.sleep(3000);
		sendFileTransferRequest(new FileInfo("user1", file1, true));
		Thread.sleep(3000);
		sendFileTransferRequest(new FileInfo("user2", file2, true));
```

```
				sendFileTransferRequest(new FileInfo("user3", file3, true));
			}
			catch (InterruptedException e)
			{
				e.printStackTrace();
			}
		}
	}

	class ShitPackets implements Runnable
	{
		private volatile Thread threadShitPackets;
		long start;
		long end;
		int packetsSent;

		public void stop()
		{
			threadShitPackets = null;
		}

		public void start()
		{
			threadShitPackets = new Thread(this);
			threadShitPackets.start();
		}

		public void run()
		{
			Thread thisThread = Thread.currentThread();
			int[] payload = new int[] { '4', 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5 };
			System.out.println("size of payload is :" + payload.length);
			packetsSent = 0;
			start = System.currentTimeMillis();
			while(threadShitPackets == thisThread)
			{
				xbee.XBeeSendAsynchronousPacket(payload);
				packetsSent++;
			}
			end = System.currentTimeMillis();
			System.out.println("STOP STOP STOP Amoiunt of packets sent: " + packetsSent + " ,
total time: " + (end-start) + " milliseconds");
		}
	}

	// Takes packet from received from socket and adds it to the data queue
	//
	class SwitchingAlgorithm implements Runnable
	{
```

```
	private volatile Thread threadSwitchingAlgorithm;

	void stop()
	{
		threadSwitchingAlgorithm = null;
	}

	void start()
	{
		threadSwitchingAlgorithm = new Thread(this);
		threadSwitchingAlgorithm.start();
	}

	public void run()
	{
		Thread thisThread = Thread.currentThread();
		System.out.println("HEY PAT HEY PAT HEY PAT switching algorithm started");

		int[] payload;
		counter_WiFi_QueueEmpty = 0;

		// move these to a radio button event listener thing
		max_WiFi_QueueEmpty = 100;
		zigbeeQueueThreshold = 6*1024;

		while(threadSwitchingAlgorithm == thisThread)
		{
			if(connectedWiFi)	// WIFI is on
			{
				// Reset ZigBee counters

				// Check to see if the queue is empty
				if(queueBytes == 0)
				{
					counter_WiFi_QueueEmpty++;
				}
				else
				{
					counter_WiFi_QueueEmpty = 0;
				}

				if(counter_WiFi_QueueEmpty == max_WiFi_QueueEmpty)
				{
					System.out.println("Queue has been empty for long time. Turning off Wi-
Fi");
					// Inform the AP that Wi-Fi is shutting off by sending a 'WIFI_STOP'
packet
					// Send the packet, check whether it was delivered successfully or not, and
if not, repeat sending it until it succeeds
					payload = new int[] { '2' };
```

```
                                while(xbee.XBeeSendPacket(payload) !=
ZNetTxStatusResponse.DeliveryStatus.SUCCESS);

                                // Turn off Wi-Fi
                                turnOffWiFi();

                                // Reset the WiFi counters
                                counter_WiFi_QueueEmpty = 0;
                        }

                }
                else    // ZIGBEE is on
                {
                        if(!connectingWiFi)
                        {
                                // Reset Wi-Fi counters
                                counter_WiFi_QueueEmpty = 0;

                                if(queueBytes >= zigbeeQueueThreshold)
                                {
                                        connectingWiFi = true;
                                        System.out.println("Data Queue is too big. Switching to Wi-Fi");
                                        // Request Wi-Fi
                                        payload = new int[] { '0' };
                                        readyToSend = false;
                                        xbee.XBeeSendPacket(payload);
                                }
                        }
                }
                try
                {
                        Thread.sleep(100);      // Sleep for 0.1 seconds
                }
                catch (InterruptedException e)
                {
                        e.printStackTrace();
                }
        }
    }
}
```

XBeeInfo.java

```
package lpcn.xbee;

import com.rapplogic.xbee.api.XBeeAddress16;
import com.rapplogic.xbee.api.XBeeAddress64;

public class XBeeInfo
```

```
{
	XBeeAddress16 address16;
	XBeeAddress64 address64;
	String name;

	public XBeeInfo(String name, XBeeAddress16 address16, XBeeAddress64 address64)
	{
		this.name = name;
		this.address16 = address16;
		this.address64 = address64;
	}

