Power Consumption Analysis of TinyOS Dissemination Protocols

Paul Gildea

A dissertation submitted to
the University of Dublin,
in partial fulfillment of the requirements
for the degree of

Master of Science in Computer Science

Department of Computer Science,
University of Dublin, Trinity College

September 2010

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:_____

Paul Gildea

13th September 2010

Permission to Lend and/or Copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed:_____

Paul Gildea

13th September 2010

Acknowledgements

I would like to thank my supervisor, Meriel Huggard, for her help in the completion of my dissertation and also for her helpful actions when personal obstacles arose. Thanks go to the NDS class for making this an enjoyable year, and also to Laura for thoroughly and attentively proofreading my thesis. Finally I would like to thank Ricardo Carbajo, both a gentleman and a scholar; I am surprised this man gets any work done with the amount of time he gives to others.

Abstract

A Wireless Sensor Network (WSN) is a network that is made up of various dispersed nodes (sensors) that are capable of measuring valuable aspects about their environment. These conditions may range from areas such as location to sound to vibration and there is an ever increasing range of applications being developed in this area. Many applications which have been unfeasible in the past due to limitations such as limited bandwidth, computational requirements and battery life are now able to be researched and developed due to advancements in the area of low power integrated circuits and wireless communications[1]. However these factors still play a large role in the feasibility of applications and using them in the most efficient way is a must in this young technology domain. Limited battery life, processing resources and transmission ranges are characteristics of many wireless environments. These constraints are a key determining factor in all facets of the design of wireless sensor networks and also have an impact on the range of applications that a wireless device may be appropriate for. In this project an analysis of the power consumption of the Dissemination protocols bundled with the TinyOS platform is performed. The aims of this analysis are twofold. Firstly, TinyOS is the operating system upon which Wireless Sensor Networks are developed and these are the Dissemination protocols that are included with it. Developers will be looking to use these supported protocols and it is valuable to have a ranking system and detailed analysis of the power consumption for each of them. Secondly by doing so a generic framework by which to rank all WSN protocols in this manner may be thought about, extending far beyond these particular protocols. This project aims to aid in the extension of the lifetime of wireless network devices that engage in distributed tasks and ensure more efficient distribution of resources by making sure the appropriate protocol is used in the appropriate situation, thereby increasing the length of life of a WSN. It is believed that maximising battery life will massively increase the usability and proficiency of Wireless Sensor Networks.

Table of Contents

List of Figures

List of Tables

1. Introduction

Wireless Sensor Networks (WSNs) is an important technology which is coming into play more as time passes. Business Week proclaimed Wireless sensor networks as one the more vital technologies for this century and it is hailed similarly by many [2] [3]. A wireless sensor network is a multiple hop, self-configuring, wireless network which consists of a collection of nodes (called motes) organised into a cooperative network [4]. Motes are generally spread in groups in different areas and they then can connect to the other motes in the locale wirelessly, forming a network. These motes can be deployed arbitrarily and employed to monitor an environment or system by the measurement of physical parameters. To deploy wireless sensor networks seamlessly into our everyday lives there are ideally a few conditions that should be met.

- Consumption of less than 100µW of average power.
- Cost less than one dollar.
- Have a size of less than $1cm^3$.

Distributed wireless sensor networks have widespread applications and these are expected to increase within coming years. These networks are intended to execute a set of processing jobs depending on the application specifics. Since WSN applications are made up of the simple accumulative formula of sensation plus power and transmission equals valuable outcome, it becomes immediately evident that there are an unlimited range of uses for this technology with some of the main areas being detection, tracking and classification. Measuring the performance for these tasks is well defined and researched, including classification errors and the overall quality of results garnered.

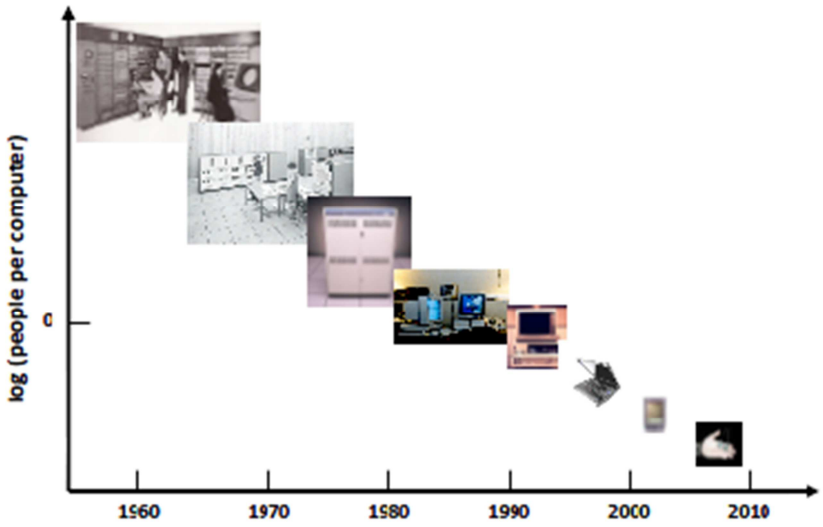

Figure 1: Computer advancement over the last fifty years. [5]

With further development and the continued Moore's law style in advancement of technology (with ever increasing processing power on smaller systems on a chip) these results and applications shall become more and more impressive and useful. Even in their current state, WSNs are already providing excellent telemetry and functions. Currently these networks may be located in a position where the sensors they employ are not easily reached (such as the wilderness, volcanoes [6] etc.) and the networks must be capable of adapting to various environments and requirements. Ideally behaviour of the network will change to manage limited resources more efficiently, recover from broken network points and change its behaviour in response to commands issued. There are numerous areas in which research could prove beneficial in improving such conditions and this thesis deals with arguably the most important of these; power management.

The intention of this project is to primarily rank the dissemination protocols bundled within the TinyOS operating system and in doing so aim towards the design a framework for the ranking of WSN routing protocols via an association of a cost with resource usage. The concept is to accumulate cost data information of protocols by creating and analysing various topologies and scenarios where data is passed between

nodes. The power consumption measured for each scenario can be used to further the view of the value of the protocol as a whole and create a ranking algorithm for Wireless Sensor Networks. This project builds on the output of the recently completed SFI RFP Project, TinyTorrents. TinyTorrents involved the creation of a peer-to-peer based content distribution system for wireless sensor networks. During the course of the TinyTorrents project it became clear that it would be extremely beneficial if there could be a better way of analysing how power is consumed in WSNs and their protocols.

By analysing different scenarios and topologies of various WSN protocols, one could then begin to get an idea of which protocol is best for which situation and use this to place a "cost" on willingness to use a protocol in network level activities such as routing. The resource cost for each protocol in a wireless sensor network can potentially be given a more accurate description. Protocols can then be subsequently chosen based on efficiency and performance versus cost in a myriad of more strictly documented ways; the ability to take advantage of knowing this information being used for all future allocations of protocols.

The research represented here will focus on investigating this avenue of power preservation. Firstly the state of the art is explored. Routing protocols and algorithms that are used to create a WSN are to be examined along with the behaviour of an individual WSN node. Using information assembled here the aim is to investigate how a cost and value in different situations may be associated with the various protocols in use within WSNs. If this is established the development of a methodology for pricing systems at the protocol level can be investigated. One such WSN system that is examined is the TinyTorrents system (which has been implemented in Trinity College Dublin) and working with the associated code base of such a system gives a good idea of how WSN systems work in practice.

Wireless sensor nodes generally run on small batteries and therefore how their energy is used in a system is very important. Increasing longevity is useful to each and every WSN application conceivable. Choice of operational protocol is one area which can be considered when trying to increase this longevity, especially with the rate of increase in the field of wireless sensor networks. In doing this a valuable way to rank new protocols being developed is also there for developers to test against. To develop such a system, the technologies, applications and routing protocols that have been used and that are currently used will be examined in more detail. This will generate a foundation on which to build from and develop any price/cost based protocol ranking algorithms in the future. It is possible to learn much from the existing TinyTorrents system in TCD.

Originally the advance of wireless sensor networks was enthused by the military; they could utilise developed appliances that would generate the ability to garner vital data from the battlefield. WSNs have slowly filtered their way into civilian usage, with the acceleration of implementations and research increasing each year once the topic became more main stream. Systems of many thousand nodes are anticipated and unlike a situation where a lot of mobile phone users are trying to contact a network, gaining no coverage, wireless sensor networks gain from large numbers of nodes in the locality with each node strengthening the capability of the system. Potentially such systems could change the way people live their lives, adding a cornucopia of potential ways to revolutionise the way we live and work. Within the next decade one might expect the world to have a wide array of WSNs spanning its surface, possibly being connected to via the internet. The assortment of possibilities, from home automation to health monitoring, along with the rate of growth in this area yields an area that is well worth researching and advancing.

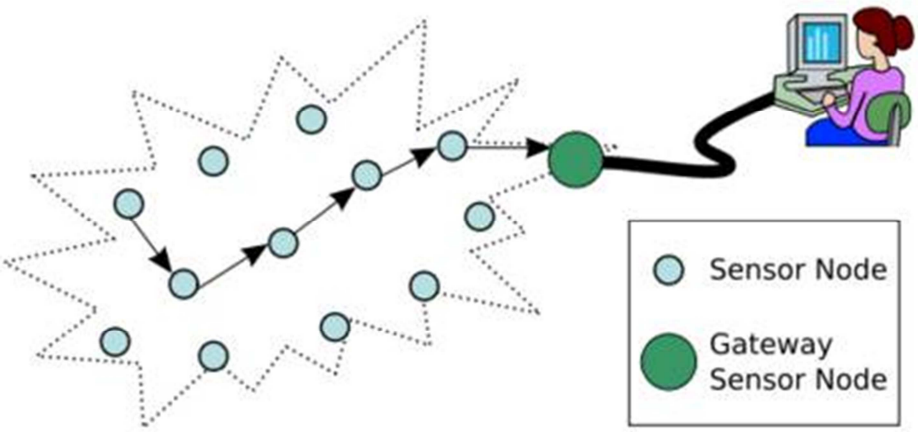

Figure 2: Sample WSN Network. [7]

In the above diagram (Figure 2) a sample Wireless Sensor Network is shown. This is a characteristic multiple hop WSN architecture and depicts a user interacting with a gateway Sensor node, which then interacts with the sensor nodes in the network. Base stations are parts of a WSN that have more energy, computational power and communication resources. Sensor nodes are scalable and are merely restricted by the bandwidth of this gateway node.

2. Hardware

A mote (sensor node) is a sensor in a WSN that has several responsibilities and characteristics. They are small devices for sensing and gathering data. They must have some form of processing power and the ability to send out and retrieve data to and from other nodes in the network. Communication is established by using radio frequency transmission. Nodes usually form a multi-hop mesh style network by contacting neighbouring nodes.

Each of these WSN nodes consists of:

- Some processing power which is made up of microcontrollers, CPUs or DSP chips.
- Along with these there may be multiple forms of memory (program, flash or data).
- A radio Frequency (RF) transceiver which will generally possess a single omni-directional antenna.
- A power source such as batteries or solar panels.
- Varied sensors and actuators.

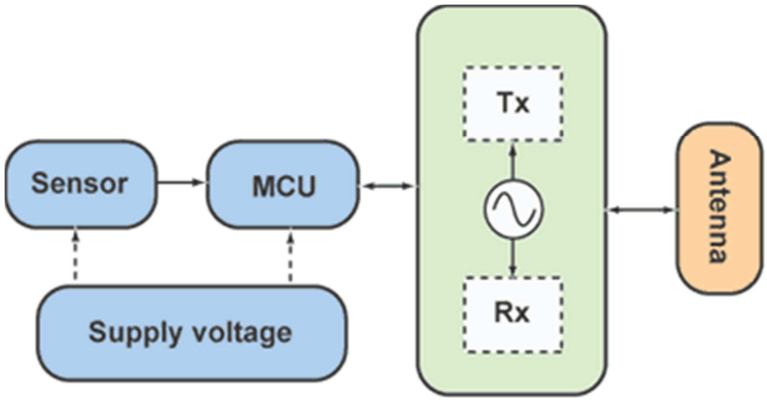

Figure 3: Wireless sensor node architecture. [8]

There are many forms of nodes due to advancements in technology over the years. Before choosing the MicaZ sensor nodes for use in this project, several options were examined. With the advancement of time nodes have become smaller and processing power has increased. Many battery types have been developed, each with advantages and disadvantages. AA style batteries are typical power sources in several forms of nodes today. However, even at this stage nodes can be much smaller than these batteries. Research is being carried out on bettering solar powered batteries, which would take power in from their surroundings.

Design constraints are imposed on making motes due to the power sources currently employed today. Motes must have low power consumption/high power conservation, be as small as is feasible, low amounts of storage and memory. With better power sources, or better conservation algorithms motes will be able to last longer than current incarnations do, which may typically be six months to a year depending on what the motes tasks involve. The aim is to develop motes which cost less than one dollar, are less than one centimetre cubed in size and have a power consumption of less than 100µW of average power. This would enable motes to be deployed seamlessly into our environment.

The concept of smart dust nodes is that there will be a WSN that contains nodes as small as a grain of sand or particle of dust in times to come. Dust nodes have not reached that goal by any means now, but nodes are increasingly getting smaller with nodes appearing that are smaller than coins currently. The motes which may be of interest in this research are Mica2, MicaZ, Telos or Sun Spots. Due to support lent (via TOSSIM and Power TOSSIM Z) and capability, MicaZ were eventually chosen over the other two motes as the mote of choice for this research. Although simulations were used for the most part, the hardware utilised was helpful in that it could be used to verify that the applications functionality was correct. As well as this, using these

motes is a worthwhile tool for a physical “hands on” approach to learning about how to develop for wireless sensor networks and is a great advantage for understanding how to generate simulations.

2.1. Mica2 and MicaZ Motes

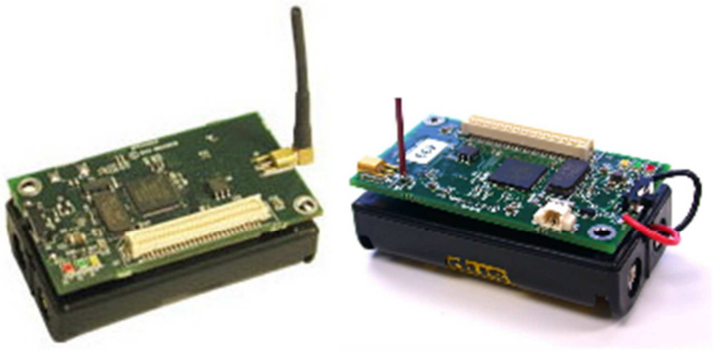

Figure 4: MicaZ and mica2 sensor nodes. [9]

The MicaZ and Mica2 motes are two of the most used motes in the world. They were designed for the NEST (Network Embedded Systems Technology) project. These wireless sensor network motes are developed by Crossbow Technology and are used in the TinyTorrent deployment [10]. They are capable of being wirelessly programmed and have a line-of-sight range of over roughly thirty metres. They have been used in a wide array of different WSN projects. Two AA dry cell batteries are used to power them with automatic battery management that is afforded by the software. The base station approach is used as a transmission paradigm.

2.2. Telos Motes

These motes (shown in figure 5) have a low power profile compared to previous designs, using up to ninety per cent less power. Along with this there is greater throughput and better performance. Telos motes come with a USB 2.0 connection and they eradicate the need for programming and support boards. This functionality allows for the easy use of these motes with WSNs in design, test bed and live deployment environment.

Figure 5: Telos sensor node. [9]

2.3. Sun Spot Motes

Dissimilarly to other obtainable WSN node, the Sun Spot (Figure 6) is built on a java virtual machine. Developed by Sun Microsystems these motes are limited to and programmed in one language. They are programmed in Java which allows developers to produce projects that once demanded an embedded system skill set. Standard Java

IDEs may be used for development. However a consequence of this is that development is at a higher level.

Figure 6: Sun Spot sensor node. [11]

Sun Spot motes have USB access and are manufactured in compliance with the IEEE standards (similar to Mica nodes). They are powered by a rechargeable battery with automatic battery management. The motes communicate using standards created by IEEE also and utilise the base station method for node networking.

2.4. Programming and Sensor Boards

The main programming board throughout the lie of this project was Crossbow Technologies serial interface board, the MIB520. Supplementary to this was the MTS300 sensor board, which could be connected to the MIB520.

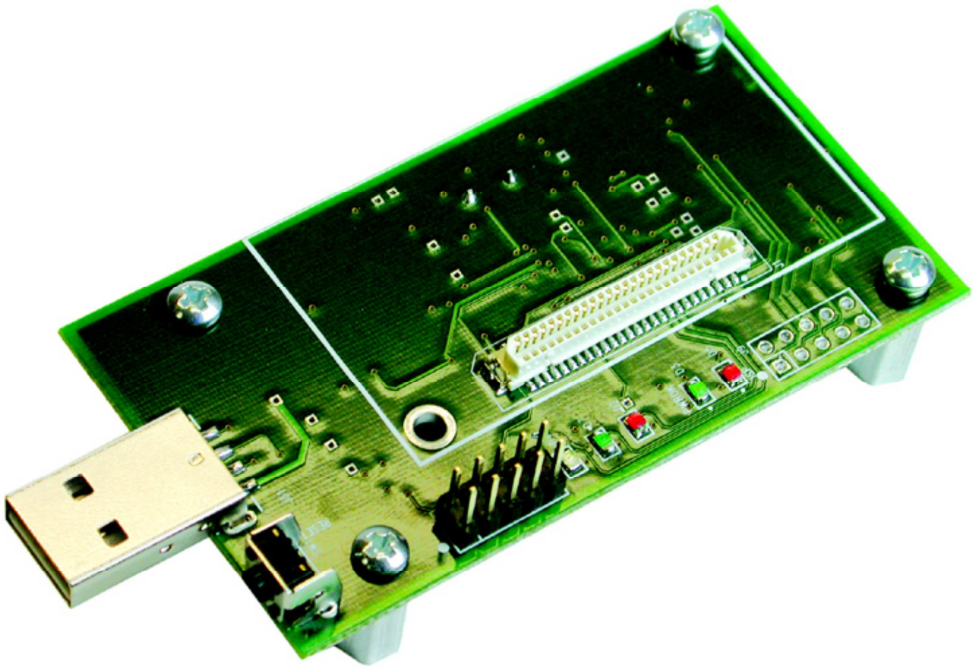

Figure 7: The MIB520 programming board. [12]

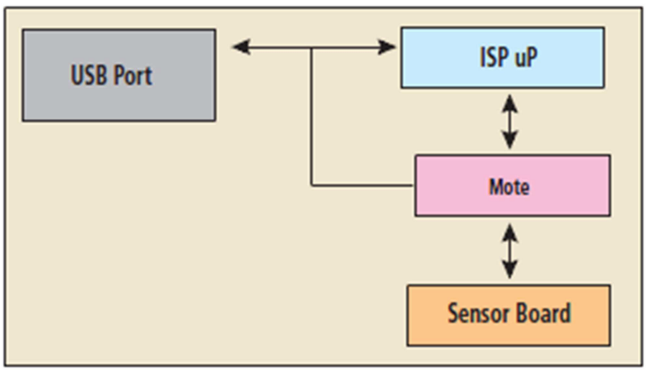

Figure 8: The MIB520 block diagram. [12]

The MIB520 programming board connects to a PC via the USB port (eliminating need for a power supply) and allows code to be uploaded to Mica and Telos motes. There are connections for the motes and various sensor boards to connect to the board shown in Figure 7. Each node can perform as a base station when it is seated into the programming board. Of the two ports provided by the programming board, one serves the function of mote programming whilst the other serves the function of communication over the USB connection. Shown in Figure 8 is a block diagram of the MIB520 which includes ports, on-board processor and possible mote and sensor board attachments.

The MTS300 sensor board (See Figure 9.) is a valuable attachment for the MIB520 programming board. It allows code that utilises measurements such as sound, temperature, light and tone detection to be used easily; one merely attaches the sensor board to the programming board and uploads the code. Upon learning how to program in TinyOS this sensor board was used for various simple applications such as making the sounder beep periodically. If more functionality is required by the user, more advanced boards such as the MTS310 are available with functionality that includes a 2-Axis Accelerometer and a 2-Axis Magnetometer.

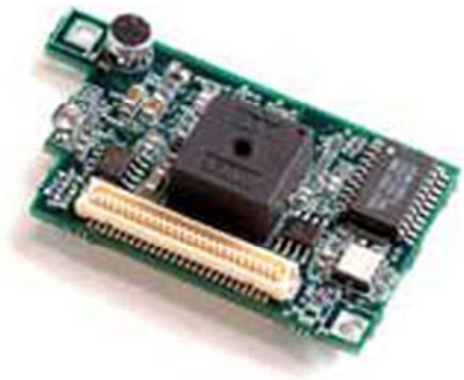

Figure 9: The MTS300 sensor board. [13]

3. Software

3.1. TinyOS

The platform of choice for these motes to run on is an embedded operating system (OS) called TinyOS, including the Mica2 and MicaZ nodes employed in the TinyTorrents project [10]. This OS is an open source project, based on the use of software components, with it being developed for use with Wireless Sensor Networks. These components are interlinked with one another via bidirectional interfaces. Some of the common interfaces granted in TinyOS are related to routing, storing information and sending data. The entire OS is non-blocking (providing high concurrency), producing asynchronous operations and making use of callbacks. It is seen as the only platform on which WSN applications are to be built upon, the platform of choice, yielding an easy choice of technology to use. This OS is written in the NesC programming language and it is intended to support the parallel computations, low memory and low power requirements needed in WSN's. Supporting add-ons are provided in the form of java applications, C tools and shell scripts.

3.2. NesC

NesC is an extension to the basic C language and is considered to integrate the structuring notions with the execution model of TinyOS. It is not a subset of C++; C++

does not understand NesC and similarly NesC does not understand C++. This new extension of C enables the development of applications for the aforementioned Operating System, TinyOS. Applications are made out of components in NesC, it introduces the concept of “wiring” applications together out of individual software components.

The comportment of a component is quantified via a set of interfaces. Interfaces can be both provided or used by components. The interfaces which are used are the interfaces which are provided by other components and are needed for an application to perform its tasks. The provided interfaces are those that embody the functionality afforded to the users. These interfaces are bidirectional, providing both utilities to be realised by the interfaces provider component (commands) and the interfaces user component (events). In general commands are seen to be calling downwards through a hierarchy of component layers towards the hardware level where the interface they might need is implemented. On the other hand events call upwards through these layers towards the original component.

If a user wishes to use the send command to send some data from an interface the command travels through the layers until it reaches the appropriate component, the sendDone event is implemented in the user component and the user will be informed that a send has been correctly achieved with a notification travelling upwards. This particular example was useful in this research as one could edit the component that implemented the send function and gather pertinent information about what nodes were sending data and how much packets were being sent to gather information about power consumption in the various protocols. In this way merely one interface has been used to create an intricate and highly useful interaction between components.

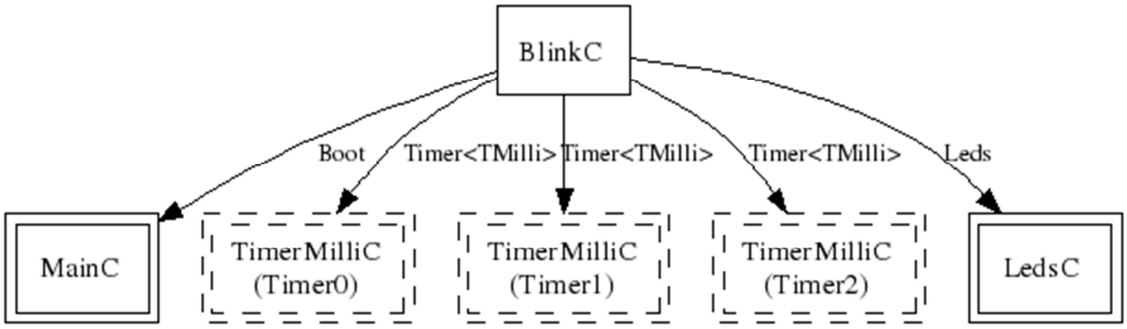

Figure 10: Component interaction in a simple NesC application which flashes LED's.

There are two main types of files in NesC, configuration files and modules. In figure 10 a simple component diagram of an application to blink LED's on and off is shown. One box, such as with the BlinkC component, represents a module with a double box being a configuration file. Dashed lines are included where a component is generic. Modules are where interfaces are implemented, the actual code for the functionality of the program is written in the module files. The configuration files are where the concept of "wiring" comes in. This is a file that connects or "wires" other files together (namely the module files) and specifies how they invoke each other. It connects the interfaces used in the module to the interfaces provided by the other modules.

This means that a WSN application developer can simply build an application from a hierarchy of module layers that are already in place, as well as any additional modules they are writing themselves. This concept of "wiring" components together is quite a useful tool, allowing complex interactions between many simple modules to create a complex application.

3.3. TOSSIM

TOSSIM is the application discrete event simulator for TinyOS. This powerful tool allows whole network simulation as well as providing an easy method of testing actual code that can be used on modes. A developer is able to test and evaluate applications in a precise and repeatable setting. The application written by the developer for the motes is the same code that is necessary for the simulation, allowing simulations to be done seamlessly. This is facilitated by replacing TinyOS components with the simulation equivalent inside of TinyOS. Events in TOSSIM are put into an event queue and sorted via time. With the introduction of the new version of TinyOS support has been added for the MicaZ motes. Since TOSSIM is a library in TinyOS it has to be configured and ran. To do this there are two programming interfaces that can be uses, Python and C++.

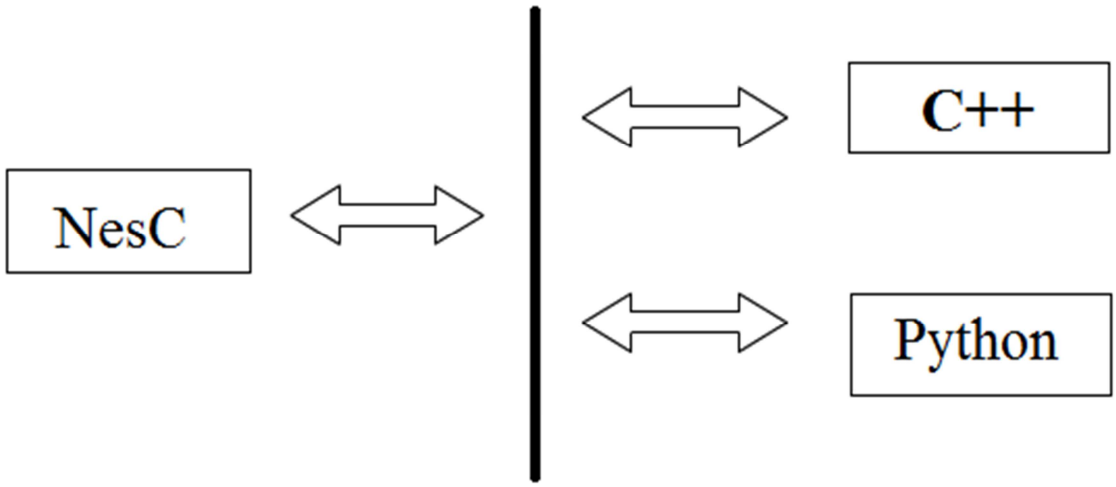

Figure 11: Application written in NesC is run via interfaces connected to TOSSIM.

Figure 11 demonstrates the aforementioned most powerful aspect of TOSSIM and the reason why this simulator is used; the developed application can be run on both motes and via the programming interfaces with the simulator. When simulating an application one has to write the unabridged implementation of it which makes simulations more challenging than an abstract simulation but also means that the code written can then be used in a meaningful way and implanted onto the motes themselves without any editing.

Python is generally the programming interface of choice as it allows one to interact with a simulation dynamically whilst it is running. Python also has the ability to inspect variables and inject packets easily. C++ cannot inspect variables whatsoever and requires a more complex way of injecting packets into a simulation (building C support for the packet type and manually building the packet). The advantages of the C++ implantation are faster speed, as no Python interpreter is needed, along with the ability to code on a standard environment (which aids in debugging). To generate a simulation one writes NesC code that would compile on motes. Along with this the script to configure and run the simulation is run. Instead of compiling and uploading this code to the mote an alternative command is used to instead simulate the application, “make micaz sim” in the case of simulating for MicaZ motes. There are now five steps which are ran through by TOSSIM for making the simulation.

- Writing an XML Schema
- Compiling the application
- Compiling the programming interface
- Building the shared object
- Copying Python Support.

In the first step an XML file is produced which describes the application, this is done via “nesc-dump” and valuable information such as variable types and names are noted. Next include paths in the make files of applications are changed to their simulation equivalent and the “sim.o” object file for configuration is generated. When creating an application, even with simulation in mind, you write it as you would write code for actual deployment, TOSSIM takes care of the make files simulation include paths for you. In the third step support for the Python and C++ programming interfaces is compiled and the “tossim.o” or “pytossim.o” object is created respectively. The penultimate step sees a shared library built of the Python and C++ support as well as TOSSIM code. The final step copies files to the local directory of the simulation, this Python code is the code that calls into the shared object. After these five steps the script that is written in either the Python or C++ programming interface for configuring and running the simulation is run and a successful simulation is attained.

3.4. PowerTOSSIM Z

One valuable feature that TOSSIM does not support at the moment is the collecting of power measurements about an application, support for this was vital in this research and the tool PowerTOSSIM Z [14] was vital in the forwarding of this project. Originally an extension for forming details about power consumption called Power TOSSIM was created for TinyOS version 1.x. This modelled the power consumption of Mica2 nodes. When TinyOS updated to version 2.x the functionality to deal with MicaZ nodes was added. The update of Power TOSSIM that followed however only continued to support the Mica2 architecture.

Power TOSSIM Z was then developed independently to provide support for the MicaZ motes by porting PowerTOSSIM, adding capacity to deal with non-linear battery effects and adding the important aspects for the MicaZ architecture. To install, and use this, a user simply replaces some of the files in the TinyOS installation and adds a CFLAG to the make file of their application. The postprocessor works at runtime and power information about a simulation can be added to a text file. The tool PowerCurses (output shown in Figure 12) has been included to graphically show how much power a mote has remaining, this graphics tool currently runs in Windows installations of TinyOS and PowerTOSSIM Z.

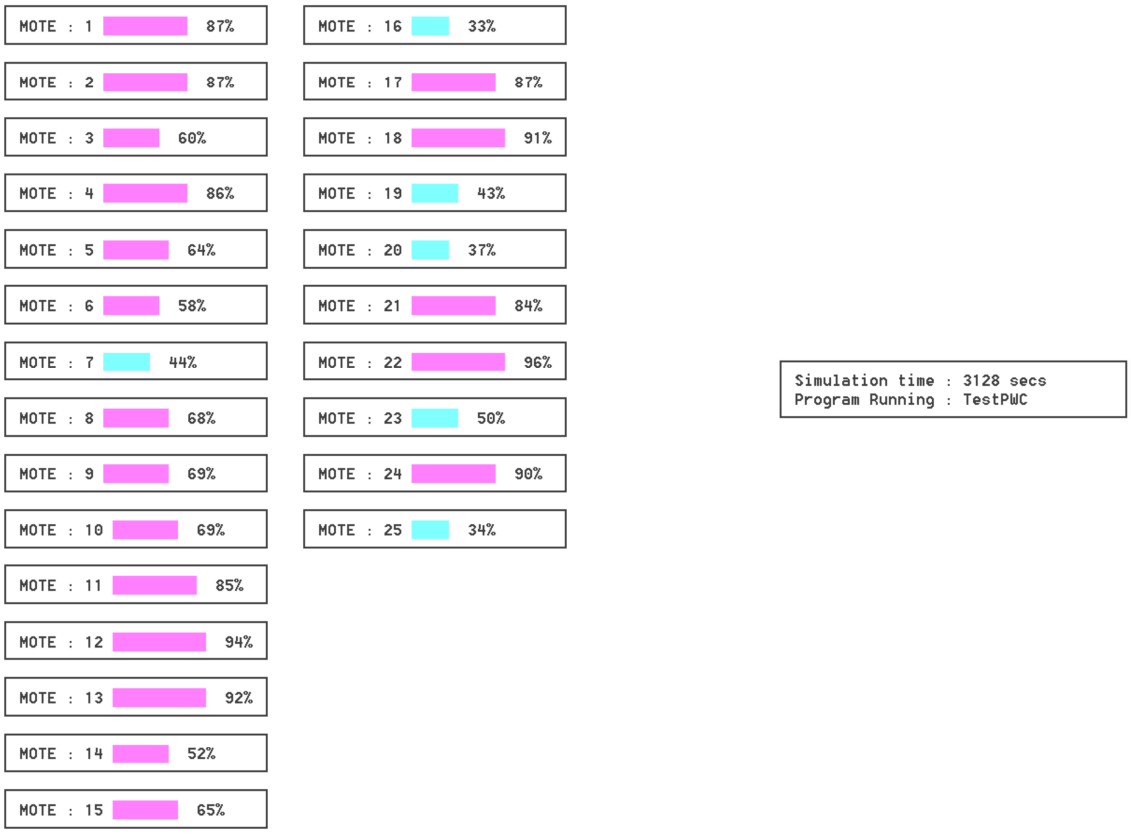

Figure 12: Graphical display of nodes power supply after running a simulation. [14]

3.5. Serial Forwarder & Data Injector

Another useful tool used in the completion of this project is the Serial Forwarder. It is useful for both real mote implementations and simulated tests. With this a packet can be read from a computers serial port in order to allow communication with a WSN via the gateway. The packets being read and written are recorded on the Serial Forwarder GUI and the contents of packets can be read in the terminal via the Java tool Listen.

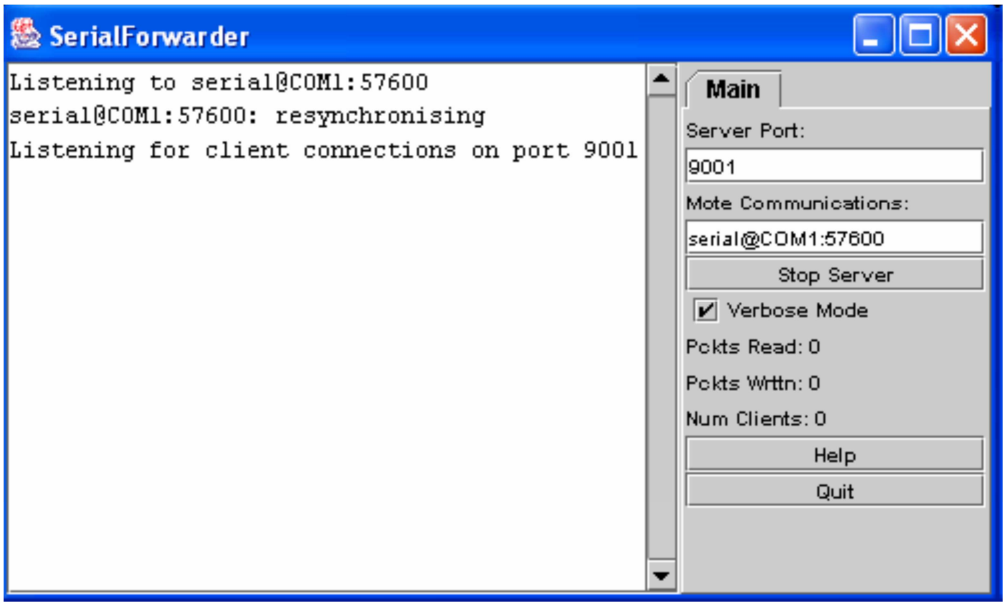

Figure 13: The Serial Forwarder Tool listening on COM1 with the MicaZ baud rate of 57600.

TOSSIM also permits a developer to inject packets into a network. In fact the DIP protocol comes with its own data injector Java application. These applications can enable the developer to send data from a base station instead of from a node and can be useful for running repeatable tests by injecting packets in certain scenarios.

4. State of the Art

4.1. History of Wireless Sensor Networks

The various ways in which wireless sensor networks can be utilised can be reasonably diverse, however they share common architectural and technical concerns. These common interests have spurned advancements in systems ranging across the spectrum of application domains. Originally, as with various technologies, research was driven by the military. Sound Surveillance System (SOSUS) and air defence radar networks were WSNs used during the Cold War. This network sat on the ocean floor and was comprised of acoustic sensors (hydrophones). Strategically placing these hydrophones allowed the Allies to detect and thus track enemy Soviet submarines.

Figure 14: The first SOSUS stations (red) along with the additional stations later employed (green) in the Mid-Atlantic Ridge. [15]

As time passed by, such networks became more advanced yielding more accurate tracking of submarines. Since the Cold War ended the use of nodes is shown by their versatility via SOSUS moving on to a completely different field where it is now used to monitor animal and seismic activity in the ocean (by National Oceanographic and Atmospheric Administration (NOAA)). [16] Further advancement in the air defence systems has led to the system incorporating aerostats sensors and Airborne Warning and Control System planes. The application can currently be operated to monitor and prohibit the entrance of drugs into a country. The application history of WSN's shows that although a system is developed and researched for one purpose, the key features in place allow for the technology to be reused for a different apt purpose in the years to come.

In the work of Mainwaring et al. [17] an in-depth analysis is provided for the use of WSN's in a real-world habitat monitoring environment. The necessary designs and requirements for such systems in general and the necessary network architecture are explored, along with a case study of a seabirds nesting atmosphere. A system is deployed on an island consisting of thirty two WSN nodes with data streamed to the internet. The authors hope that in this way a case for application driven design is put forward, enabling benefits in the future for areas such as communications, data sampling, health monitoring and network tasking.

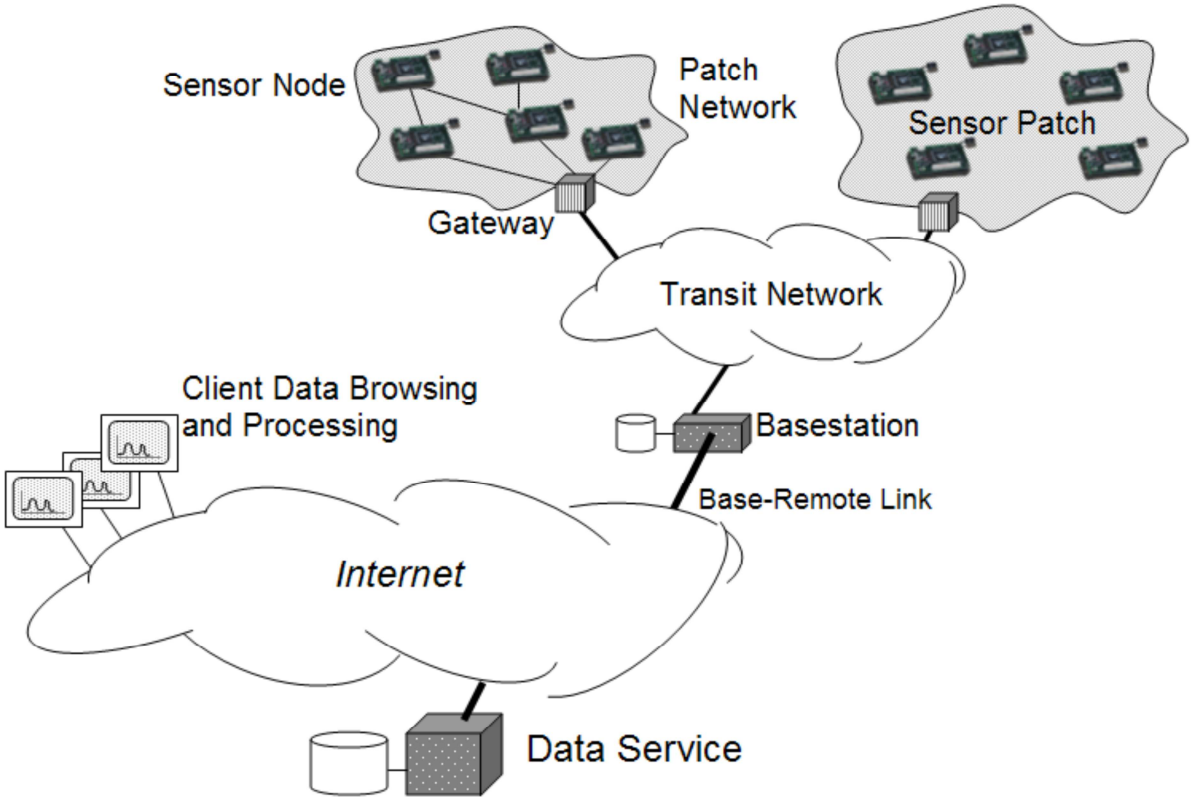

Figure 15: System Architecture for habitat monitoring. [17]

A second prototype network has been instantiated in the San Jacinto Mountains Reserve (JMR) in California to compare and evaluate the implementations offered by the authors. It is hoped that by application driven design a generic habitat monitoring kit can be created. This would be useful to researchers across numerous fields, with the ability to tailor the kits setup parameters to specific research purposes and collect data from areas previously inaccessible. Also, this monitoring provides a basis for sensing modalities which may inspire other forms of WSN networks in varying application domains.

4.2. Application Cost

The nodes in Wireless Sensor Networks contain limited, sometimes irreplaceable, power sources. This is due to limitations in wireless sensor motes; they can only be furnished with a bounded quantity of power. With the capability and purpose of each WSN system comes an inherently different sensing power cost. Many elements vary such as the polling rate for detection of events, complexity of each task and so forth.

In many traditional networks the focus is on quality of service, however in sensor networks one of the most important focuses is the necessary low power consumption. Although power conservation is a concern in other ad-hoc networks also, it is not the most important feature of note as power supplies are seen as something that can be easily replaced. As this is not the case in WSN, additional concern is prevalent. Conservation of power is a key feature in these networks so there must currently be trade-off mechanisms employed which give an operator the option of enhancing network life span at a price. This price would typically be lowering throughput and/or increased delay of transmitted data. It is vital that the power sources in these nodes be maximally used to extend optimal use of such networks. To improve a WSN by means of the domains surrounding power one can:

- Better current power supplies.
- Generate new mote technology/hardware that requires less power.
- Alter protocols/algorithms such that more efficient ones are utilised.
- Allocate a dynamic pricing index to commands and functions so that the network processing pool can be used in the most appropriate way to maintain a correctly functioning system.

In this research the protocols and algorithms used in this area will be focused upon. It is in this way that a possible routing protocol pricing table can be brought about. Maximising the usefulness of current power sources will allow for more focus on the quality of service of these systems and altogether more useful applications being availed of. There is much need for research currently going on in this quarter and there is much focus on protocols and algorithms performance metrics.

There are three primary tasks for a sensor node in a sensor field. First an event is detected, next some data processing is performed and then finally the retransmission of data takes place. This leads to the three classifications of power consumption domains. These include the cost of:

- Sensing
- Processing
- Communication

4.3. Communication Cost

The most intensive use of energy comes from the data communication of the nodes. Nodes both transmit and receive data. When attributing cost to component and tasks in the nodes one takes into consideration everything from frequency synthesizers to power amplifiers and it is necessary that careful consideration is given to each aspect of the power consuming components. This means that along with active power the starting up power consumption is considered.

The starting time, although measured in micro seconds, is non-negligible due to the lock time in phase locked loops. This consumption (P_c) is given as: [18]

$$P_c = N_T[P_T(T_{on} + T_{st}) + P_{out}(T_{on})] + N_R[P_R(R_{on} + R_{st})]$$

Where $P_{T/R}$ is the power consumption by the transmission or reception; T/R_{on} is the length of time that these components are active for; P_{out}, the transmission output power; T / R_{st}, the component start up delay and $N_{T/R}$, the number of times the components are switched on for. Feeding the values for the systems (at the technology level of Shih, E. et al. [18]) quickly leads to the conclusion that as packet size is reduced it is necessary to leave the transmitter on at all times, as the power needed to start up starts to overshadow the active consumption. As we can see there has been much inefficiency and needed improvements in the past of this young technology. Over the years these areas have received much research into making low powered transceivers. [19-21] Continual improvement in both the hardware cost and algorithmic cost has been a foremost concern as transmission is generally hailed as a big cost in ad-hoc networks. Apart from the aforementioned, both processing and sensing are the major energy consuming areas left to contend with.