	XBeeAddress16 getAddress16()
	{
		return this.address16;
	}

	XBeeAddress64 getAddress64()
	{
		return this.address64;
	}

	String getName()
	{
		return this.name;
	}

	void setAddress16(XBeeAddress16 address16)
	{
		this.address16 = address16;
	}

	void setAddress64(XBeeAddress64 address64)
	{
		this.address64 = address64;
	}

	void setName(String name)
	{
		this.name = name;
	}
}
```

XBeeInterface.java

```
package lpcn.xbee;

import java.util.ArrayList;

import org.apache.log4j.Logger;
```

```
import com.rapplogic.xbee.api.ApiId;
import com.rapplogic.xbee.api.AtCommand;
import com.rapplogic.xbee.api.AtCommandResponse;
import com.rapplogic.xbee.api.PacketListener;
import com.rapplogic.xbee.api.XBee;
import com.rapplogic.xbee.api.XBeeAddress16;
import com.rapplogic.xbee.api.XBeeAddress64;
import com.rapplogic.xbee.api.XBeeException;
import com.rapplogic.xbee.api.XBeeResponse;
import com.rapplogic.xbee.api.XBeeTimeoutException;
import com.rapplogic.xbee.api.zigbee.AssociationStatus;
import com.rapplogic.xbee.api.zigbee.NodeDiscover;
import com.rapplogic.xbee.api.zigbee.ZNetRxResponse;
import com.rapplogic.xbee.api.zigbee.ZNetTxRequest;
import com.rapplogic.xbee.api.zigbee.ZNetTxStatusResponse;

public class XBeeInterface
{

	/* CLASS VARIABLES */
	public XBee xbee;
	public XBeeAddress16 destinationAddress16;
	public XBeeAddress64 destinationAddress64;
	public boolean readyToSend;
	public ZNetTxRequest request;
	public ZNetTxStatusResponse response;
	public ArrayList<XBeeInfo> networkTable;

	public final static Logger log = Logger.getLogger(XBeeInterface.class);

	/* CLASS METHODS */
	public XBeeInterface(String XBeePort, int XBeeBaud) throws XBeeException
	{
		networkTable = new ArrayList<XBeeInfo>();

		xbee = new XBee();
		//try
		//{
			xbee.open(XBeePort, XBeeBaud);
			// wait 1/2 second to allow association with network
			//Thread.sleep(500);
			//xbee.sendAtCommand(new AtCommand("RE"));
			//nodeDiscovery();
		//}
		/*catch (InterruptedException e)
		{
			e.printStackTrace();
		}
		finally
		{
```

```
		}*/
		setDestinationAddress64(new XBeeAddress64(0, 0x13, 0xa2, 0, 0x40, 0x30, 0xc1, 0x47));
		request = new ZNetTxRequest(new XBeeAddress64(0, 0x13, 0xa2, 0, 0x40, 0x30, 0xc1, 0x47),
new int[] { 0x00 });
		request.setFrameId(xbee.getNextFrameId());
		setReadyToSend(true);
	}

	public int[] XBeeReceivePacket()
	{
		ZNetRxResponse response = new ZNetRxResponse();
		try
		{
			response = (ZNetRxResponse) xbee.getResponse();

		}
		catch (XBeeException e)
		{
			e.printStackTrace();
		}
		return response.getData();
	}

	public ZNetTxStatusResponse.DeliveryStatus XBeeSendPacket(int[] payload)
	{
		// Set the destination address and the payload of the packet
		request.setDestAddr64(getDestinationAddress64());
		request.setPayload(payload);

		try
		{
			// Send the packet
			response = (ZNetTxStatusResponse) xbee.sendSynchronous(request, 10*1000);
			/*
			System.out.println("---");
			System.out.println("request is: " + request);
			System.out.println("---");
			System.out.println("response is :" + response);
			*/

			// Update frame ID for next request
			request.setFrameId(xbee.getNextFrameId());

			// If the packet was successfully delivered
			if (response.getDeliveryStatus() == ZNetTxStatusResponse.DeliveryStatus.SUCCESS)
			{
				setReadyToSend(true);

				// Check to see if the 16-bit address has changed from the stored value. If it has
changed, then update it. This allows for faster routing
				if (response.getRemoteAddress16().equals(XBeeAddress16.ZNET_BROADCAST))
```

```
                {
                    request.setDestAddr16(response.getRemoteAddress16());
                }
            }
            else
            {
                return ZNetTxStatusResponse.DeliveryStatus.NETWORK_ACK_FAILURE;
            }
        }
        catch (XBeeTimeoutException e)
        {

//response.setDeliveryStatus(ZNetTxStatusResponse.DeliveryStatus.NETWORK_ACK_FAILURE);