4.4. Processing

The energy overhead associated with processing is seen to be inconsequential in comparison to communication. An example expressed in the work of Kaiser et al. [21], shows the lengths of disproportion. With a few basic assumptions such as Rayleigh fading and distance loss, some conclusions can be drawn. To transmit one kilobyte of

data over a distance of one hundred metres is approximately equivalent to the energy taken for executing three million instructions (by an hundred million instructions per second processor). Pressed here is the desire for local processing practice, minimising the large overhead occurred with transmission as much as possible. Therefore each node must have the processing capabilities to necessitate any processing that may occur due to the data sensed in its surroundings.

Algorithms can be generated here for maximal energy use such that the voltage and frequency of the processor can be altered dynamically to yield the most efficient use of a system with low power. In fact, if a system is running low on power it ideally should be running only essential tasks and letting the network be aware of this. By applying a price to task components it should be possible that a mote does not run certain tasks and simply not receive the commands and data to perform certain other tasks. This could mean that a node would no longer take part in routing, but may still sense and data process at a possibly altered rate.

4.5. Routing

Routing in a WSN is a complex challenge since the unique inherent characteristics of a WSN differentiate this type of network from other types of wireless networks. The main issue that is to be taken into consideration when designing and implementing routing protocols is the one of environment. Where and how the system will be exercised is foremost for consideration. One cannot simply apply a traditional IP protocol to a network with the particular needs of WSN. The constraints enforced by the technology must be addressed and limitations expunged as much as is feasible. Motes are positioned in an impromptu manner and since there may be large numbers of nodes deployed a global addressing scheme is not practical; it would cause large maintenance overhead. [22]

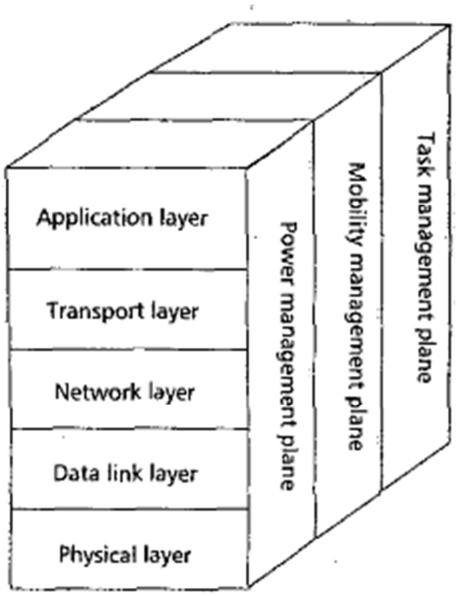

Figure 16: WSN protocol Stack. [23]

In figure 16 above the protocol stack for Wireless Sensor Networks is laid out. This is used by the sink node and also the sensor nodes in a network. The main aims of this stack are to amalgamate power and routing awareness, integrate efforts between wireless nodes, manage power efficiently in a wireless domain and incorporate data with networking protocols.

4.6. Task Management Plane

- Nodes do not have to sense within the same region. Ones that do not have to are turned off.
- This plane schedules the sensing tasks, accounting for balance.

4.7. Mobility Management Plane

- Catalogues and finds movement of motes in the network. The path to the sink mote is established along with the ability of motes to trail their neighbours.

4.7. Power Management Plane

- Decides how a nodes power is used.
- Commands to increase and decrease power along with functionality are issued.
- If battery in a particular node is low this is where one would tell a node to broadcast that it does not wish to take part in routing of messages etc.
- More important tasks can then be completed.

4.8. Application Layer

- Creating rules for data aggregation, clustering of nodes etc.
- Movement of nodes
- Turning nodes on and off.
- Security, such as key distribution.
- Time synchronisation.

4.9. Transport Layer

- Deals with network being interacted with via external networks such as the internet.

- Seals with the use of TCP (with splitting due to window size of TCP not suited to harsh WSN requirements) or UDP protocol.

4.10. Network Layer

- This deals with routing and data supplied by the Transport Layer.
- Finding optimal energy efficient routed based on remaining power and the energy cost of the transmission.
- Ideally methods such as flooding or gossiping are avoided because of energy consumption issues and/or long delay.

4.11. Data Link Layer

- In charge of :
    - Multiplexing data streams
    - Error handling
    - Medium Access Control (MAC)
    - Data Frame Detection

4.12. Physical Layer

- In charge of:
    - Selection of frequency
    - Signal detection

- Modulation of Frequency
- Encryption.

There are many proposed routing protocols that focus on different areas of a network and types of networks themselves. These vary from more traditional wired networks to ad-hoc networks to WSN networks and each of the protocols has specific tasks in mind, taking into account topological constraints. For wireless sensor networks one must take into account this ad hoc distribution, as well as power conservation demands that other network topologies would not consider as important. This is because either the network is relying on a steady power source or even if the network is ad hoc, power sources are seen to be easily interchangeable.

This is not the case with WSN as often the batteries in each mote are incredibly difficult to change, due to hazardous or remote surroundings. Once the nodes are positioned they organise themselves and the coterie forms connections with those members in the adjoining area. If several nodes fail, rerouting of packets and perhaps reorganisation of the network may need to take place, adding importance to the routing protocol in use. The network will not be maintained, as they are mostly unattended by the systems employers, so the nodes themselves must take a large responsibility in the nodal distribution and organisation of themselves. It is ideal that they can be left alone for as long as possible, functioning well. Whereas many traditional networks focus on quality of service and some to a lesser degree focus more significantly on power consumption.

Narrow constraints in storage, power and processing make each action of utmost importance, it is better to yield some functionality over more important tasks such as sensing (design requirements for the WSN network change depending on the

application), so very specific protocols must be implemented. Some of the areas however have a common goal across all applications; these are the physical and link layers. These put the spotlight on the power awareness of the system itself, such as voltage scaling and energy efficient MAC protocols etc. [24-28]. At a network level the purpose of routing is to route messages energy-efficiently whilst maintaining reliability. If several nodes sense the same data due to proximity of an event of interest a lot of redundancy is created, the network level should cater for this.

4.13. Types of routing

A multitude of approaches have been designed appealing to power constraints in WSNs, whether it is reducing the power consumption of the WSN in its entirety or for an individual node. Many focus on constraints such as power consumption and some on other avenues such as quality of service. Some of the various types of protocols developed in these areas will be discussed here. A wide array of such protocols exists, each falling under certain classifications pertaining to the type of routing algorithm used. It is vital to research into adaptation of protocols so that a thorough knowledge of the existent developed protocols is gained.

As discussed in the work of Al-Karaki et al. [22] routing protocols can be generally split into three types, depending on the network structure being used. These are flat-based, hierarchal and location based routing.

- Flat-based Routing
    - Each node is given an equivalent role in the system, along with identical functionality.

- Hierarchal-based Routing
    - In this style of routing each node may play a different role, a de facto hierarchy.
- Location-based Routing.
    - Here a nodes position in a network is taken advantage of when routing messages.

After a protocol has been classified based on the network topology, it can also be split up into categories based on what the implementation of that protocol is being used for.

These protocols can be split up into quality of service, query, multipath, negotiation or coherent based routing dependent on the function. A type of network structure is put in place; then an operational protocol is applied. For instance a Flat-based network routing protocol using a negotiation based protocol is discussed in the work of Heinzelman et al. [29] while a Flat-based network routing protocol using Multipath-based routing is discussed in the work of Govindan et al. [30]

There are three categories that these protocols can also be classified under, pro-active (table-driven), reactive (on-demand) and hybrid (mix of both). Protocols are assigned into a category based on the way the source is able to route to the destination.

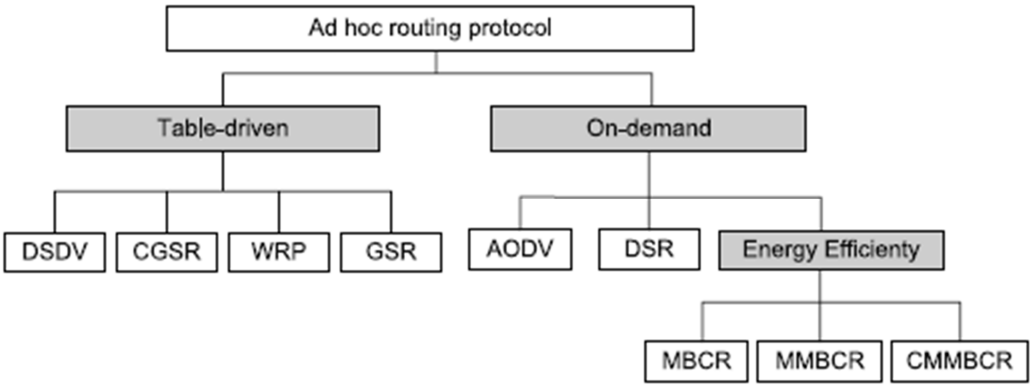

Figure 17: Ad-hoc routing protocols. [31]

The trade-offs between proactive, reactive and hybrid routing protocols are multifaceted. Many factors need to be taken into consideration including the scope of the network, its mobility and the generated message interchange rate.

4.14. Proactive routing

With this protocol genre up to date lists are kept of all destinations and their routes in the background, independent of whether they are needed or not, as well as not taking heed of the traffic on the network. Each node stores some information about the network in tables that are updated each time a new message is received.

The protocol achieves this by periodically releasing routing tables throughout the network. Route information is maintained by nodes periodically sending messages. There are several advantages and disadvantages to this type of routing protocol. There is much data for maintenance, these protocols waste data when there is no traffic on the network due to the superfluous transmission of messages sent by nodes to retain accurate route information.

The following things are needed in implementing this protocol, power, bandwidth and storage which may make it unsuitable to some networks. Another

downside to implementations of this type of protocol is the slow response to the restructuring of a network and as a corollary, a slow response to failed nodes in the network. Conversely there are also advantages to using these protocols, such as the evident ability to gather route information quickly to institute sessions. An example of a proactive protocol is discussed in the work of Chen et al. [32] where a protocol called Global State Routing (GSR) is introduced. In this protocol nodes share link states amongst close peers whilst routing messages are being traded.

It is in this way that an accurate map of the network is obtained. The authors' goal is to achieve a protocol that is efficient at Medium Access Control (MAC) layer for wireless sensor networks. An investigation into varying the size of the control packet (which maps the network topology) to maximise MAC throughput is carried out. The idea is to maintain the knowledge of the entire network gained by proactive routing but to also avoid the inefficient flooding aspect. A routing protocol based on the Link State (LS) [33] routing method and the Distributed Bellman-Ford (DBF) routing protocol is developed. A cost is assigned to find the shortest distance between nodes via Dijkstra's algorithm.

This cost based algorithm benefits inspires work into further developing algorithms on cost based routing. With the LS routing protocol, each node floods the link state information into the whole network (global flooding) once it realises that links change between it and the surrounding nodes. However, flooding causes overhead, particularly if the topology is dynamic. GSR combines two of the best features of these protocols, the topological accuracy gained by LS and the advantage of no flooding from DBF. Some other protocols in this area are DSDV [34], CGSR [35] and WRP [36].

As with LS, in the Destination Sequenced Distance Vector protocol (DSDV) each node cyclically transmits routing information to maintain the routing data gathered by

each mote. The variant of GSR, Clusterhead Gateway Switch Routing (CGSR) founds its routing methods on a hierarchical cluster layout on DSDV. Clusters are made up:

- cluster members
- cluster head
- gateway nodes

A member cluster sends the information to its cluster head when it wants to send data to another cluster, a cluster group, which then forwards the relevant information to the cluster in question via a gateway node. The head of this cluster then transmits the data to the destination member.

The Wireless Routing Protocol (WRP) is introduced in the work of Murthy et al [36]. This exercises an adapted version of the distance-vector routing protocol. Each node in the network keeps a table of routes, distances, a message retransmission list (MRL) as well as associated link costs. The Bellman-Ford algorithm is used for calculation of routes. A record of the distance to a destination mote and its adjoining nodes across the path is kept and classified. This tells the network what is the best path for a message, if a path is incorrect etc. This protocol retains the same advantages of the work proposed by Perkins et al. [34] whilst generating faster convergence and needing less updates to tables, due to the multiple table design employed. However there is a greater cost introduced by the addition of multiple tables which have to be maintained. Better performance comes at the cost of greater processing and storage use.

4.15. Reactive routing

Unlike the table based proactive model for routing, reactive routing reacts to the need to send data. Without such tables there is no information stored on how to reach any node in the given topology. However, data is stored from previous searches of the network. A node looks up a table of these, already searched, routes when it is sending data. The first time a route is needed it is perceptibly not in this database of routes. The best route is found via flooding the network with route request packets. After obtaining the desired route this gets saved to a table so as to make a record and avoid unnecessary searching. To avoid the problem of out of date routes timestamps are used to ensure the route is currently accurate.

This type of protocol should be used in a network with low mobility, as there is a high latency in the discovery of new routes. However static networks should use a table driven method, as adapting routes is not an issue. As well as other disadvantages, large use of flooding in a network leads to congestion and a slower network. Ideally reactive protocols should be used in an ad hoc network with low mobility, where data transfer will account for most of the necessary transactions. Some of the reactive protocols in use are AODV [37], DSR [38], LAR [39], ABR [40], TinyAODV [41], NST-AODV [42] and Tinyhop [43].

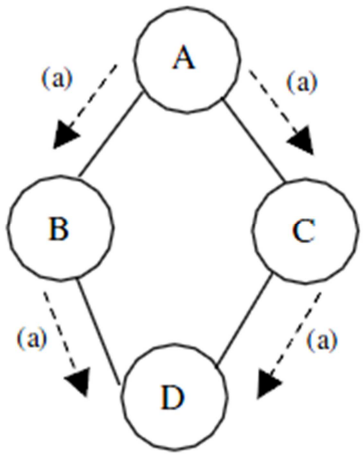

Figure 18: Disadvantage of flooding, node A transmits data to all its neighbours. Node D ends up receiving two copies of the information. [44]

Ad hoc on Demand Distance Vector (AODV) is a reactive protocol presented by Perkins et al. [37]. In this protocol both unicast and multi cast routing is possible. If a node breaks down a new shortest path is calculated, but unlike some other protocols, AODV avoids the Counting to Infinity problem [45] by using sequence numbers on an update. This was also developed by Perkins in the Destination-Sequenced Distance-Vector Routing (DSDV) protocol [46]. Each time a message is sent a request is made for this sequence number which evades the aforementioned counting to infinity problem by eluding the need to repeat route requests.

Since this method reduces the communication traffic and is also a simple practice, little processing or storage is needed. This also carries a downside due to the fact that sessions take a longer time to establish, with more resources needed than with some other approaches. Furthermore, issues can arise with inaccurate entries for routes if the sequence number of the source is old along with high overhead and bandwidth possibly being a factor due to multiple packets being sent with route data and AODV's periodic beaconing process. Two lightweight adaptations of this protocol are Tiny-AODV and Not So Tiny AODV (NST-AODV) which are used with TinyOS, a platform which will be useful in this research.

Like most protocols that can be used for low-power wireless personal area networks (LoWPANs), TinyAODV is an adaptation of the AODV protocol. It is a minimalist adaptation for use with WSN nodes using the TinyOS platform, such as the TinyTorrents system[10], which may be used as a basis for this research document as a good idea on how protocols differ in order to save energy and give ideas on how routing/dissemination protocols can be ranked. Upon implementing version three and for subsequent versions of TinyAODV a new feature was introduced which altered how the protocol worked.

In version two, only the sink node could be the target for a message. However, the newer releases contain the ability for communication between any two nodes in a network. Similarly to AODV, if a message is sent without a table entry existing for that route, the discovery process takes place. The first packet that wished to use this route is dropped, allowing the next packet to use the newly discovered route. If this process fails, a discovery retry can be made several times with a configurable default. Route response messages (RREP) can only be sent from the destination node in this adaptation like many other adaptations in this area. To detect link failures a Link Layer Notification (LLN) is supported, yet is not switched on by default. If a data packet is undelivered, the LLN makes is so that the message is abandoned and a fresh message retried.

In the work of Gomez et al [42] NST-AODV is introduced. This builds on top of the work of AODV and TinyAODV, adding new features to the already existent protocols. The authors discuss an implementation of AODV that is designed to yield better network reliability, lower power consumption as well as lower message latency. To appraise whether their own work is successful or not Gomez et al. calculate the tradeoffs generated by this adaptation of the AODV protocol on a mesh sensor test bed.

As with TinyAODV the hop count method is used as the routing metric, dissimilarly from TinyAODV a LLN is enabled by default and packets that wish to travel an undocumented route are not discarded. As opposed to discarding and retrying such a message, the first message simply uses the path that has just been discovered. Whereas in the previously developed TinyAODV a retried packet could be resent a default of three times, allowing configurable change, with NST-AODV up to two retries are possible. Two first in first out (FIFO) queues are introduced. The first is utilised as an output queue and the second one to save packets (which discover routes) ensuring that they are not dropped akin to TinyAODV. After running test scenarios on the newly presented NST-AODV protocol the authors were able to show demonstrable in the desired areas. These were namely decreased message delivery latency, increased network reliability and reduced message transmissions in the network. However, saving this power comes at a cost of roughly sixty per cent additional memory storage over the original protocol.

In two further papers improvements on these reactive protocols are put forth, OR-RSSI [47] and TinyHop [43]. In both of these works advancements on AODV based protocols are put forward for evaluation. In particular OR-RSSI is compared with TinyAODV and TinyHop is compared to both TinyAODV and NST-AODV. In the work on OR-RSSI Huo et al. it is hoped that new improvements in Mobile Wireless Sensor Networks (MWSN) over the TinyAODV protocol are brought about. As the amount of nodes lowers in a network, the likelihood that an end-to-end path breaks increases.

Many Existent routing protocols need an end-to-end network path to exist currently for messages to be sent and recieved. The implication of this is that current Dijkstra based protocols are not sufficient. Taking this into account the authors put forward an opportunistic routing protocol for WSNs. An opportunistic probability is used (based on the Received Signal Strength Indicator (RSSI) of a sink nodes messages) as well as a Mobility vector (MV) to find the most suitable node. The node with the highest opportunistic probability is used to save and forward messages for each hop.

By running analysis on this protocol the authors hope to show improvements over the traditional minimalist TinyAODV protocol.

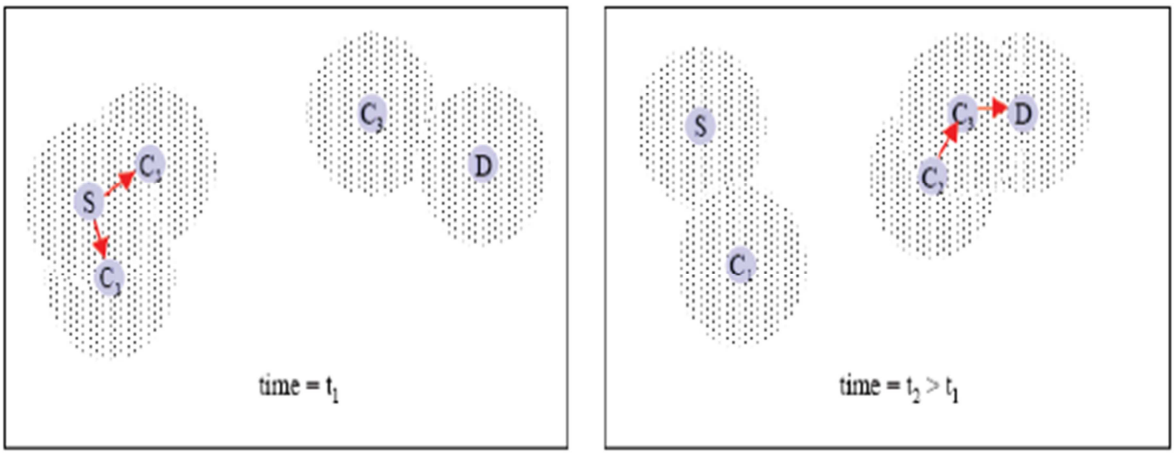

Figure 19: Opportunistic network. [47]

In figure 19 one can see an opportunistic network in practice. Initially node C_2 is not in range of destination D. Node C_2 in the network is moving towards destination D over time and at a given time t_2 where $t_2 > t_1$ gets an opportunity to deliver a message to D where it could not have done so before.

The proposed protocol is made up of:

- Finding and maintaining a nodes opportunity probability.
- Sending.
- Forwarding.

Periodic beacon messages are sent by the sink; other nodes in the network update their OP values with the RSSI information gained from the beacon message. The node with the greatest OP value that a given node can reach at that time is sent

the message. As a sink node broadcasts these messages with a higher power they can reach a further distance, encompassing more of a network. To investigate how the OR-RSSI performs versus the widely used TinyAODV the authors implement the protocol in the NS2.29 by using txinfor_.RxPr of ClassPacket to gather the necessary RSSI information [47]. It is demonstrated that in several areas the newer protocol outperforms the TinyAODV protocol. A higher Successful Delivery Ratio (SDR) is achieved along with a better goodput and a lower energy overhead.

Since OR-RSSI has a higher ability to deliver messages than with TinyAODV a path is more likely to be broken in a protocol that keeps certain paths stored for a given time as opposed to a protocol that creates routes dynamically. In these simulations OR-RSSI was shown to average an eighty per cent SDR compared to a fifty per cent average with TinyAODV.

Figure 20: SDR of OR-RSSI versus SDR of TinyAODV. [47]

As in often the case with an advantage in a new protocol, it is a trade off in another area. The time taken for messages to arrive in OR-RSSI is increased over that of in TinyAODV. This is due to TinyAODV messages always taking the shortest path to

their destination. The trade-off here is that although there is a quicker time taken for a message to arrive with TinyAODV, it does not have the high delivery success ratio of OR-RSSI. When designing protocols such as these, one must look at the environment and applicability of the protocol in question. With some applications TinyAODV may be a best fit due to the high speed delivery of any message along the shortest path an conversely in non-critical time based applications such as MWSN the higher delivery success ratio may be more desirable.

The final two areas that have been investigated in this paper are goodput and Energy Overhead. Goodput is a measure of the bits/s that the sink node receives. Yet again the application the protocols are being used for plays a large role in which is the better of the two to choose from.

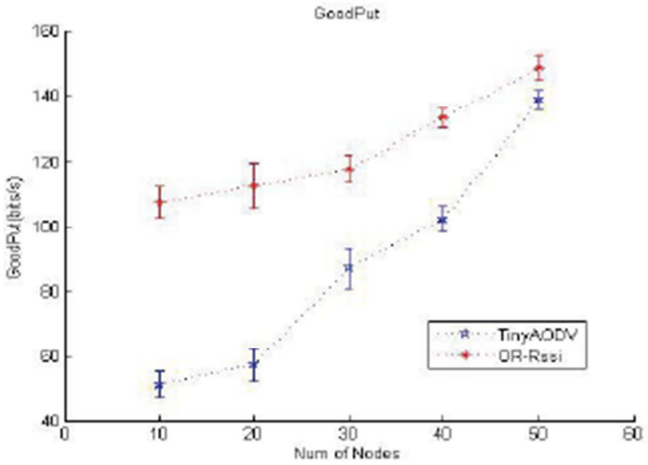

Figure 21: Goodput of OR-RSSI versus TinyAODV. [47]

In a dense network OR-RSSI is shown to have approximately the goodput as TinyAODV. However in meagre node populations OR-RSSI has a drastically higher goodput than the compared protocol. An example given by the authors is Node 10 in the diagram (Figure 21) above. The goodput shown for the OR-RSSI protocol is over

one hundred bits per second whilst with the TinyAODV it is only around fifty bits per second.

The final attribute measured is energy overhead. This is proportional to the amount of transmissions in a network. The average transmissions per successfully received message in a low density network is calculated and displayed in Figure 22 below. Due to a higher rate of unsuccessful packets (due to broken links) TinyAODV must broadcast too many routing packets.

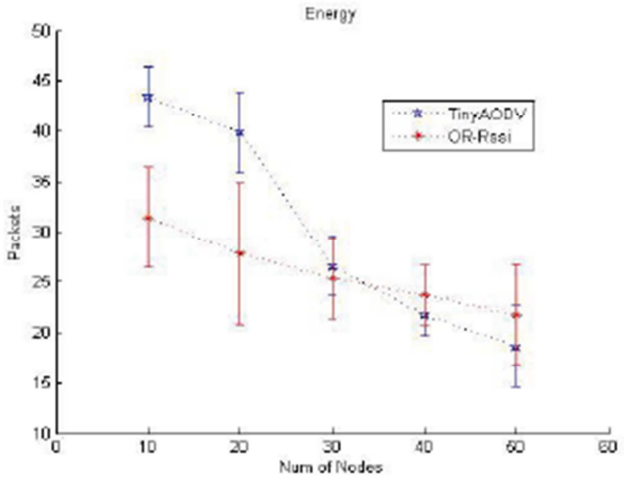

Figure 22: Energy Overhead of OR-RSSI versus TinyAODV. [47]

As the amount of nodes increases the energy overhead of TinyAODV decreases whilst that of OR-RSSI increases, until OR-RSSI eventually surpasses TinyAODV's energy overhead. This brings forth the trade-off factor which once more drives home the need for good forethought when designing a system and the protocols used. The authors make a final conclusion that in a sparse network OR-RSSI outperforms TinyAODV. However, as the results show, one can conclude that for a more densely populated network this is not the case. How these aspects affect energy efficiency and a dynamically based energy cost is an area of great interest.

The authors of An End-to-End Routing Protocol for Peer-to-Peer Communication in Wireless Sensor Networks [43] discuss further work in this area and present a new protocol called TinyHop. This is a Flat-based reactive protocol whose main hope is to reduce the amount of messages needed to execute routing in a network. Many protocols employ a periodic beacon message (including the aforementioned protocols) which consumes more power than a normal message, in the aid of improving routing. However, this is power intensive and not an ideal solution for a domain where power management is critical. [48] Clearly defined clusters in a network means protocols can specifically be configured to route in that particular cluster. TinyHop limits flooding into these areas and provides communication between any two nodes in the network that are in reach of one another. A comparison is made between the features of several of the protocols discussed here with TinyHop in Table 1 below:

Protocol	TinyAODV	NST-AODV	TinyHop
Connectivity Maintenance Mechanism	LLN (disabled)	LLN	End-to-End
RERR message	Yes	Yes	No
Local Repair	No	Yes	Yes
Only destination generates RREP	Yes	No	Yes
Routing Metric	Hop Count	Hop Count	Fastest RREQ
Precursor List	No	No	No
Implementation Status	TinyOS 1.x	TinyOS 1.x	TinyOS 2.x

Table 1: Comparison of reactive WSN protocols: TinyAODV, NST-AODV & TinyHop. [43]

Differences between the three protocols are then discussed by the authors. The first mentioned is that NST-AODV improves the TinyAODV protocol by introducing a local repair system for the reparation of routes without the need for the Discovery process that TinyAODV is dependent on. By enabling nodes to contain data about a preceding node TinyHop can avoid cycles as well as reducing the power and memory needed due to removing the need for each node to retain information on several routes.

Routes between two nodes are independently discovered and only accurate functioning paths are stored in tables. Every node in the network contains two tables. In the first is the list of available nodes for which a route already exists and in the second is the details necessary for routing. Apart from the route discovery part of the TinyHop protocol there is also the Route Traversal Phase for transmission of data over the network and the Maintenance Phase for minimising storage of information. Invalid routes as well as routes that are never used are trimmed to maintain good network performance. By measuring the frequency that a route is used and checking if a route is broke, a protocol can minimise stored data. Once the three main phases of TinyHop are explored by the authors an evaluation takes place between it and other protocols in the WSN domain.

Simulations have been carried out using TOSSIM, a TinyOS simulator for simulating MicaZ motes. The main attribute measured in these facsimiles is the round trip time (RTT) over different network layouts. Two scenarios, a high and low connectivity, are outlined to signify node connectivity.

Assessment of the RTT is shown to evaluate the effects of scaling with TinyHop with regards to local repair latency and route discovery. As is likely with a reactive protocol, TinyHop is shown to perform better in low mobility networks. The standard of link connectivity is examined, showing that this affects the reliability of the connection and the performance of the protocol. If the network is of higher mobility the protocol is revealed to need the execution of more stages to deliver a message. Further experiments are carried out to investigate possible overhead associated with the protocol. If several paths are generated simultaneously (more nodes) it is intuitive that the overhead will increase.

Stressed by the authors is the fact that overhead is greatly increased as the number of nodes in the network is raised, due to scalability and link connectivity. However, this overhead is sought to be below that of other protocols, whilst still

having reliable message delivery. In conclusion the authors present a viable alternative to other reactive protocols discussed, TinyAODV, NST-AODV etc. Keeping end-to-end communication in a network is a valuable task and further research into optimal ways to do this has been added. Further work on this protocol is mentioned as extension to include wider routing metrics such as power consumption, which ties in with the research carried out in this dissertation. A necessary area to look into is how easily and valuably it would be to add a form of power consumption metric for routing into the discussed protocols.

4.16. Hybrid Routing

Hybrid routing protocols are protocols that combine proactive and reactive routing protocols. Distance-vectors are used to find more exact metrics to establish the best routing paths. Change in the topology of the network is reported. This method can bend its routing strategies depending on the requirements of a network. However the network data is sometimes difficult to acquire and store. It is tough to generate routing methods dynamically due to this difficulty. Hybrid routing protocol examples are Zone Routing Protocol and Zone Based Routing.

4.17. Power Efficient & Cost Based Protocols

An essential part of this research is learning about power efficiency, how and why one can reduce power consumption along with how can one route in a more advantageous way if various power related scenarios arise. Several of these scenarios may include nodes failing or having a low amount of power available to them. To minimise energy consumption many routing techniques have been researched and

brought into practice employing special attributes. This is one of the most, if not the most important research areas for WSNs at the moment. Unnecessary power consumption and dealing with any power related routing or efficiency issue is vital in advancing the WSN technology. By offering features such as changing the role of nodes, clustering and in this case, operation cost/price allocation it is hoped that a more wealthily featured WSN technology can come about.

In the work of Schurgers et al [49] the idea that perfect energy efficient routing is infeasible is put forward. Alternative solutions are put forth and discussed. It is noted that these merely scratch the surface of what is a wide and complex problem domain and edge cases where their ideas may have to be bended are included. If a certain node failing is paramount to complete network failure due to its duties an equal resource allocation, such as the one proposed, cannot be used. Algorithms motivated by this research are also discussed and why the authors feel that their proposition is a solution to the energy problem.

The initial idea explored is one of spreading network traffic in a way to prolong life of the WSN. Research here has found that when one is looking for the shortest path in a network quite often some nodes are much more used than others. These motes die much quicker than their counterparts and thus decrease the lifespan of the network. In Figure 23 below, an example energy histogram is shown to demonstrate how some nodes are much less used in practice.

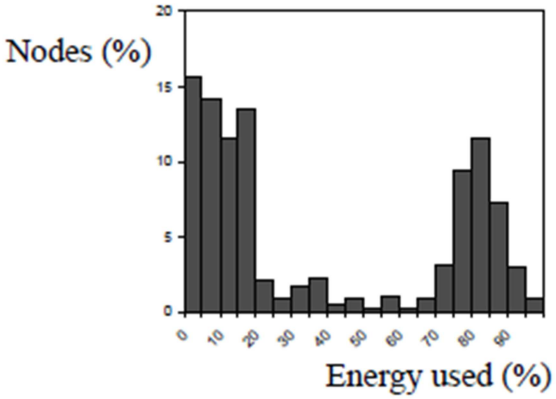

Figure 23: Energy distribution that should be avoided. [49]

By shaping the traffic to give a more even distribution (Figure 24) the authors feel that they can provide a good method for prolonging WSN lifetime.

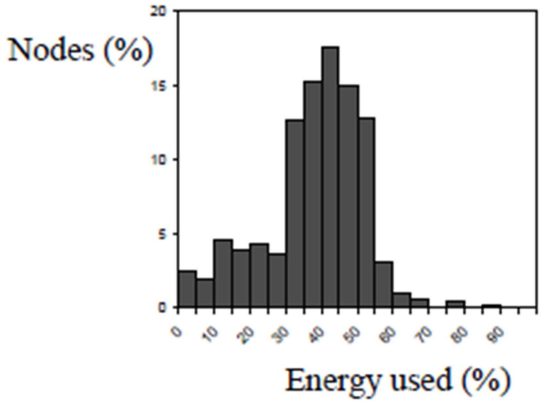

Figure 24: Energy distribution that should be aimed for. [49]

Once this general concept has been established several algorithms (based on directed diffusion) are delved into with the comparison of results in mind. Gradient-Based Routing (GBR) is introduced. The end user asks the system for information and a flooding technique is used to generate a measure of how suitable the possible subsequent hops in a system are. As a message propagates through the system it measures the amount of nodes used, the path with the least is the shortest route.

Comparing this value with neighbours yields gradient values which are used to guide packets in the network. Next data combining Entities is introduced. This is where each mote processed its own sensor data and then passes it along. Inspection of other motes can generate better energy efficiency. Further functionality is detailed along with simulations showing that low energy is used with this scheme.

Following on from this some traffic spreading schemes are outlined. The first employs a method of randomly selecting nodes with the same gradient to spread traffic more evenly, whilst the second employs an energy based rule set. In this scheme a node acts upon detection that its energy has dropped under a certain level. It changes its gradient making it more likely that routing will occur on other nodes and not on itself. In the final algorithm a stream based system is implemented. When a message is received by a mote it informs its neighbours (except from the one that sent the message) that its gradient is changing. As the gradient of more used paths are changing, lesser used paths will be operated on incoming messages. Further simulations on these methods show better energy distributions on nodes in the network. In conclusion it is noted that although these solutions tackle the energy problem, much more work has to be done in this area. The work carried out here should be able to be used in further research, perhaps in aid of other techniques to route efficiently. Similar to the energy based routing used is the work carried out by Chang et al [50] where the cost based path is chosen by measuring the energy for transmitting and receiving nodes. Lowest values form a path similar to a gradient over which messages should travel.

In the work of Singh et al [51] five parameters, apart from some of the more traditional metrics such as counting hops, are put forward for generating routes in WSNs based on power consumption and cost. Cost per packet is shown to significantly be reduced by applying these metrics and an overall lengthier network life is maintained. An important concept pressed here that cost based routing is superior to shortest path based routing for a more efficient system. This is stressed via simulations

carried out between more standard metrics and the newer ones presented. The first metric introduced is that of minimising energy consumed per packet. Intuitively obvious is that tampering with this to reduce power consumed is advantageous, however also put forth are some drawbacks of this. At any time a route that is chosen may not be the shortest route available. This can lead to load balancing issues and an inconstant message energy cost. The algorithm routes traffic around loaded areas and therefore some inconsistencies arise in power usage, ultimately leading to inconsistent node life span.

The second metric laid out involves maximising time for partitioning a network. The premise consists of finding a nominal collection of motes that, upon removal, will cause partition of the network to take place. This set of nodes is critical in communication between parts of the network and are therefore more important than other nodes in the network. By providing a method to increase the length of life in these motes, one can extend the life span of the network. This is not enough to assure that each of these critical nodes uses power equally however. One of these may fail much more quickly than one of their counterparts. This causes a hindrance to network latency so it is best to ensure that these nodes consume power uniformly. Due to the routing algorithm set up here this is only possible if messages are of equal size, if this is not the case a problem is highlighted.

A further metric that may be used is minimising the power variance in motes. Motivation here is that all motes remain running together, with no failures. To implement this functionality every node sends messages through the neighbour with the smallest transmission queue. Furthermore, by expanding the field of view to encompass nodes several hops away, messages can be sent along the path with lowest queues easily. Once more, if messages are of equivalent length then power consumption can be equally distributed. Further variations on these themes are then measured, such as minimum cost per packet and minimising the maximum node cost. The first of these metrics states that if the remaining battery power is quite different

from other motes then routes should not use the lower power paths. A cost is associated with each packet depending on the willingness of a mote to participate. The final metric suggests that although motes may have a minimal cost per packet, not all do so. Various motes may have a huge cost and the final metric prevents this disparagement.

Controlled simulations are then carried out and it is put forward, based on these, that using such metrics is advantageous. Power cost is reduced in various areas and it is mentioned that the larger the network, the more the savings that can be made. Finally mentioned is the fact that if nodes move erratically then a lot of metrics mentioned do not yield significant power savings. Further work is carried out in this area in the work of Woo et al. [52] where the Local Energy Aware Routing (LEAR) is discussed. This is based on the previously mentioned DSR algorithm. Here remaining battery level is the determining factor for whether Motes take part in routing or not. More important tasks are put ahead of those of lesser value, such as routing. When power is low a mote may refuse messages and perhaps, just sense its surroundings. If one of these nodes happens to be a critical node for partitioning of a network then the value at which it stops taking part is revaluated upon retransmissions and messages can be passed.

In the work of Ye et al. [53] a proposal for a new medium access control (MAC) protocol called S-MAC is detailed. The premise that this is based on is that nodes sleep and sometimes need to awaken for participatory action. Traditional MAC protocols do not support the necessary features needed for a WSN (Energy preservation etc.). Shown in Figure 25 are the three main components of S-MAC.

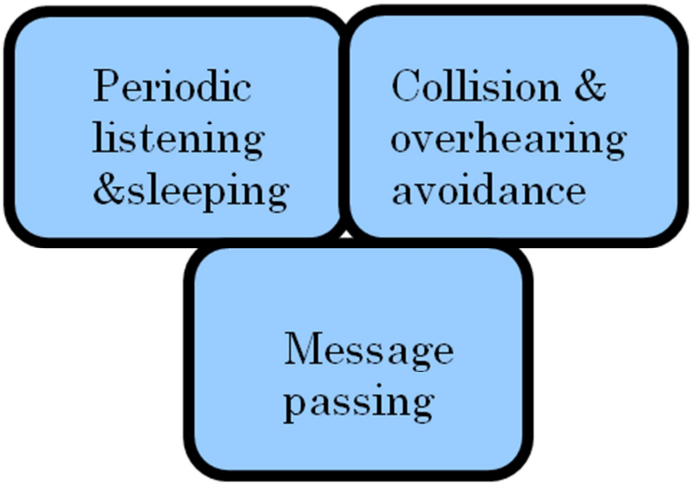

Figure 25: Main components of S-MAC protocol. [54]

- Nodes periodically sleep.
- Syncing with neighbouring is desired.
    - Auto-sync process.
- neighbours of communicating pair sleep on receiving RTS/CTS
- Messages passed lowers latency
- Energy is saved over IEEE Standard.

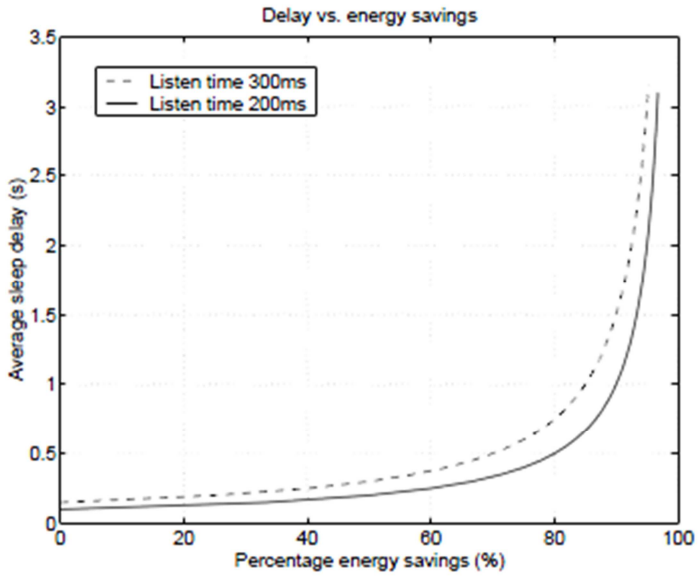

Figure 26: Energy savings versus average sleep delay. [53]

The SMAC protocol not only saves energy over traditional approaches (Figure 26) but also allows for the trade-off between wanting better energy conservation or better latency in the WSN. In the work of Van Dam et al. [55] further advancements over this protocol are made in the T-MAC protocol, as low message rates in S-MAC cause inefficiency. This is due to the fixed active time employed, which is replaced by an adaptive cycle in T-MAC, generating further energy savings.

In conclusion, much research has been carried out in the area of WSN routing protocols and by focusing on the domain of improving the energy profile of these networks by adjusting and creating new algorithms the technology will be greatly advanced. Many trade-offs occur when adjusting the aspects of a WSN and it becomes a challenge of increasing energy efficiency whilst maintaining as good a protocol as competitors in other aspects. A dynamic cost based protocol ranking system for routing in WSN is a valuable research avenue and by examining each aspect of previous protocols, pitfalls and ideas can be transferred into any design put forward.

4.18. Security Considerations in WSNs

As stated previously, with it being the basis of this project, an important factor in a wireless sensor network is the battery life of each node. To provide seamless integration of wireless sensor network (WSN) nodes into our environment there are generally some agreed upon constraints which will be reiterated.

- Consumption of less than 100µW of average power.
- Cost less than one American dollar.
- Have a size of less than $1cm^3$.

These constraints place an added burden on the development of and security solutions for WSN over their more powerful wired counterparts. In particular the following should be considered in developing a secure system: Power issues, adaption to change, mote concentration, dependability of motes, scalability, self-healing and self-configuration etc. Security considerations for wireless sensor networks are vast as the application area is vast, encompassing a large array of possible application domains. Processing taking place in the network makes end-to-end security more difficult due to the transitional nodes need for direct access to the contents of the packets. [56]

Distributed wireless sensor networks are spreading rapidly and this rate is suspected to increase within the coming years. This is causing concern for those developing wireless sensor network protocols with regards to security, these networks have very different demands from wired alternatives and thus far nearly all WSN protocols have been developed and deployed without any thought for inclusion of security. They operate on an unreliable communication channel and nodes are often unattended for long periods of time. The nature of a wireless packet-based routing system is innately unreliable. Packets can often be lost due to issues such as congestion and because of this good error handling, as well as how to handle the loss

of security packets, is an issue. Due to long periods of nodes being unattended, possible physical attacks on the nodes themselves must be thought about. Attackers may gain physical access to the device they wish to tamper with, which makes it easier for them to manipulate. Two approaches can be used here, either the system is developed in such a way that each node has some form of protection for critical data (key data); meaning attackers cannot receive information from any captured node. In the second scenario securing the data on each node is not as much a concern, damage limitation is the priority. Although an attacker may gain information by infiltrating the node this data is to be limited and only a certain set of defined knowledge is gathered in a form of 'limited gain'. Depending on the application at hand either method may be used, with higher security sensitive applications, such as military, the higher cost keyed data system is justified. [57]

Four main areas of discussion arise from exploring security considerations for WSN: the requirements of the WSN itself, the possible impediment to security concerns, attacks on the network and finally defences for these attacks. Many types of attacks can occur (along with multiple ways to deal with them) and so flexibility and resiliency are an issue. In an outside attack [58] [59] motes foreign to the WSN try to attack the network and in an inside attack [58] happens when motes inside the network start to perform in unplanned and illegal ways. A WSN must deal with the possibility of both and have a strategy in plan to deal with any such corrupt nodes [58]. Attacks can also be in the form of active and passive. The former include techniques such as eavesdropping and monitoring messages between nodes on the network. By nature active is more intrusive, possibly modifying data in transit or creating their own set of messages. In seemingly harmless applications, such as those that gather information on heat and light in a company, malicious information can be generated. Patterns can be extrapolated from data and used for malevolent intent, such as attack on a company. This is one of the reasons why even passive forms of attacks can be quite detrimental.

Following on from this are the possible methods and capabilities of the devices carrying out the attacks. In one method the opposing party uses motes similar to those in the WSN. Another possible situation is that the device being used is much more powerful than the motes in the network. This could come in the form of a desktop or a more mobile laptop. The latter devices pose a significant threat due to the nature of their increased processing capabilities, unlimited power and increased transmission range. [60] [61] Particular forms of attacks can now be specified such as Denial of Service, traffic analysis, Sybil and Node Replication.[62] The list goes on with a possibility for a multitude of possible attacks to be performed on ill designed systems and even those that are relatively well designed.

Next we look to defence measures for these security issues. Most importantly we have the use of keys. Key establishment and sharing are the main issues here, however keys are the foundation for a lot of security measures and so it is worthy to investigate these thoroughly. Many of the same topics that are important in this area are important for the possible issues discussed above. A network must prevent corruption or at least mitigate any losses. Issues here include secure transmission, Denial of Service (DoS) attacks, prevention of traffic analysis and management of trust etc. Primarily of concern is the aforementioned key management in WSNs. In many forms of networks today public key algorithms are used for key establishment but these are primarily unsuitable in the WSN domain which requires the use of low power, highly constrained components. These are unsuitable due to the fact that public/private asymmetric cryptography is computationally expensive in most cases. This leaves us with the less expensive symmetric cryptography (block ciphers include MISTY1 and KASUMI etc.) which is where a key is shared between two nodes transmitting and receiving data.