            //ZNetTxStatusResponse.DeliveryStatus failed =
                //NETWORK_ACK_FAILURE;
            //System.out.println("Request timed out");
            System.out.println("Failure! Packet failed to send!");
            System.out.println("The request is:");
            System.out.println(request);
            System.out.println("");
            System.out.println("The response is:");
            System.out.println(response);
            //return response;
            return ZNetTxStatusResponse.DeliveryStatus.NETWORK_ACK_FAILURE;
        }
        catch (XBeeException e)
        {
            return ZNetTxStatusResponse.DeliveryStatus.NETWORK_ACK_FAILURE;
        }
        return response.getDeliveryStatus();
    }

    public void XBeeSendAsynchronousPacket(int[] payload)
    {
        // Set the destination address and the payload of the packet
        request.setDestAddr64(getDestinationAddress64());
        request.setPayload(payload);

        try
        {
            // Send the packet
            xbee.sendAsynchronous(request);

            // Update frame ID for next request
            request.setFrameId(xbee.getNextFrameId());
        }
        catch (XBeeException e)
        {
        }
    }
```

```
	public void nodeDiscovery() throws XBeeException, InterruptedException {
		try
		{
			// Clear the network table
			networkTable.clear();

			// the default Node discovery timeout is 6 seconds
			long nodeDiscoveryTimeout = 6000;

			// Add a packet listener to listen for a response

			PacketListener packetListenerND = new PacketListener()
			{
				public void processResponse(XBeeResponse response)
				{
					if (response.getApiId() == ApiId.AT_RESPONSE)
					{
						NodeDiscover nd = NodeDiscover.parse((AtCommandResponse)response);
						System.out.println("Node discover response is: " + nd);
						XBeeInfo responder = new XBeeInfo(nd.getNodeIdentifier(),
nd.getNodeAddress16(), nd.getNodeAddress64());

						// If the responding node is not in the list, add it
						if(!networkTable.contains(responder))
						{
							networkTable.add(responder);
						}
					}
				}
			};
			xbee.addPacketListener(packetListenerND);
			System.out.println("Sending node discover command");

			// Send the Node Discovery command
			xbee.sendAsynchronous(new AtCommand("ND"));
			// wait for nodeDiscoveryTimeout milliseconds
			Thread.sleep(nodeDiscoveryTimeout);
			xbee.removePacketListener(packetListenerND);
			System.out.println("Time is up!  You should have heard back from all nodes by now.  If
not make sure all nodes are associated and/or try increasing the node timeout (NT)");
		}
		finally
		{
		}
	}

	public void associationStatus() throws XBeeException {
		// get association status - success indicates it is associated to another XBee
		AtCommandResponse response = (AtCommandResponse) xbee.sendAtCommand(new
AtCommand("AI"));
```

```
            System.out.println("Association Status is " + AssociationStatus.get(response));
        }

        /* GETTERS AND SETTERS */
        public void setDestinationAddress16(XBeeAddress16 destinationAddress16) {
            this.destinationAddress16 = destinationAddress16;
        }

        public XBeeAddress16 getDestinationAddress16() {
            return destinationAddress16;
        }

        public void setDestinationAddress64(XBeeAddress64 destinationAddress64) {
            this.destinationAddress64 = destinationAddress64;
        }

        public XBeeAddress64 getDestinationAddress64() {
            return destinationAddress64;
        }

        public void setReadyToSend(boolean readyToSend) {
            this.readyToSend = readyToSend;
        }

        public boolean isReadyToSend() {
            return readyToSend;
        }

        public void setRequest(ZNetTxRequest request) {
            this.request = request;
        }

        public ZNetTxRequest getRequest() {
            return request;
        }
}
```

usbcontrol_OFF.sh

```
cd /sys/bus/usb/devices/2-0:1.0/power/
sudo ifconfig wlan1 down
sudo ifconfig wlan0 down
sudo rfkill block 1
sudo dhclient -r wlan1
#sleep 1;
sudo dhclient -r wlan1
#sleep 1;
sudo echo suspend > level
echo POWER DOWN
```

usbcontrol_ON.sh

```
#!/bin/bash

echo $1 $2 $3 ' -> echo $essid $mode $ap'

echo ESSID: $1
echo MODE: $2
echo AP: $3

cd /sys/bus/usb/devices/2-0:1.0/power/
echo on > level

#echo SLEEP 10 Seconds
#sleep 10;
#ifconfig wlan1 down
#dhclient -r wlan1
rfkill unblock 1
#ifconfig wlan1 up
#sleep 2;
#ifconfig wlan1 down
iwconfig wlan1 essid $1
iwconfig wlan1 mode $2
ifconfig wlan1 up
#echo Sleeping 3 Seconds
#sleep 3;
dhclient wlan1
echo POWER UP AND CONNECTED TO $1
```

Browsing Simulation Script