Another concern here is the distribution of keys, the key exchange. Pre-distribution is not always possible so this is a complex challenge to face. Beyond these specific common attacks such as DoS attacks can be looked into. Possible workarounds for this particular attack may be finding out where the attack is taking place and routing around it as well as methods to provide for redundancy and replication. As with attacks, the solutions and defences too are vast and must be carefully be considered before implementing them.

Security of a system should not be viewed as a separate constituent of a system; it should be incorporated into each step. This is why current protocols are lacking, security must be thought about in the original design of the routing algorithms and not an afterthought. All facets must be scrutinised, from routing to the formation of node groups and aggregation of data. Early research held assumptions such as all nodes were always trustworthy whereas this is not the case. One must be careful when considering how to trust and share information over a network that needs to be able to scale to potentially hundreds of thousands of nodes. Trust must be established, along with integrity of messages. Due to computational constraints (along with limited storage and power) a myriad of security issues become more of an issue, such as key generation and sharing a complex concern.

5. Dissemination

To scatter or to spread widely, disseminating in WSN's is a method to broadcast a message from a node without expecting acknowledgment or feedback. From observations of wireless sensor networks it can be seen that the jobs performed by nodes in a network vary over periods of time. This might include sensing frequency, duty cycles, what is being sensed etc. It is important to have a mechanism to not just update one node with some piece of new code but to also make sure that all nodes in the network have the same view on data and the goals that are needed to be performed.

Increasingly the notion of node mobility is being used in WSNs with motes being able to dynamically leave one network, join another and possibly return to the original in a myriad of different scenarios. With the mobility of these nodes also comes the possibility of mobility of the tasks performed by the individual node. In one network a node might be measuring temperature and vibration, however as this node moves from one network to the next its job may vary to now sense many other conditions such as carbon emissions and so forth. It now needs to be reprogrammed to have the code and parameters necessary for this new task.

If some form of ‘reprogramming period’ is employed this yields a high probability that some nodes will not have correct and up to date code on board, with either a node moving into the network dynamically from another network or perhaps a node that lost connection to the network re-establishing it, only to have an older code version. This means it is essential that there be some provisioning done so that there is a method of keeping track and making sure that all nodes in a network have an up to

date version of code and this is where disseminations protocols are becoming more useful and popular.

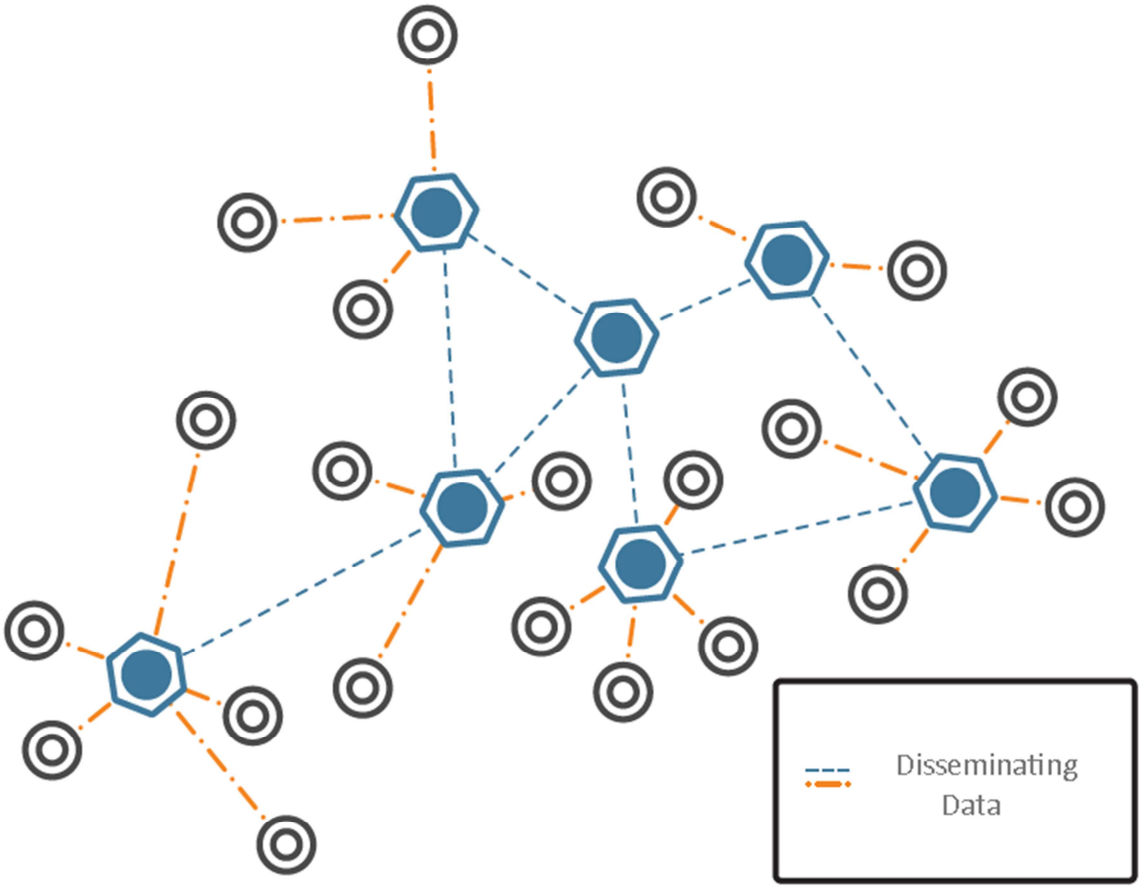

Figure 27: Dissemination of data throughout the various nodes in a WSN.

There is a demand for better and more fully equipped dissemination protocols out there and a developer must have the ability to configure the network via these, send binary images and code containers at a minimum cost. Sending data to every node reliably is an inherently expensive process and so it is best to minimise this as best possible, reducing energy used whilst keeping a high convergence speed and the ability to scale adequately with the density of the network and the amount of data items being introduced. The way in which this is done is via dissemination protocols.

Individual motes figure out when their data differs from that of the rest of the network and acts accordingly.

The aim of disseminating is to deliver an iota of data in a reliable fashion to each node in the network in such a way that eventual consistency is established, whilst getting nodes to ask clever questions at appropriate times to figure out if an update is needed at a minimum usage of energy. Such protocols transmit data reliably using a key and version tuples. A value is sent by the transmitting node with each node in the network keeping a copy of this data. Once received by the appropriate nodes data is processed, possibly returning information if required.

The protocol informs the sensors of when a value has been changed and messages are exchanged until consistency is reached. A node can inform the network that a piece of data it has received is newer than the view currently held. It is possible for nodes to hold a different view of the newest data but with each exchange the amount of such occurrences is lessening until a global view of the newest value is established. Dissimilarly to previous flooding protocols which may not have reached consistency, dissemination ensures a value will be agreed upon with every node, even if multiple nodes try to update the network simultaneously, presuming each node remains connected.

Many applications find the ability to disseminate small values to each node in the network invaluable and it is a handy tool upon which many WSN applications are based. Power consumption being a big factor in WSNs has led to the creation of multiple protocols hoping to improve performance over one another. A developer merely has to introduce their software or commands to the network and can be ensured that all nodes will now have the needed parameters, configuration data and programs. The purpose of the sent message can be anything from simple value changes to advertisements and public announcements. These allow the WSN to be queried, configured and programmed. WSNs are often subject to high packet loss, disconnections and so dissemination protocols play an important part in making sure

nodes receive up to date data reliably. If a node does not receive a packet or is disconnected from a network it will receive the newest value upon re-entering the network reliably. Possible values that it missed are not received but also are not needed due to being irrelevant to the network at such a stage.

The choice of dissemination protocol can vary a large amount depending on the size of values and such data fragments. Data sent in the order of bytes and kilobytes will have different purpose, from configuration to binary data (control vs. data payload). These must be handled in the most efficient and appropriate way.

5.1. Dissemination in TinyOS

Included in the TinyOS 2.x installation are three dissemination protocols, Drip, DIP and DHV. These protocols can deal with the dissemination of small data, in the order of it fitting the data into a single packet. To disseminate larger values one would need alternative interfaces and functionality than those provided. If a user wishes for only certain nodes to apply a new value they can form a message with a condition as well as the data value. Power consumption and efficiency plays a large role in WSNs and again in the design of any protocols to be used on them. This must be improved but also delivery of data and the scaling of controlling traffic.

An in depth analysis of these three protocols, their mechanisms for dissemination and how this affects power consumption is a valuable avenue of research to explore, particularly as these dissemination protocols are the standard set that are rolled out with each TinyOS installation and implementation.

There are two interfaces provided for dissemination in TinyOS, DisseminationValue and DisseminationUpdate. DisseminationValue is for the receivers of a value that is to be disseminated and disseminationUpdate is for the nodes that are generating the values for dissemination.

DisseminationUpdate:

```
interface DisseminationUpdate<t>
{
 command void change(t* newVal);
}
```

DisseminationValue:

```
interface DisseminationValue<t>
 {
 command const t* get();
 event void changed();
}
```

In the former, the command *DisseminationUpdate.change()* needs to be called each time a producing node needs to disseminate data, the one parameter that needs to be passed is the new data to be disseminated. The receiving nodes make use of DisseminationValue. When they receive new data via dissemination that needs to be changed *DisseminationValue.changed()* is called. This new value can be obtained by calling the get command.

Each of these interfaces uses the fact that space is allocated (by the dissemination service) for disseminated values to allow for multiple components accessing and using the same data via a pointer to the *DissemnationValue.get()* command. Although DisseminationValue has a set command this must be used sparingly, to initialise variables, as changing values via the set command may lead to inconsistency of the view of a value across the network if called after the change command.

In cases where a node has been inactive and it calls the change command to try and disseminate an older value throughout the network, this is ignored and the newer values maintained via inspection of the packets. Each protocol must take into account that as multiple nodes may be disseminating different values at the same time, that there must be some form of method in place to decide which value should be the one to be disseminated. This needs to be done to the extent that one value is seen as the value that should be used by each node, when multiple new values are available, to ensure eventual consistency. [63]

Maintenance cost and detection latency are the crucial factors in efficiency and performance trade-offs in dissemination protocols. The former is the amount and rate of data sent when a WSN is neoteric and does not require data to be currently updating. It is the cost associated with making sure that the network is still up-to-date. By the nature of these two attributes it can be intuitively seen that they are conjugated, tampering with one plays a role in how the other responds.

Lower latency means a bigger maintenance cost and a larger transmission interim means less packets sent but a larger latency in checking if the network needs an update. An algorithm called the Trickle Algorithm deals with lessening the impact of this coupling by dynamically allocating the size of the interval instead of having it hardcoded into a protocol. When a network is unstable it is necessary for more information to be passed and conversely when the network is experiencing a high level of stability fewer packets need to be sent and increasing the interval size becomes

more attractive. Trickle introduces a method of faster dissemination and better efficiency whilst not dealing with how the need for an update is detected.

5.2. The Trickle Algorithm

This algorithm is an algorithm that is used for the propagation of code and maintenance in Wireless Sensor Networks; it regulates itself [64]. Trickle brings nodes across a network to an eventual consistent state and does so in an efficient and rapid manner whilst adjusting for the density and topology of the network. Originally created with the aim of dissemination of code in mine the trickle algorithm was shown to have a greater use, finding neighbours, maintaining routes and so forth. A crucial factor in WSNS is efficiency and performance trade-offs with energy, little power must be used and these kept high.

Trickle aims to reduce energy used in a network whilst still keeping a networks performance and efficiency at the same high level. Each of the Dissemination protocols bundled with TinyOS (and dissemination protocols in general) uses or augments this algorithm. A form of “polite gossip” is used to control send rates; the main functionality is borrowed from gossip and scalable multicasting. The idea here is that a node sends data to its neighbours in a regular time period and the receiving node can then compare the received data to its own using version numbers associated with each data transmission to tell if data is newer than the current data held.

To improve upon efficiency, if a node has heard a broadcast that is the same as its own then it does not need to broadcast its own summary. However if the data differs the node then broadcasts an update message. As previously outlined, maintenance cost and detection latency are critical factors that perpetuate all WSNs.

The Trickle algorithm lessens the impact caused by this coupling. To achieve this it dynamically controls the send rate of data messages in the network. Trickle detects instability or stability in the network and changes the broadcast interval suitably, exponentially increasing it if the network is stable and reducing it if not. This limits the amount of energy expended in the network, making sure less unnecessary packets are broadcast**.** Alternative approaches such as simply flooding the network with packets are not as efficient or desirable; with the Trickle algorithm the rate of sending messages is strictly controlled, allowing nodes to save energy by only sending and receiving the trickle of packets necessary to stay up to date.

An example of how this may work is the developer introducing new data at just one node. The version number of this data is incremented since the last introduced value and this node then broadcasts to inform other nodes that its information is newest. The sending rate has now been dynamically changed, minimised to a small value so that the node broadcasts that it has new data frequently. The nearby nodes then sense that their data is out of date and inform the original broadcaster. This node then sends the data and the new code is propagated throughout the network, repeating these steps until each node has the new code. The network is now stable and assuming no further new data is introduced the broadcast interval starts to dynamically increase until it reaches its maximum value.

Each and every node hears the minimal amount of transmissions needed and in this way the algorithm is shown to scale incredibly well(in a logarithmic fashion) regarding network density, up to thousands of changes, with a maintenance cost of merely a few sends an hour per node (costing merely a few bytes). New data retains its speedy propagation with new values taking mere seconds to traverse the network.

5.3. Dissemination Protocol (DIP)

The first of the three data discovery and dissemination protocols bundled with TinyOS to be discussed is Dissemination Protocol (DIP). Similar to each of the other two protocols DIP uses a trickle timer underneath. Values to be disseminated in DIP are treated as a group which yields all dissemination control and any parameters applied being used for each value being unified. Advertisements can be varied based on size of messages via DIP.h in the TinyOS installation.

The DIP protocol developed by Levis et al [65] aims to improve on previous approaches such as SPIN and Trickle. These protocols are seen to have large overheads due to the linear scaling incurred by increasing amounts of data to be disseminated. The scaling factor for these dissemination protocols was in the order of with O(T) for T total data messages. Trade-offs occurred here between the tolerated latency and the data cost. Motes were seen to have good latency with the forwarding of O(T) data items or have a flat line cost with an O(T) latency.

DIP improves on this in a manner such that for T packets O(log(T)) values can be found whilst achieving a O(1) latency. To achieve this DIP dynamically chooses between two scanning algorithms, randomised scanning and directed tree searches. Having this flexibility allows DIP to better previous protocols in both transmitting data (by sending up to sixty per cent fewer packets) and the speed of dissemination (by up to two hundred per cent). This incurs an extra cost of O(log(log(T))) additional state bits per data value for T values. [65]

DIP introduces the idea that several pieces of data can be amalgamated into one advertisement. In this way information is compressed and it is sometimes impossible to know what needs to be updated, merely that an update must be completed. Due to this a new algorithm called Search is introduced in DIP. This algorithm uses a hash tree of the data versions to find out if an update is needed. Dissemination protocols used one of two approaches to apply a trickle interval before the introduction of DIP. These were serial and parallel scanning. In the former a single Trickle counter is used to serially scan version numbers of disseminated data for advertising purposes whilst the latter uses many parallel Trickle timers. With this new protocol Levis et al. introduced a third method, namely the aforementioned hash tree method of receiving the maintenance cost and detection latency perpetually (O(log(T)) extra overhead upon update detection required).

Efficiency is improved by using hashes over key space ranges to determine there are differing versions and then a bloom filter is applied to locate the differing values. Theoretically this works well, however real life networks tend to yield some problems with the Search algorithm over a less realistic model. A big issue is packet loss; this can make navigation of the tree difficult. The second issue stems from this packet loss issue, inordinate advertisements are generated when packets are lost. This is where the dynamic nature of the DIP protocol comes into play.

To further provide clever and appropriate sending times for messages DIP generates a constant appraisal of whether or not data needs to be updated or not. DIP stores this value and it is applied to the relevant neighbour. It retains this as a percentage and it is facilitated through message exchanges. DIP can proceed if this appraisal is inaccurate or unavailable however, it simply aids in the dissemination process. This is unlike version numbers; these have to be completely reliable and stable. Upon reaching a hundred per cent probability that data needs to be changed in

the network DIP transfers the data. For a hash of x items, if a difference is found then the protocol knows that at least one value in the hash is not the same as its own, each item is given a value of 1/x. For larger hashes this gives a less accurate estimate than with smaller hashes, as the value for x increases. Lowering the size of hashes is one way to advance probabilistic precision.

Through extensive testing Levis et al. determined that in cases of high packet loss or in cases of large amounts of nodes needing an update a randomised solution was more efficient than the tree search alternative. The analysis shows that three factors must be taken into account when switching between the two algorithms, not only the rate of packet loss and the amount of nodes that need updating but also the density of the network. Through the use of these criteria one can see that the hybrid method employed yields more efficient dissemination.

5.4. Drip

The second of the three dissemination protocols bundled with TinyOS to be discussed is Drip. It is the earliest of the three to be written and implemented, DIP is built upon it; it is therefore a much simpler protocol. It was written to better the previous Bcast and was introduced in a very early version of TinyOS. Some of the main features of this protocol are outlined here. Reliable injection is afforded, with the ability to inject data to be bi-directionally communicated, between the host and the network. This host is aware of if and when a data item is introduced and injected, allowing for the detection and repression of radio failures at an early stage.

When a mote takes part in the transmission of new code in the network the surrounding notes delay their own broadcasting, this ensures a lowering of the amount

of messages sent with less unnecessary data being transferred. Version numbers are handled in the Drip header of passed Drip messages, ensuring no identifier collisions.

Previous to this version numbers were stored in the highest level header of the sent message, allowing possible collisions of identifiers to occur. Similar to each of the other two protocols Drip uses a trickle timer underneath, however each value to be disseminated in Drip is treated as an independent singleton. This means that data which needs to be disseminated can be independently advertised and then sent. Parameters and so forth are not shared which means data sets do not have to be agreed beforehand.

The Trickle timers can be configured through the DisseminationTimerP module in the Drip implementation provided with the latest TinyOS. The rates configured here will be applicable to all data sent. The tau values apply to all dissemination items. Every instance of a node retransmitting leads to the doubling of the wait period until it retransmits again. This is to make sure that every node will receive data but will do so in an efficient manner, lowering the rate of traffic.

However, since Trickle uses a timer for each data item this causes a much larger overhead due to headers and parameters being sent with each one. Much of this is wasted as it is merely the same data being propagated around the network unnecessarily. A large amount of flexibility and intricate control is granted due to this, regarding the speed at which one might want values disseminated. A fixed maximum latency is used for keeping the network up to date; this is not as desirable as the dynamic approach that Dip introduced, due to unnecessary large intervals in different scenarios. As more and more items are introduced into the network to be updated the broadcast rate of Drip increases with a factor of O(T) with T being the total number of data items; linear scaling.

As more nodes are introduced to networks and the size of networks increase in both motes and complexity this may become an obstacle to using drip, as the results analysed in this dissertation also compound. Linear scaling is becoming a hindrance with this in mind, bigger networks result in developers having to put a preference on either efficiency or alacrity. With this in mind, although Drip is capable of maintaining both a good speed of propagation and good performance, it and other similar linear scaling protocols are shown to be not ideal.

With the ever spreading and heightening of interest with this technology larger networks are emerging leaving Drip as an unmanageable protocol in these scenarios. On smaller scale applications however Drip can be seen to be more efficient than some of the other more advanced protocols, saving on energy and messages transmitted due to their more advanced and costly algorithms and scanning techniques, leaving Drip room to be a contender to be used in some edge cases and topologies.

5.5. DHV

The third and final Dissemination protocol provided in the core networking area of the TinyOS installation is called DHV. This is the newest developed dissemination protocol that is included, being presented as a protocol in 2009 and added to the TinyOS installation as of April of this year. This protocol is based on the consideration that when new code is being propagated around a network, the older and newer codes version numbers only vary in several least significant bits of their binary representation. [66] By virtue of this observation DHV provides the functionality to a node of selecting much less data bits and transmitting these throughout a network when wishing to see if the network is up to date or needs an update.

Version numbers are seen as arrays of bits in DHV, each version has a representation as a two dimensional array with bit slicing being utilised in the identification phase to help reduce amount of bits transferred around the WSN. By selecting only the several necessary bits, instead of the whole set, to transmit DHV provides higher efficiency in the way data is passed and propagated around the network. In the work of Dang et al [66] it is put forward that DHV can detect and find differences in versions in O(1) messages and latency. DHV has been tested via simulation and test bed implementation and the scaling nature significantly improves upon other logarithmic scaling of many other protocols. It is claimed that DHV can lower the amount of messages sent, the convergence time and the bits transmitted over other protocols.

The work of Dang et al primarily focus on comparing DHV with DIP, the newest dissemination protocol bundled with TinyOS before the introduction of DHV. Similarly to DIP, DHV employs the use of advertisement messages and groups of data messages with a single Trickle timer. This provides the same efficiency boost as it does with DIP, compressing data sent by not having unnecessary values sent multiple times. As well as this the cost for transmitting a byte is much lower than the cost of turning on and off a radio transceiver. The radio is seen to be a huge resource cost in motes and it is turned off whenever possible, again supporting the use of one larger group message over multiple smaller messages. However datasets must be agreed upon beforehand, unlike with a simpler method with a Trickle value being set for every data item, like Drip.

Advertisements can be varied based on size of messages via DHV.h in the TinyOS installation.

As mentioned above, DHV aims to reduce energy costs and transmissions by use of the observation that it is unneeded to broadcast the whole version numbers of data groups around the network, as this yields many unnecessary bits being transferred. This is based on the assumption that every time data is updated and sent

that its version number is incremented by one. Quite a reasonable assumption, both DIP and Drip employ this, but one must be wary of other protocols not heeding this convention and thus breaking DHV in its current format. The title of the protocol comes from the three steps utilised in it for disseminating code. These are namely: Difference detection, Horizontal search and Vertical search, in that order. The cost associated with each step is relatively small, incurring a cost of only O(1) total number of data messages and latency with regard to the whole amount of items in the network.

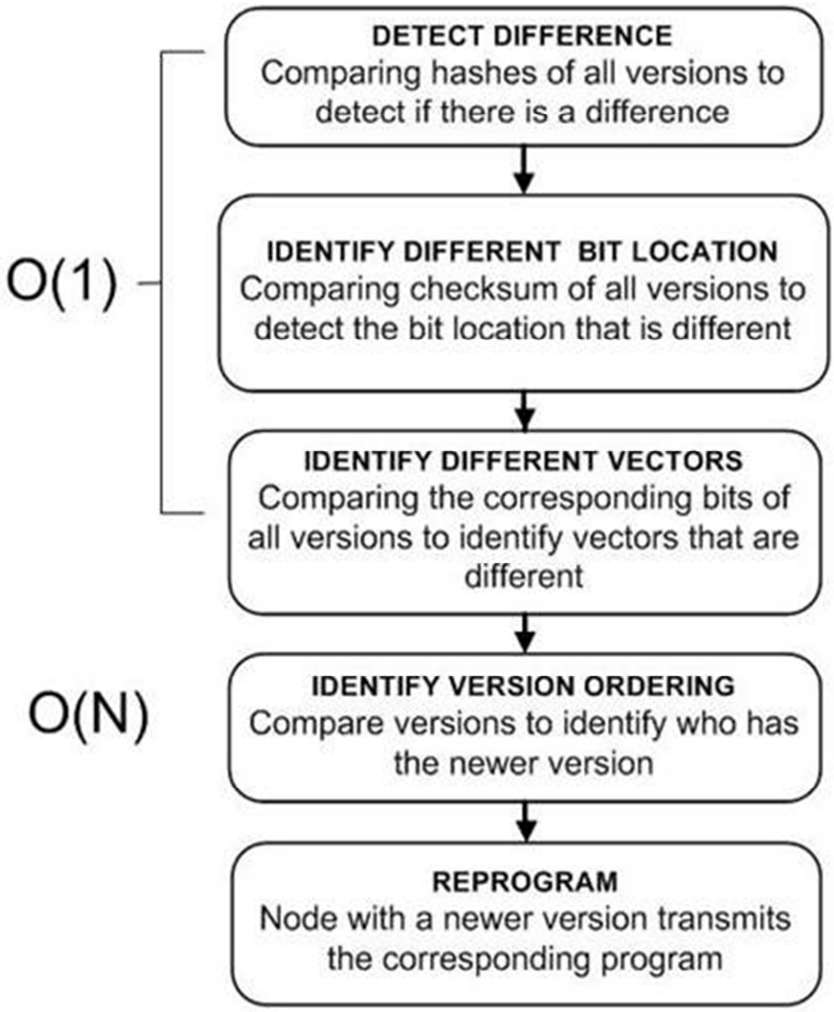

Figure 28: The important steps in the DHV Detection and Identification phases. [66]

In Figure 28 the stages of the three phases to detect and identify differences are outlined. Of the three aforementioned phases there are two important parts. The first is the detection part of the algorithm. This establishes that there is a change needed in the network, a code value is not up to date and therefore some broadcasting must be done. The latter two parts are in the identification process, Horizontal search and Vertical search. These enable the difference to be found and updated once the detection section has passed on that an update is indeed necessary. In the detection phase every node sends a hash of its versions, this is called a summary message. When a node receives this code summary it then compares the original hash of version numbers to its own hash. Upon finding a differing hash the receiving node knows that there is at least one value in the hash with an old version number.
Once there is now at least one value to be updated the protocol can move onto the second phase, identification of which items are the ones that need to be updated.

Firstly Horizontal search is engaged. In this a node sends a checksum of all versions (called a HSUM). A node that receives this HSUM compares it to its own HSUM and in doing this figures out which bit indices differ between both HSUM's.

Once this has been concluded the third and final stage of the protocol is initialised and run. This is the vertical search phase in which a node sends a VBIT message, bits (a bit slice) starting at the least significant bit of all versions. In doing this the vertical search mechanism can check if bit indices are the same between two nodes, as well as their hashes. If the bit slices are similar and hashes differ then the node starts at index 0 and continues broadcasting and incrementing the bit index until both hashes are equivalent. When a node receives a VBIT message it uses the comparison of its own VBIT to discover where the dissimilar (key, version) tuples are.

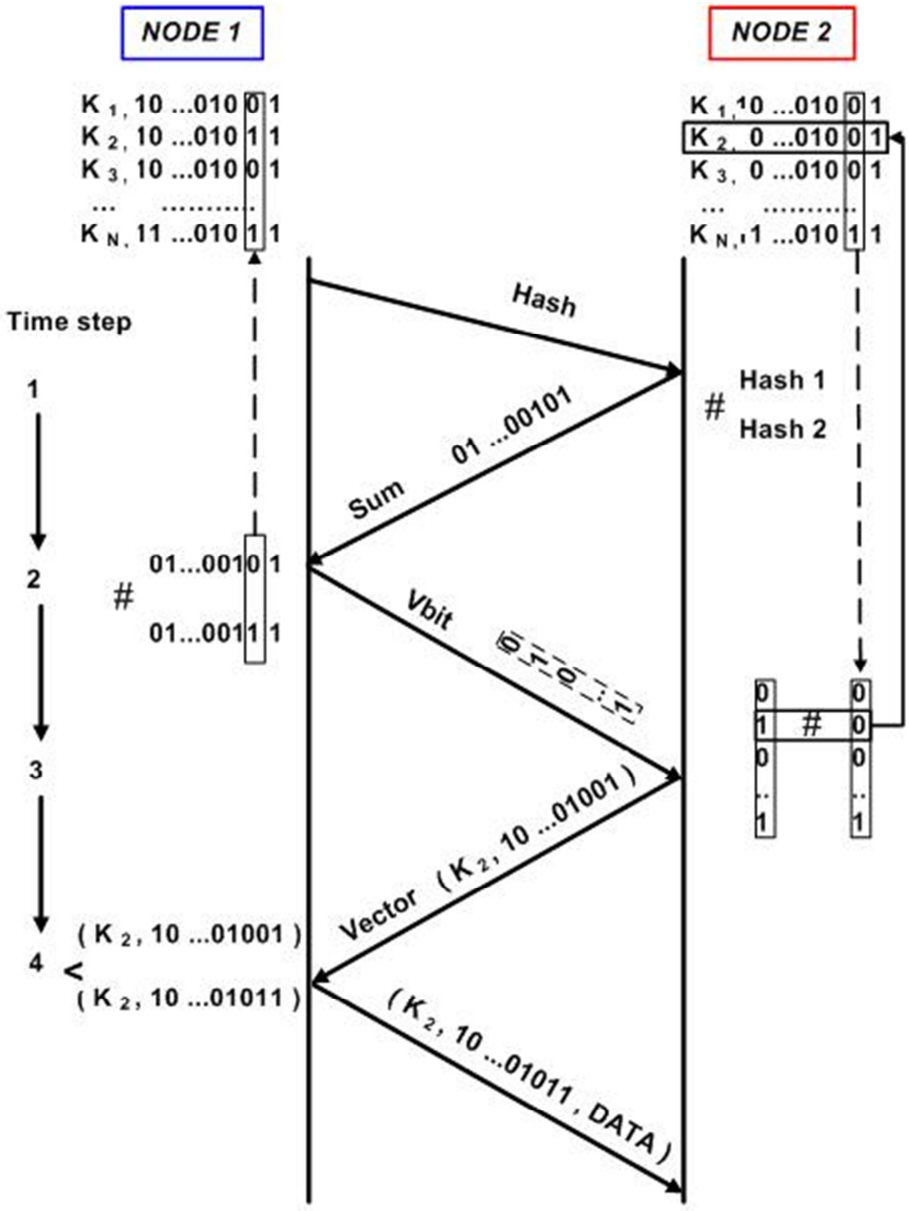

Figure 29: Three steps to identifying a difference in DHV. [66]

Having now established which (key, version) tuples need to be changed the mote sends this in a vector message to be compared by another node to determine which has the newer piece of code and thereby deciding which node should broadcast its data proclaiming it to be the data that is up to date.

An example of this algorithm in operation is shown in Figure 29. Each of the nodes in question has its own version numbers and keys. Each of these is the same apart from the value at key two, K_2.

In the first step Node 1 sends its summary message with the hash of all the versions that it possesses. This is received by the second node and this node is then able to tell that the hash is different due to the second keys values; the detection phase ends. Node 2 now sends a HSUM message to Node 1 containing a checksum of all the versions. This is then received and compared to Node 1's own checksum. The Horizontal search phase now ends with the detection that the second bits are different.

Now the Vertical search phase begins, we know that the second bit differs, but not the key yet. The first node copies all second bits into VBIT message/s and transmits them. Upon receiving these, the two VBIT's are compared and it can be seen that the second bits are different. This allows the protocol to hone in on exactly what needs to be changed (in this case key two) and subsequently a vector can be sent by node two containing (key 2, version 2) to Node 1. Node 1 can now see that it has a more up to date version of the code than Node 2 and finally transmits the new version of K_2 to Node 2, bringing the network fully up to date. An example like this shows just how much data can be prevented from being sent over the network in DHV, with large savings on the amount of bits being sent (with the transmission of a single row in this case).

6. Implementation

To initially get a good grasp of the concepts and new programming languages involved in this project many and varied tutorials were followed, code was written and applications tested on the MicaZ motes as well as in the simulator, TOSSIM. Firstly a TinyOS sample application was compiled and uploaded to a mote, the Blink application. This is the application that most developers start out with and a good test to check that the TinyOS environment is set up correctly is to compile and upload this program to see if it runs correctly. This simple program turns off and on LEDs on a mote via a counter, essentially a binary counter of zero to seven with every two second period. LED0 turns on and off at 4Hz, LED1 turns on and off at 2Hz and LED2 turns on and off at 1Hz.

Although this application does not use all of the features available it is a good starting point and if this runs it is likely that all other applications will. However this is not always the case as will be discussed in the challenges section. Through inspection of this application and its code an initial understanding was granted about the concepts of components and wiring in NesC, as well as the syntax of the NesC language.

Composed of a single module and configuration file and providing simple code that could be used to tamper with to get a handle of the NesC code, the Blink application was invaluable as the projects implementation got underway. This application was the initial application modified and tested with the system implemented until some code for disseminating values was written.

6.1. Documentation

As well as teaching a user about the basic NesC and WSN development concepts these tutorials also played a part in furthering comprehension by introducing the automatic generation of documentation that is bundled in with the TinyOS installation. This is provided by a program that produces graphs of the components written, called Graphviz. By simply writing 'make platform docs' (with platform being the type of mote you are using) the nesdoc tool is used to create an extensive set of component diagrams and documentation is built. This documentation is saved to the /doc/nesdoc/platform directory and by opening the generated index.html file the component interaction of your system is easily viewed in a browser.

This was useful in implementation due to the complexity of component relationships inherent in the NesC programming language. Hierarchies of layers of components cannot be avoided and frequently this becomes quite complex, with the developer having to dig through many text files to find the necessary code. Although this is sometimes necessary and helpful in understanding the code and what is going on, in many cases it is not and this is where the Graphviz output comes in useful, with the composition and structure of components laid out in an easy to navigate manner.

6.2. Further Implementation and Testing

Moving on from the Blink application and subsequent edited versions led to the generation and use of other more complex applications with the aim of learning about how data is transferred/routed between motes and how various operations such as sensing and the radio are controlled in wireless sensor networks. The first step was to

establish mote-to-mote communications and study how this worked. Messages were sent and received between small numbers of motes over their respective radios. The Blink tutorial was modified to now increment a counter and have the counter's three least significant bits indicated on the three LEDs of the motes. These motes could transfer packets with the counter information via their radios and send a message with the counter value over the radio. Once this program had been compiled and uploaded to a couple of motes the single counter and timer could be seen to be in working order with the motes blinking their LEDs at the same time. If you reset one mote the others LEDs would stop changing and upon letting both motes run again their LEDs would again synchronise.

Moving on from this some sensing applications were used to gain a further understanding of the code I would be developing, even if the physical act of sensing data about the motes environment was not necessarily a part of this research. These applications used sensors and displayed values on the LEDs yet again. The MicaZ motes do not have any sensors by default so the MTS300 sensor board was attached and in cases where it was not the DemoSensorC component is used to return constant values for testing purposes. The next skill learned and implemented was that of how to store permanent information in nodes. This helped in aiding the learning about the various types of data storing in TinyOS, logs to small and large objects which could still be retained when nodes were disconnected from the network and how interfaces such as the mount interface worked.

Further supplementary tutorials and applications were written to explore more features in the early days of development. These included: resource arbitration and power management for attaining access to shared resources and creating shared resources; concurrency for the use of asynchronous code and learning about how concurrency works in TinyOS/NesC; use of the printf library within TinyOS as a good alternative to the dbg debugging command; writing low-power applications for

instructions on how to use sensing at a lower cost by using devices for the least amount of time whilst still getting the task at hand completed; TOSThreads on how to generate and use TOSThreads from the TOSThreads library and finally security considerations which illustrate how security functionality can be added to WSN applications. These tutorials and subsequent applications developed to test the capabilities laid the groundwork for the future applications developed to test and evaluate the dissemination protocols bundled within TinyOS.

6.3. Dissemination Applications

Once a decent foundation had been built for developing applications and coding modules and configurations in TinyOS the applications were developed for the three dissemination protocols, DIP, Drip and DHV. Each of these protocols require similar yet different code to run, DIP being built on top of Drip and with DHV using much the same interfaces with slight differences such as the developer not being expected to control the radio in one protocol and expected to in another etc. Therefore each application had to be tailored specifically to the protocol in question so that the set of applications for the three protocols would yield the same functionality and the evaluation would be fair and meaningful.

As well as this multiple sets of three applications were developed for these protocols, disseminating date at different rates, from different amounts of nodes, putting various nodes to sleep at different periods and so forth, to test if this had any meaningful effect on which protocol should be chosen over another and to generate an idea as to why this was happening if it did. Upon each of the various applications other metrics were tested such as changing any tuneable facets of each of the

protocols. These included trickle values (both high and low), amount of bytes sent in a packet and also the advertisement size.

The first application built for each of the protocols disseminated a couple of values periodically, with the dissemination tutorials for TinyOS online being quite helpful with developing and altering the code for disseminating. When each node received data it toggled its LEDs, a further version of this flickered LEDs based on the parity of the data received. First nodes booted, the node designated with an ID of 1 at installation time was used to disseminate the values. The timer starts and LEDs and values are manipulated. This was built for each of the three protocols with slight differences permeating the code depending on the protocol in question.

```
components ActiveMessageC;
EasyDisseminationC.RadioControl -> ActiveMessageC;
```

The code snippet displayed here was used in the Drip configuration files and is for switching on and off the radio transceiver. It was only used for the Drip protocol as this is the only one of the three dissemination protocols that expects the user to control the radio, with the other protocols starting the radio automatically at boot up. Although DIP will still function if you have starting the radio implemented (as it is built on DIP) DHV will not and one must be careful when coding here, disabling or trying to start the radio in DHV stops it from functioning. Again these differences are shown in the module component of the applications developed with drip having an extra radio interface.

```
uses interface SplitControl as RadioControl;


…



event void Boot.booted()
{
    call RadioControl.start();
}



event void RadioControl.startDone(error_t err)
{
    if (err != SUCCESS)
      call RadioControl.start();

    else
   {
      call DisseminationControl.start();
      counter1 = counter2 = 0;

      if ( TOS_NODE_ID  == 3 )
        call Timer.startPeriodic(100);
   }
}


  event void RadioControl.stopDone(error_t er) {}

}
```

In this code Drip calls the radio start function upon boot up, as well as starting the dissemination start function. When changing to other protocols it is important to have any functionality employed in the RadioControl function (such as starting the dissemination controller) employed elsewhere or the protocols will not function correctly, if at all. Also seen here is some functionality to start a timer if a node has the

ID 3. This is used to update the counters implemented and thereby disseminate some values and control some LEDs. This code then hinges on the fact that one node must have an ID of 3 to work sufficiently. With this code of the first dissemination applications developed installed on several motes LEDs were seen to change in unison with every period. TOSSIM and Power TOSSIM Z simulations could now be run and the first information about the power consumption of the protocols gathered.

Along with the configuration and module files a make file must be written for each protocol and application. These include the paths to the directories of the protocols and so forth, as well as any addition CFLAGS or includes needed (e.g. Power TOSSIM Z or the mig tool for generating stubs for the encoding and decoding of messages).

```
COMPONENT=EasyDisseminationAppC
CFLAGS += -I$(TOSDIR)/lib/net
CFLAGS += -I$(TOSDIR)/lib/net/dhv
CFLAGS += -I$(TOSDIR)/lib/net/dhv/interfaces
include $(MAKERULES)
```

A simple standard make file for the DHV protocol is shown here, all necessary files and interfaces are included from the TinyOS installation. Support for Power TOSSIM Z can be simply added by adding *"*CFLAGS *+= -DPOWERTOSSIMZ"* to each of these make files.

```
if ((TOS_NODE_ID == 1) && (sleep_period == 5))
{
          call RadioControl.stop();
          call WaitTimer.startOneShot(100);
}
```

```
if ( TOS_NODE_ID % 5 == 1 )
{
          call Timer.startPeriodic( 1024 * 20 );
}
```

Some simple elements of functionality that can be changed and implemented to find valuable data about protocols, when disseminating values, are shown here. In the first code snippet functionality was easily changed by setting up simple variables and counters the radio of certain nodes was turned on and off periodically to see if this affected how the protocol consumed power in any meaningful way. The 100ms sleep interval value could then easily be changed as well as the node ID. In the latter code snippet it is shown how disseminating nodes could easily be changed, in the if statement. In this particular example all nodes where the modulus of 5 is equal to 1 were disseminating nodes. By changing the values of both code snippets and other such criteria many applications to measure power consumption of the protocols could be built and observed.

Useful functionality was also established by delving into the interfaces supplied and implemented within TinyOS to edit functionality given in the components. An example of this would be the TossimActiveMessageC.nc file where the sendDone function could be amended to provide data about packets sent by each node in a simulation.

```
    event void Model.sendDone(message_t* msg, error_t result)
    {
      if (result == SUCCESS)
      {
              pckSentDissemination++;
      }
     if (result != SUCCESS)
      {
              pckLosttDissemination++;
       }

     dbg("PACKETS_STATS","Packet Sent: %d\n", pckSentDissemination);
     dbg("PACKETS_STATS","Packet Sent: %d\n", pckLostDissemination);

     signal AMSend.sendDone[call AMPacket.type(msg)](msg, result);
    }
```

The information gathered here could then be sent up through the hierarchy of layers of components to the application developed and also piped to text files (with the PACKET_STATS handler) via the scripts written to generate simulations.

6.4. Scripts and Topologies

To configure and run TOSSIM simulations Python scripts were written and edited (in the case of PowerTOSSIM Z a useful script was included). These read in topology and noise files, specifying the layout of the network. As well as this the length of the simulations was configured and also TOSSIM debugging channels were specified and used. The output of these channels has no destination unless specified and these messages are shed, however a user can specify a channel to output information to the standard output on screen or perhaps pipe the channel to a file. Each of these channels are specified individually so typically boot up information of nodes in the network topology information was displayed on the screen at runtime and more useful evaluation criteria implemented such as packets sent, actions taken by the motes and amount of energy used by each mote in an application were piped to text files to later be analysed to gather information on the performance of each dissemination protocol. An example of some of the data saved to flat files is:

DEBUG (4): Received new correct 32-bit value @ 0:0:2.006779272.
DEBUG (4): Received dissemination value 0x00001234,0x00020000 @ 0:0:2.006779272

This represents mote 4 receiving a new dissemination value to be updated. Data such as this was trawled through to get an idea about how nodes were disseminating values, what happened if multiple nodes disseminated data at the same time and so on; this helped gain an understanding of the scenarios generated.

Topologies for simulations are created and specified via text files written by hand; the inbuilt configuration tool provided and can also be generated with external java based topology creators. For smaller networks it is easiest to create these manually and as networks grow in nodes it is more efficient to generate the topologies with the aforementioned tools. These, along with noise levels, are read in with the Python scripts implemented and used to evaluate the protocols.

7. Evaluation

This chapter demonstrates an evaluation of the results measured from the applications implemented in the previous section. To evaluate the power consumption of the three dissemination protocols in TinyOS the data congregated was carefully scrutinised to see if the data associated with each scenario was valuable or not. The implementation of these applications is written in NesC and the scenarios are controlled via Python scripting.

The majority of work has been down via the TOSSIM simulator with hardware being used to primarily test the functionality of applications created (and for learning how to use TinyOS and NesC). The evaluation is centred on various aspects such as the amount of messages passed, how much the radio transceiver was used and how much the LEDs were used by each protocol under various scenarios and topologies. Numerous experiments were tried (both successful and unsuccessful) to garner valuable information about the dissemination protocols and to rank them based on these.

Initially an in depth research process into how each of the protocols functioned was done. The papers introducing the protocols, about the protocols and using the protocols were read and studied. The tuneable parameters for each protocol were examined. Moving on from this research, now with a good idea of what to look for and evaluate in the protocols, these tuneable parameters were first inspected. The first step in this process is to set up the experimental environment for the current simulation. This involves specifying a network topology, a noise level scenario for this topology and configuring the length of the simulation.

The first of these files to be discussed is the specification file for network topologies. This data is stored in a flat file and is easily created once the correct format is input. The flat file could then be loaded using a Python script and data could be stored in the radio object. The layout for these files (that would load correctly) was one which specified each link in a graph as a line (with three values). These three values specified the source node, the destination node and the gain experienced in dBm. An example of this would be a node transmitting with an ID 1 to a node with an ID of 3, with node 3 hearing it at -72dBm (e.g. 1 3 -72 would be written to the flat file).

A radio frequency noise/interference file can also be read and used by TOSSIM. These can describe the noise heard from other nodes and from the surrounding environment of the motes. The Closest Pattern Matching (CPM) algorithm is used to take this noise file and spawn a statistical model. Although not completely analogous to the real world, due to the great difficulty in modelling the world accurately, it is a much better solution than alternatives such as the independent packet loss models. Surges of interference and other associated occasions of portent can be modelled accurately in CPM which is a vast enhancement in comparison to previous models. To use this algorithm noise traces were read in from flat files (via a Python script) and the necessary functions were called to process this (addNoiseTraceReading etc.).