```
// …
// IMPORTANT NOTE: This file is truncated because it spans over 60 pages
// …

//CasBrowsingSimScript
//Mimic my casual browsing
//Browses through wikipedia, then watches a 1 minute youtube video.
//This script only adds files, or simulates downloading/receiving, the server
//AP should run this
//Follows BrowsingNatural.ods
//File sizes and names:
//file1 40 bytes
//file2 132 bytes
//file3 341 bytes
//file4 552 bytes
//file5 576 bytes
//file6 777 bytes
//file7 1500 bytes
//file8 808 bytes
```

```
//file9 1450 bytes

//Start
class AddFilesToQueue implements Runnable
{
	private volatile Thread threadAddFilesToQueue;

	long file1;
	long file2;
	long file3;
	long file4;
	long file5;
	long file6;
	long file7;
	long file8;
	long file9;

	public void stop()
	{
		threadAddFilesToQueue = null;
	}

	public void start()
	{
		File file1temp = new File(FILEPATH + "file1");
		File file2temp = new File(FILEPATH + "file2");
		File file3temp = new File(FILEPATH + "file3");
		File file4temp = new File(FILEPATH + "file4");
		File file5temp = new File(FILEPATH + "file5");
		File file6temp = new File(FILEPATH + "file6");
		File file7temp = new File(FILEPATH + "file7");
		File file8temp = new File(FILEPATH + "file8");
		File file9temp = new File(FILEPATH + "file9");

		file1 = file1temp.length();
		file2 = file2temp.length();
		file3 = file3temp.length();
		file4 = file4temp.length();
		file5 = file5temp.length();
		file6 = file6temp.length();
		file7 = file7temp.length();
		file8 = file8temp.length();
		file9 = file9temp.length();

		threadAddFilesToQueue = new Thread(this);
		threadAddFilesToQueue.start();
	}

	public void run()
	{
		try
```

```
                {
                        sendFileTransferRequest(new FileInfo("file1", file1, true));
                        System.out.println("Starting add file thread");
                        Thread.sleep(3000);                                     //wait 3 sec
                        sendFileTransferRequest(new FileInfo("file1", file1, true));            //queue
the 40 byte file
                        Thread.sleep(3000);                                     //wait 3 sec
                        sendFileTransferRequest(new FileInfo("file1", file1, true));
        //intending to send 140 kB total, sent 40 bytes Spike 1
                        sendFileTransferRequest(new FileInfo("file2", file2, true));            //132
                        sendFileTransferRequest(new FileInfo("file3", file3, true));            //341
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //777
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //777
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file5", file5, true));            //576
                        sendFileTransferRequest(new FileInfo("file4", file4, true));            //552
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //777
                        sendFileTransferRequest(new FileInfo("file1", file1, true));            //40
                        sendFileTransferRequest(new FileInfo("file3", file3, true));            //341
                        sendFileTransferRequest(new FileInfo("file2", file2, true));            //132
                        sendFileTransferRequest(new FileInfo("file4", file4, true));            //552
                        sendFileTransferRequest(new FileInfo("file1", file1, true));            //40
                        sendFileTransferRequest(new FileInfo("file2", file2, true));            //132
                        sendFileTransferRequest(new FileInfo("file3", file3, true));            //341
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file7", file7, true));            //1500
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //777
// …
// NOTE: This file is truncated because it spans over 60 pages
// …
                        //sent 17840 bytes in 20 packets
                        Thread.sleep(1000);             //wait 1s
                        //send 30059 bytes in 40 packets
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file9", file9, true));            //1450
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //576
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file5", file5, true));            //552
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file9", file9, true));            //1450
                        sendFileTransferRequest(new FileInfo("file6", file6, true));            //576
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file8", file8, true));            //808
                        sendFileTransferRequest(new FileInfo("file5", file5, true));            //552
```

```
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file9", file9, true));         //1450
                sendFileTransferRequest(new FileInfo("file6", file6, true));         //576
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file5", file5, true));         //552
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file9", file9, true));         //1450
                sendFileTransferRequest(new FileInfo("file6", file6, true));         //576
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file5", file5, true));         //552
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file9", file9, true));         //1450
                sendFileTransferRequest(new FileInfo("file6", file6, true));         //576
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file5", file5, true));         //552
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file9", file9, true));         //1450
                sendFileTransferRequest(new FileInfo("file6", file6, true));         //576
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file8", file8, true));         //808
                sendFileTransferRequest(new FileInfo("file5", file5, true));         //552
            }
        }
}
```