A single noise reading is specified per line in a noise trace and there are examples supplied within the TinyOS installation. These are the noise models that are used in particular with the simulations ran, namely "*meyer-heavy.txt*" and "*meyer-heavy-short.txt*". These samples are actual noise readings taken from the Meyer Library in Stanford University. The shortened version is used to save RAM as the CPM algorithm can take many megabytes of RAM for each node in the simulation. The smaller the sample that is used of an actual trace the less accurate the noise model will be in comparison to the original, the user must ensure they use a trace of over one hundred readings or a statistical model will not have enough readings to be accurate/be generated. If one edits a noise file to see the effects on a simulation, a

user must also be wary of perpetual back off created by having too high an interference. Alternatively the model can be changed from event basis to a time basis to assuage this effect. Through examining this technology and possible traces a well informed decision to use the "*meyer-heavy-short.txt*" noise trace for the simulations was made. This trace would be used throughout each simulation to give an equal statistical model for each protocol and scenario.

7.1. Experiments and Results

Once the topology and noise model flat files had been specified the Python script was edited to specify the run length for the simulations and the first sets of data could be gathered. The first set of simulations were ran on the tuneable parameters of the protocols to see if changing these would yield any valuable changes to power consumption. These parameters include the trickle values of all three protocols and in DHV/DIP alone the amount of bytes sent in a message as well as the advertisement size. The trickle sizes were changed for both the network being stable and unstable with millisecond granularity, the byte array containing the amount of bytes in the data size for transmission were varied between one and (the maximum) sixteen bytes and the advertisement sizes were varied for change the amount of information per advertisement. These early experiments proved to be unfruitful with minor changes in trickle values having unusable results, the changes in data size yielding negligible energy changes in the protocols with them still using the same amount of energy and similarly for the advertisement sizes. The main power use in these protocols is receiving messages and then sending them, with the radio. These components dwarf the amount of power used up in the initial tuneable parameters and these simulation results could not be used in any meaningful way.

Due to these experiments yielding little information upon which meaningful conclusions could be drawn (even with varying parameters and scenarios) it was decided to focus more on the power hungry components of the nodes, with messages sent by the protocols and the amount the radio and LEDs on the motes being emphasised upon. The first scenario to be looked at which yields some valuable information is a network composed of 129 nodes, with the meyer-heavy-short model.

Upon these test parameters an application was run that disseminated several values periodically with each node receiving data toggling its LEDs depending on the payload. As well as this nodes were set to sleep and wake up periodically to see how this affected their power consumption, per protocol. A simulation was run on each of the protocols, the sleeping period of any nodes set to sleep was then increased and the simulations were run again. This process repeated, per each series, for each of the ten points in the graph. Initially nodes boot up and any node with an ID of 1 is assigned to disseminate the values. The timer starts and LEDs and values are manipulated for each, with the amount of time nodes sleeping for varying per application. As well as this the length of time the simulations were run for was one, three and five thousand seconds (with longer simulations for one and three days to see if the trends witnessed were continuing or changing). Each of these simulations were also ran dozens of times to gather an average power used in the simulations, as it can vary from simulation to simulation, meaning hundreds of simulations were performed to create each scenario, fruitful or not, in this research. The average power used by a node was then graphed, with increasing sleep intervals. The results of this first simulation would prove to be the results gathered most often. The vast majority of cases where simulations were run pointed towards DHV as the least power hungry protocol with DIP being the second and Drip being the most power hungry of all three.

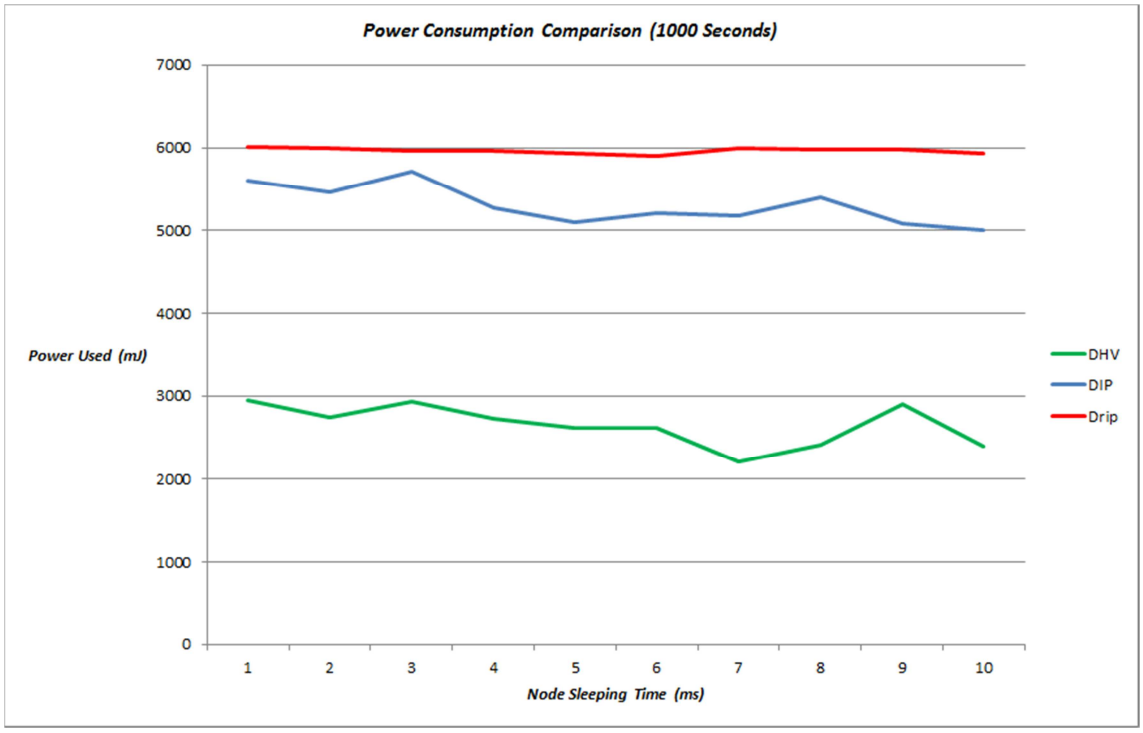

Figure 30: DHV is observed to use less power than the other two dissemination protocols.

These results followed the results I believed I would achieve from reading the literature of the protocols and studying the cost associated with each of the three. Power TOSSIM Z was vital in gathering information upon which I could analyse and graph this data, helping gain an understanding of ways to possible rank protocols via power consumption.

Protocol	Cost	Latency
DIP	$O(Nlog(T))$	$O(log(T))$
Drip	$O(T)$	$O(T)$
DHV	$O(N)$	$O(1)$

N = *Total New Items in Network*
T = *Total Items in Network*

Table 2: Cost and latency associated with each dissemination protocol.

As most scenarios in large simulated networks yielded DHV as the most suitable protocol for use the simulation scenarios then focused on smaller topologies and edge cases, to determine if or when the other two dissemination protocols could be used over DHV. The second scenario which was looked at (that garnered valuable information) was performed on a network of twelve nodes (as seen in Figure 31), using the same noise model as before, but with different nodes disseminating data.

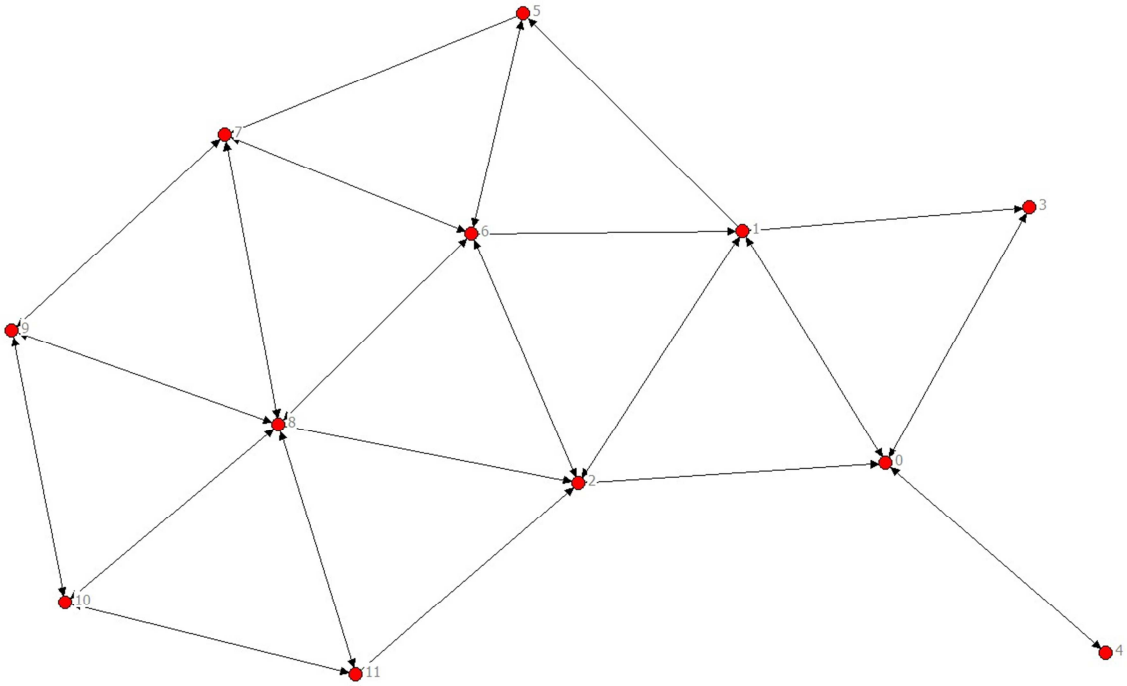

Figure 31: Layout of the twelve node topology

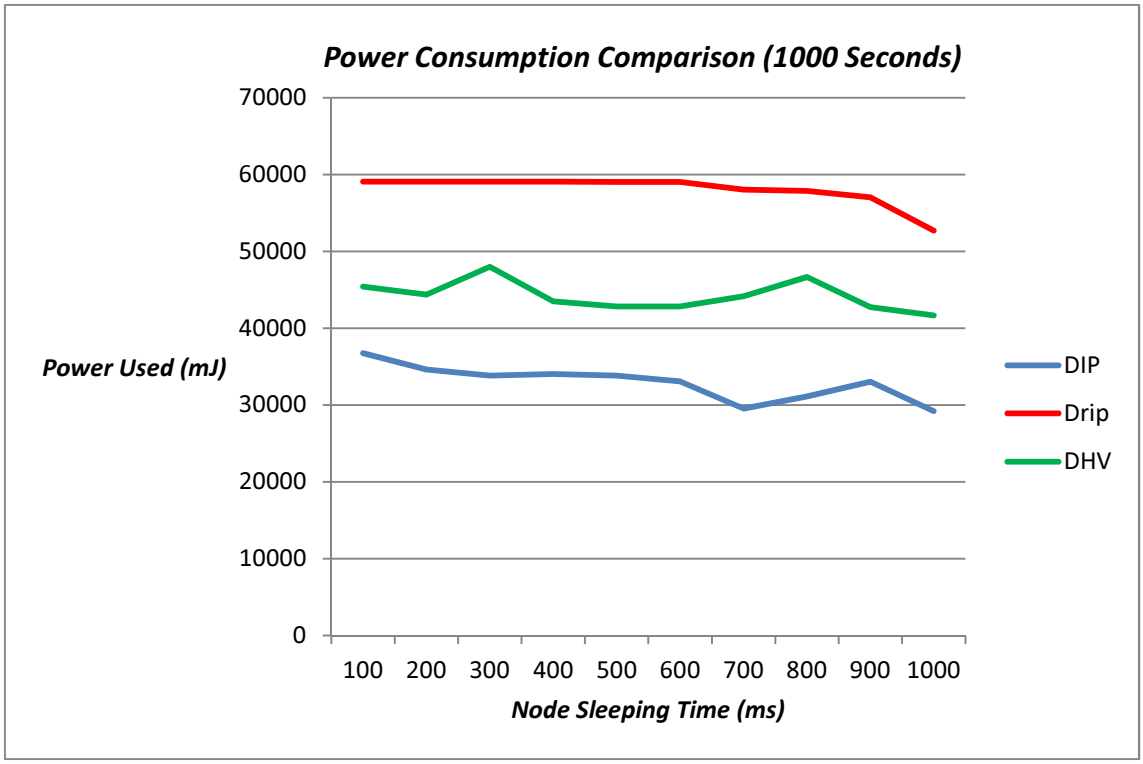

Figure 32: Data disseminating in a smaller network (with increasing sleep length).

Simulations on this smaller topology yielded some very different results (with DIP now being the most power efficient protocol) as seen in Figure 32. Drip remains the most wasteful protocol with DHV slipping into the middle ground. DIPs new lower latency of convergence and power consumption is a sign that it performs better in smaller networks but scales badly. This is due to larger networks having inherently more nodes, and due to DIPs process for honing in on where data needs to be changed becoming much more inefficient as data items increase.

To ensure that these results were accurate several hundred more simulations were run, measuring the power used for 3000 seconds, 5000 seconds, one day and three days. The trend continued with DIP saving more energy as greater lengths of time passing. To provide a visual for this, the power consumption for the 3000 second simulations is presented in section A of Figure 33 with the results of the 5000 second simulations shown in section B.

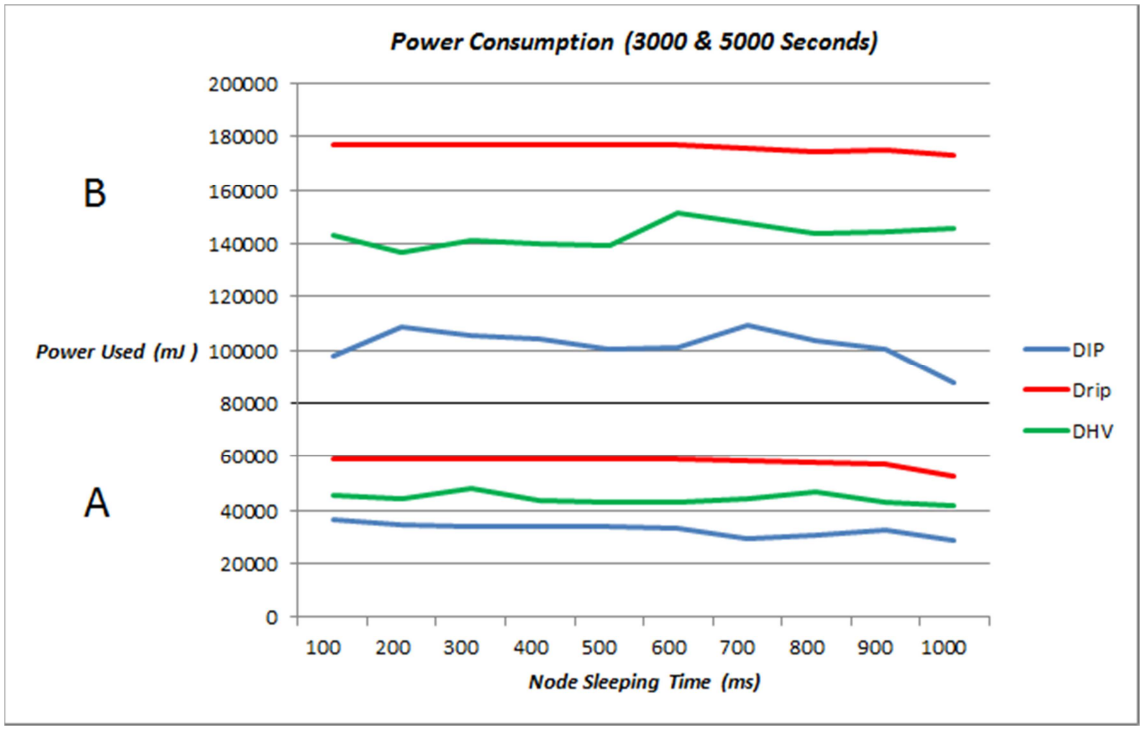

Figure 33: Trend continues with longer simulation time, emphasising power saved.

Several more scenarios are presented here to demonstrate the cases in which Drip should be used and not used. As Drip is seen to scale so badly a small five node topology was generated to test if Drip should be used in smaller networks, if not in the larger. Again an application was built to disseminate values throughout this network with the radio and LEDs of the motes being utilised. Motes were not periodically put to sleep in this scenario, unlike the previous examples. It was observed that Drip was the most power efficient in this small network (See Figure 34.), sending less messages and consuming less power than its counterparts. DIP was witnessed to rival Drip in this scenario but DHV was involved in much more message passing, due to its three phase detection and identification process.

Again the simulations were run for a longer time and trends were seen to continue, with Drip sending the least messages in the network (See Figure 35.) and the difference in messages sent between each protocol increasing.

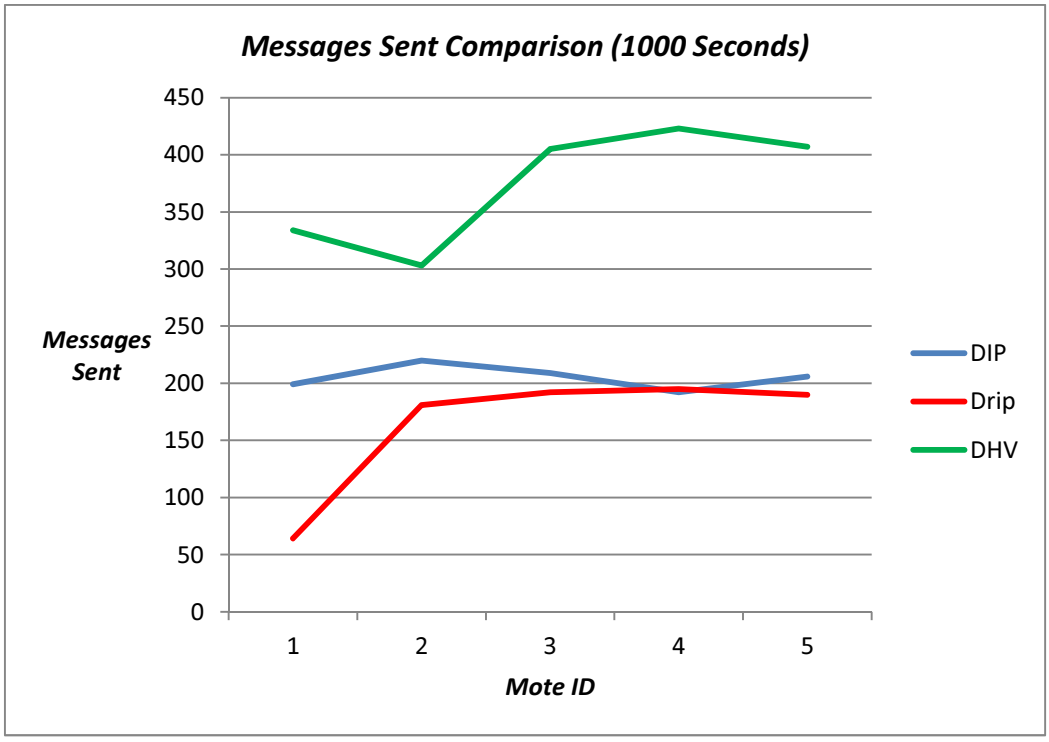

Figure 34: Drip is shown to reverse its standing as most power hungry when the network is sufficiently small.

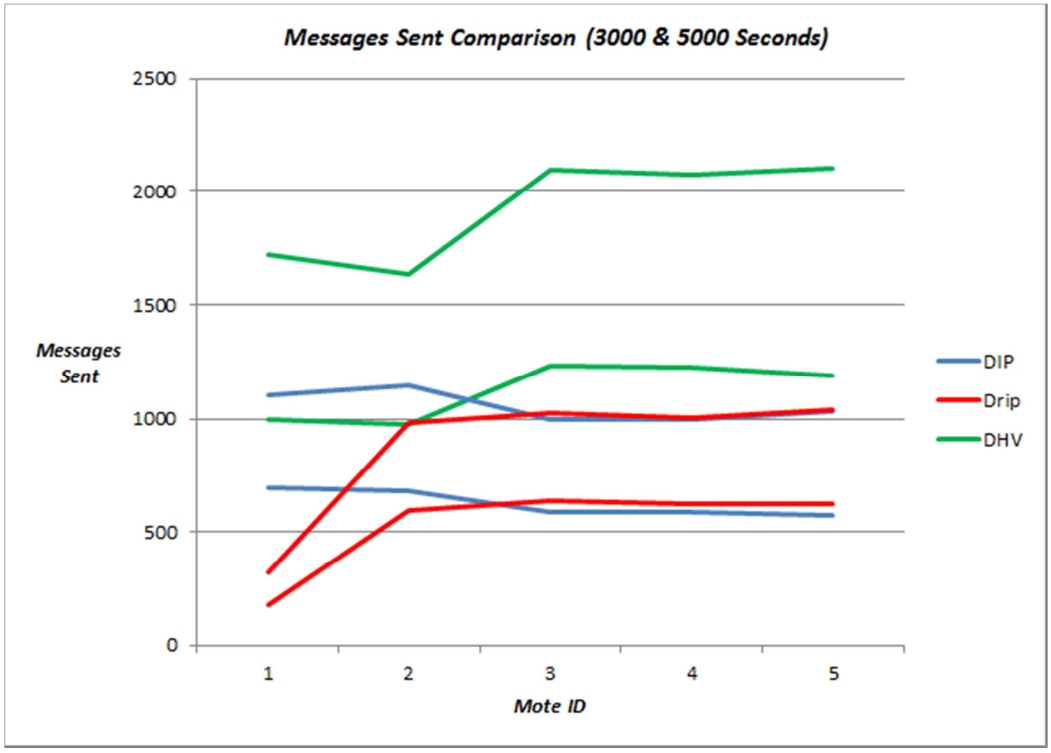

Figure 35: Drip remains the most efficient protocol in longer simulations.

To further test Drip under this topology another application was written, changing which nodes disseminated data. In Figure 36 a simulation of 86, 400 seconds (one day) is presented showing that even with a much larger simulation time, with many more messages sent that Drip can remain the protocol of choice in this small topology.

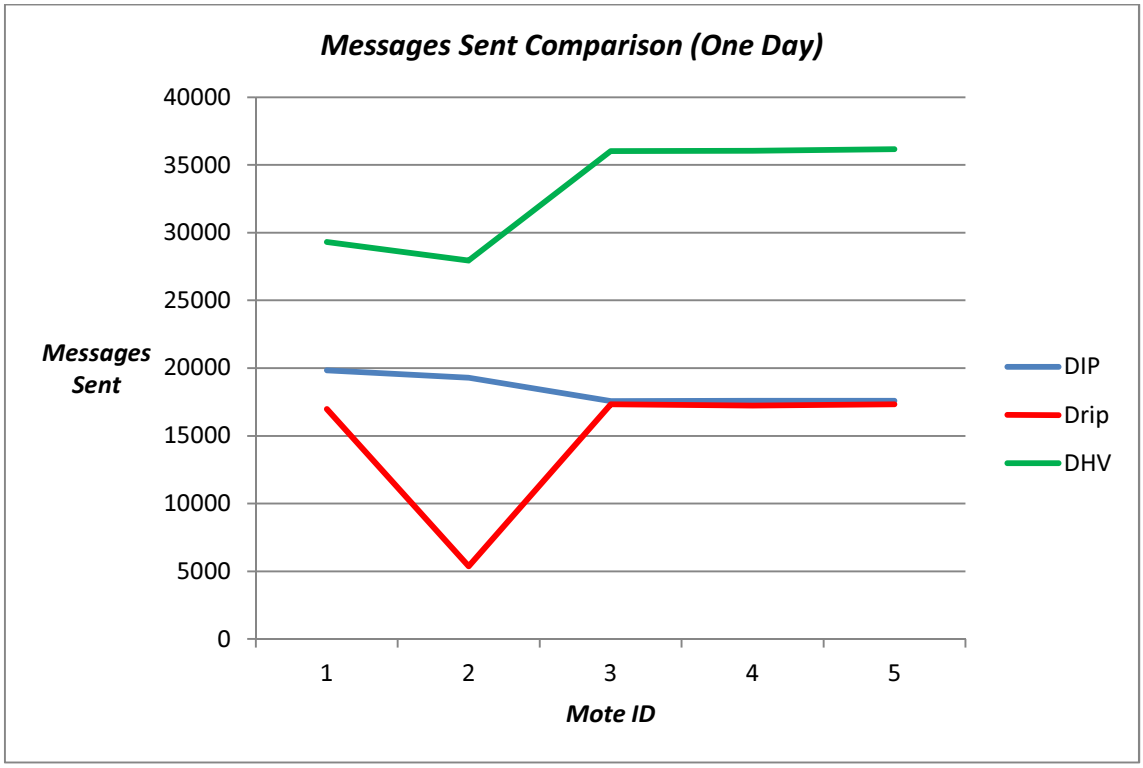

Figure 36: Changing which nodes disseminated data in the topology yields the same results.

However Drip scales the worst of the three protocols, even in such a small topology. This can be seen in a further scenario where an application was built to disseminate data values more frequently. In Figure 37 Drip can be seen to be by far the most inefficient protocol when applied to the same topology, simply because the amount of data items being passed around the network in the same time frame (1000 seconds) has increased from a range of hundreds to a range of thousands, with most messages passing through node two, a critical node for routing in the topology.

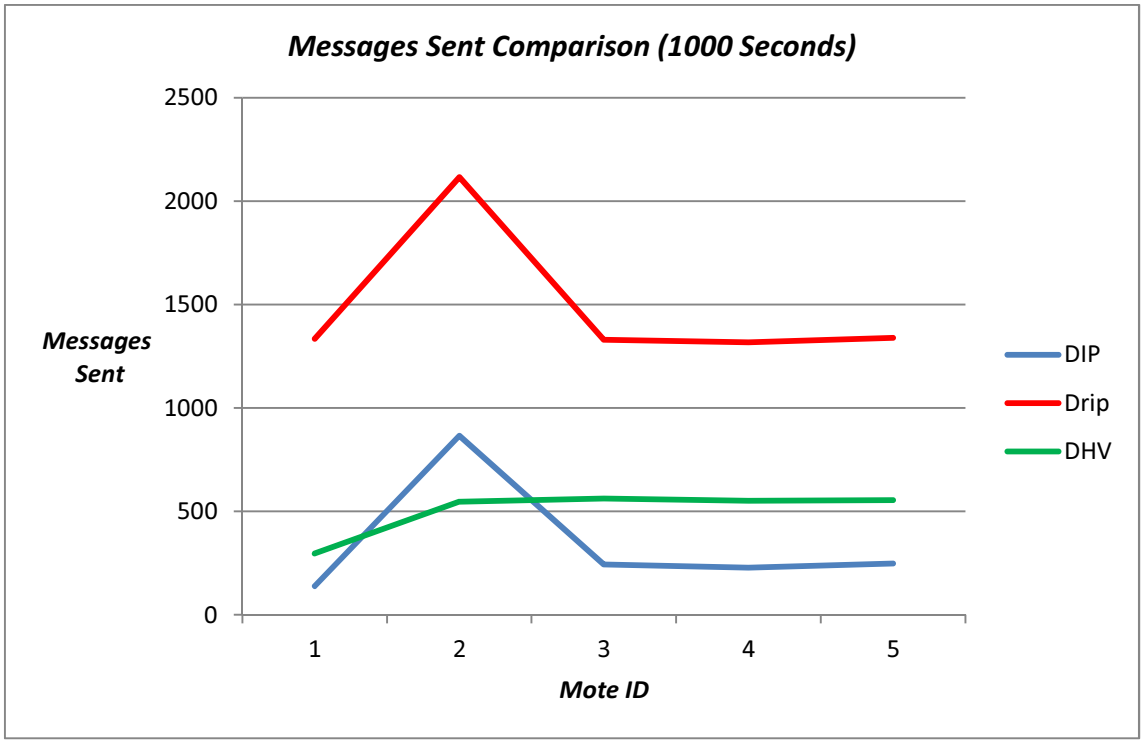

Figure 37: Drip becomes more inefficient as the amount of messages sent in the same time period increases.

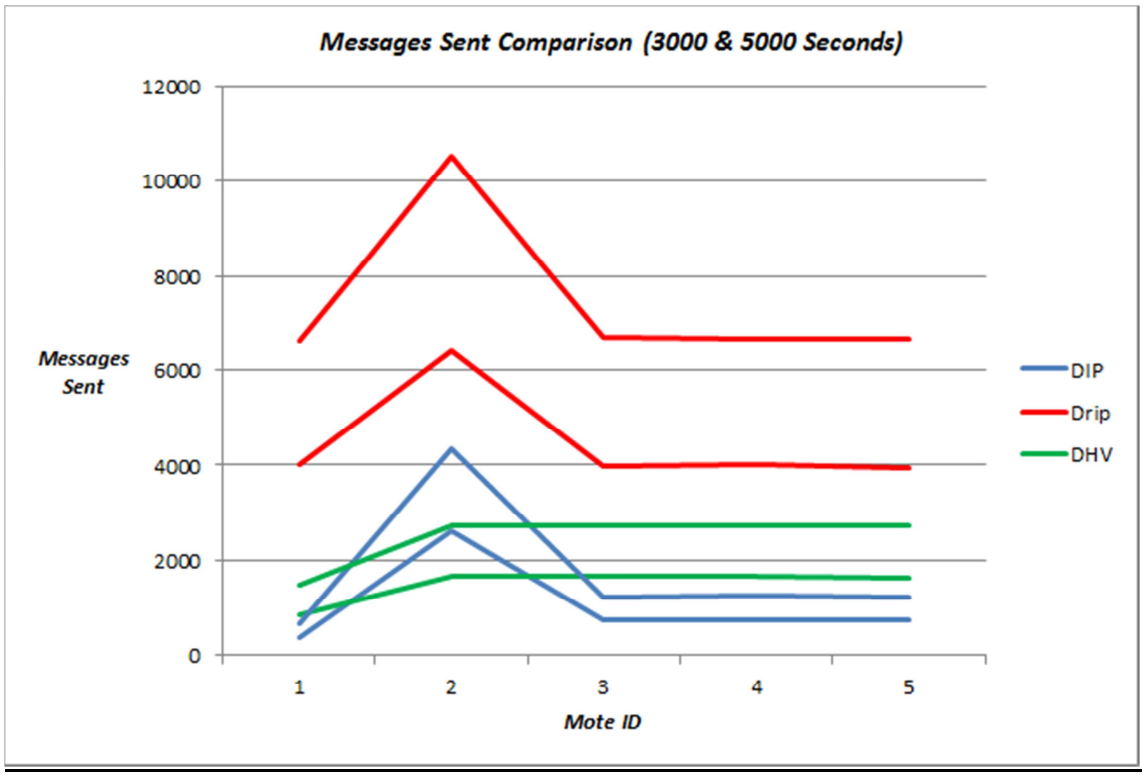

Figure 38: Longer simulation times confirm this inefficiency.

Figure 38 drives this inefficiency in scaling home to an observer by presenting simulations for 3000 and 5000 seconds. Drip has become so inefficient with the increase of data items passing around the network that the amount of messages it passes in 3000 seconds being far higher than even the messages sent by DHV and DIP after 5000 seconds.

Several conclusions can be drawn from the evaluation of these protocols and the sample of scenarios generated that is presented in this section. Drip is the least efficient protocol for the scenarios considered, with more messages sent and higher power consumption. It allows for high flexibility and can perform well if there are few data values being transmitted around the network and a large uncertainty is held over what each node has. DIP improves upon Drip by dynamically choosing between a search algorithm and a randomised model for updating nodes.

DHV extends this increase in performance exhibited by DIP over Drip further by using fewer messages than either Drip or DIP in most scenarios and by exhibiting much faster convergence and lower power consumption. However when DIP is used on a small amount of nodes it has a lower latency of convergence and power consumption than even DHV. Both Drip and DIP scale badly compared to DHV, whilst there are edge cases where these protocols may possibly be used it is the DHV protocol that is the most efficient in most scenarios. DHV scales particularly well versus the other protocols as the size and density of the topology grows, with the amount of nodes and data items in a wireless sensor network increasing. The methods and technologies used in this evaluation are a good foundation for the creation of a method to rank all routing protocols in wireless sensor networks, with the dissemination protocols bundled with TinyOS ranked here within.

8. Conclusion

The main goal of this project was to analyse the power consumption of the dissemination protocols within the TinyOS platform (DIP, Drip and DHV) and I feel this has been satisfactorily accomplished in the given timeframe. In doing this various applications were implemented and utilised to gain a deeper understanding of power consumption for the three protocols under consideration, as well as development for wireless sensor networks in a more general sense. This codebase was initially tested on a small number of motes to check that functionality was as expected and to get a feel for where errors may lie in the developed applications.

Moving on from physical testing on motes led to valuable and pertinent information being garnered from running simulations via the Python scripts developed for TOSSIM and PowerTOSSIM Z. This information was congregated by means of editing the components already in TinyOS, as well as my personally developed modules and configurations, to make them print valuable information about how nodes were behaving into text files. The data written to text files included how much packets were lost and sent by nodes, what actions were completed during the simulation (such as nodes powering up and down, what data was disseminated, was this data correct and what node this data was transmitted from), in what order they were undertaken and how long each action took) A suitable number of topologies and scenarios were used with these applications to glean data on which protocol should be used in which instance, if at all. Supplementary to the applications developed and scenarios tested I feel that this project is a good basis for the design and development of a framework for ranking routing protocols via their power consumption and that tools like Power TOSSIM Z in conjunction with TOSSIM are vital to this goal.

8.1. Challenges

Initially this project had quite a steep learning curve, with learning about the TinyOS operating system, NesC and the hardware involved. Although NesC is an extension of C the NesC linking model differs largely from C. Combining components turns out to be more difficult that the coding implementation of the modules. This different, component based, style of programming from conventional languages took some thinking about and practice to get a good understanding of what was going on. Wiring components together was often quite troublesome, laborious and involved many hours of bug searching and pursuit through the dissemination protocols and TinyOS component files to find out where an error was. This was somewhat aided by the Graphviz tool displaying how components were related but some errors frequently caused this to misbehave and a lot of tracing through text files and leaving debug print statements had to be carried out.

The Operating System I was working with changed an additional two times from the start of the project to the end, with two versions of Fedora and a version of Ubuntu used. This was due to an unassailable bug with the AVR tools in TinyOS, some programs I wrote worked and some did not. However even when these errors appeared in code I had written they worked perfectly on other peoples installations of TinyOS with a postgraduate and TinyOS help list confirming this for me. This slowed down development significantly until I realised that I could not solve this and updated the flavour of fedora in use. Finally I changed to Ubuntu and things were a lot more stable, although Graphviz ceased to work correctly. An issue to address may be the compatibility of TinyOS with some development platforms and also making the install process easier. On Ubuntu the install was quite easy and TinyOS was seamlessly added to my system but in some other operating systems RPMs, files and dependencies had to be searched for and installed separately, with files being moderately difficult to locate in some cases.

Motes, programming and sensor boards themselves proved a bit temperamental with code refusing to upload, resetting necessary etc. but this was a minor, if frequent, issue. A further issue that arose was the need to code applications differently for different protocols to achieve the same result, although this in and of itself should prove a very doable task it was made much more difficult by the tutorial on the three protocols being incorrect.

The DHV protocol example was coded erroneously with the tutorial postulating that the same code could be run with each protocol, by just changing the make file to point to the correct protocol interfaces. This is not the case as only the drip protocol expects the user to control the radio but the example given not only manipulates the radio but has functionality dependent on this. This caused much confusion for quite a while searching for an answer through debug statements and components until figuring out that manipulating the radio like this was causing DHV to malfunction even if it did not for DIP, the other protocol that did not expect the user to meddle with the radio. By editing out the radio manipulation functionality and implementing it again elsewhere I was able to make the application work for DHV.

In conclusion, programming in and using TinyOS can lead to some abstract errors as this technology still is in its relative youth. An invaluable resource I would recommend to anybody working with TinyOS development is the help mailing list (tinyos-help-request@millennium.berkeley.edu) which has a dedicated group of people willing to help you out with errors and guide you, including some of the coders for the operating system and protocols that you will be using.

8.2. Further Work

Various applications, topologies and scenarios have been designed, implemented and scrutinised thus far. There are many ways to further this project along. Some of the more rudimentary ones include a deeper empirical study of the vast range of possible evaluation criteria including:

- More varied topologies and scenarios.
- The effect of increasing noise on behaviour.
- Real world deployments.

More varied topologies and scenarios yield a better and deeper understanding of just when and where to use each protocol. Varying noise behaviour will give a good indication of if different protocols will save a significant amount of energy in adverse versus ideal conditions and all of the stages therewithal. Due to a simulation never fully describing the world in which we live it will obviously be beneficial to add real world deployments with many motes and topologies to harvest some experimental evidence from a physical environment. As well as these additions to the type and extent of metrics tested this research could also be used in further research towards a framework for ranking all routing protocols in TinyOS by power consumption, a valuable resource in all wireless sensor networks.

9. References

[1] Estrin, D., et al. Instrumenting the world with wireless sensor networks. in Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on. 2001.

[2] “21 ideas for the 21st century,” Business Week, pp. 78–167, Aug. 30, 1999.

[3] “10 emerging technologies that will change the world,” Technol. Rev., vol. 106, no. 1, pp. 33–49, Feb. 2003.

[4] J. Hill, R. Szewczyk, A, Woo, S. Hollar, D. Culler and K. Pister, System Architecture. Directions for Networked Sensors, ASPLOS, November 2000.

[5] Culler, Maneeb. A. “A Brief History of Sensor Networks”, 2004.

[6] Werner-Allen, G. et al. “Deploying a Wireless Sensor Network on an Active Volcano”, 2006.

[7] Adi Mallikarjuna Reddy V. “Wireless Sensor Network.” 2007. http://en.wikipedia.org/wiki/Wireless_sensor_network Last Accessed: 3/4/10.

[8] http://www.mwrf.com/Article/ArticleID/11071/11071.html Building Wireless Sensor Networks Roshdy Hafez,Ibrahim Haroun, Ioannis Lambadaris September 2005.

[9] The President and Fellows of Harvard College, 2005. “MoteTrack User's Manual v2.1”.http://www.eecs.harvard.edu/~konrad/projects/motetrack/manual/MoteTrack-Manual-2.1.html Last Accessed: 4/4/10

[10] McGoldrick, C.; Clear, M.; Carbajo, R.S.; Fritsche, K.; Huggard, M.; , "TinyTorrents - Integrating Peer-to-Peer and Wireless Sensor Networks," Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on, vol., no., pp.119-126, 2-4 Feb. 2009.

[11] Leal M. “2nd Day: Tom presents a talk about Sun Spot.” 2009. http://blogs.sun.com/cparty/entry/tom_presents_a_talk_about Last Accessed: 4/4/10.

[12] Technology, C. “MIB520CB DataSheet”, Crossbow Technology Inc., San Jose, CA, 2003.

[13] Technology, C. “MTS300 DataSheet”, Crossbow Technology Inc., San Jose, CA, 2003.

[14] Carbajo et al. “PowerTOSSIM Z: realistic energy modelling for wireless sensor network environments.” In Proceedings of the 3nd ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks PM2HW2N '08. (2008)

[15] Edward C. Whitman. SOSUS: The "Secret Weapon" of Undersea Surveillance", Undersea Warfare, Winter, Vol. 7, No. 2. 2005.

[16] C. E. Nishimura and D. M. Conlon, “IUSS dual use: Monitoring whales and earthquakes using SOSUS,” Mar. Technol. Soc. J., vol. 27, no. 4, pp. 13–21, 1994.

[17] Mainwaring, A., D. Culler, et al. Wireless sensor networks for habitat monitoring. Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. Atlanta, Georgia, USA, ACM**:** 88-97. 2002.

[18] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, A. Chandrakasan, Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks, Proceedings of ACM MobiCom’01, Rome, Italy, July 2001, pp. 272–286.

[19] Enz, C., N. Scolari and U. Yodprasit. Ultra low-power radio design for wireless sensor networks. in Radio-Frequency Integration Technology: Integrated Circuits for Wideband Communication and Wireless Sensor Networks, 2005. Proceedings. 2005 IEEE International Workshop on. 2005.

[20] McEwen, N.C., et al. A low-power, digital transceiver for wireless sensor networks. in DSPenabledRadio, 2005. The 2nd IEE/EURASIP Conference on (Ref. No. 2005/11086). 2005.

[21] G.J. Pottie, W.J. Kaiser, Wireless integrated network sensors, Communications of the ACM 43 (5) (2000) 551–558

[22] Al-Karaki, J.N.; Kamal, A.E.; , "Routing techniques in wireless sensor networks: a survey," Wireless Communications, IEEE , vol.11, no.6, pp. 6- 28, Dec. 2005.

[23] Akyildiz, I. F., T. Melodia, et al. "A survey on wireless multimedia sensor networks." Comput. Netw. **51**(4): 921-960. 2007.

[24] W.R. Heinzelman et al., Energy-scalable algorithms and protocols for wireless sensor networks, in: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP _00), Istanbul, Turkey, June 2000.

[25] R. Min et al., An architecture for a power aware distributed microsensor node, in: Proceedings of the IEEE Workshop on signal processing systems (SIPS_00), October 2000.

[26] A. Woo, D. Culler. A transmission control scheme for media access in sensor networks, in: Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom_01), Rome, Italy, July 2001.

[27] W. Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of IEEE Infocom 2002, New York, June 2002.

[28] E. Shih et al., Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks, in: Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom_01), Rome, Italy, July 2001.

[29] W. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," Proc. 5th ACM/IEEE Mobicom Conference (MobiCom '99), Seattle, WA, pp. 174-85. August, 1999.

[30] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," Proceedings of ACM MobiCom '00, Boston, MA, pp. 56-67. 2000.

[31] Chen, C.-W. and C.-C. Weng. "Bandwidth-based routing protocols in mobile ad hoc networks." J. Supercomput. 50(3): 240-268. 2009.

[32] Tsu-Wei, C. and M. Gerla. Global state routing: a new routing scheme for ad-hoc wireless networks. Communications, 1998. ICC 98. Conference Record.1998 IEEE International Conference on. 1998.

[33] John M. McQuillan, Isaac Richer and Eric C. Rosen, ARPANet Routing Algorithm Improvements, BBN Report No. 3803, Cambridge, April 1978.

[34] Perkins CE, Bhagwat P. “Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. “Comput Commun Rev October:234–244. 1994.

[35] Chiang CC, Wu HK, Liu W, Gerla M. “Routing in clustered multi-hop mobile wireless with fading channel.” In: Proc of IEEE SICON 1997, pp 197–211, April 1997.

[36] Murthy S, Garcia-Luna-Aceves JJ. “An efficient routing protocol for wireless networks.” ACM Mobile Netw Appl J, Special Issue on Routing in Mobile Communication Networks, pp 183–197. 1996.

[37] Perkins CE, Royer EM. “Ad-hoc on-demand distance vector routing.” In: Proc of 2nd IEEE workshop mobile computing system and applications, pp 90–100, February 1999.

[38] Johnson DB, Maltz DA. “Dynamic source routing in ad-hoc wireless networks.” In: Imielinski T, Korth H (eds) Mobile computing. Kluwer, Dordrecht, pp 153–181. 1996.

[39] Ko Y, Vaidya NH. “Location-Aided Routing (LAR) in mobile ad hoc networks.” In: Proc of ACM MOBICOM 1998, pp 66–75, October 1998.

[40] Toh, C-K . “Associativity-based routing for ad hoc mobile networks.” Wirel Pers Commun 4(2):1–36. 1997.

[41] Tymo source code repository. Tymo: DYMO implementation for TinyOS. http://tymo.sourceforge.net. December 2007.

[42] C. Gomez, P. Salvatella, O. Alonso and J. Paradells. “Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical discussion and performance evaluation in a real environment.” In WOWMOM ’06: Proc. 2006 Int. Symp. on World of Wireless, Mobile and Multimedia Networks, 159–170, Washington, DC, USA. IEE Computer Society. 2006.

[43] Huggard, M. McGoldrick, C. Carbajo, R. “An end-to-end routing protocol for peer-to-peer communication in wireless sensor networks.” Proceedings of the 6th workshop on Middleware for network eccentric and mobile applications. Glasgow, Scotland, ACM**:** 5-9. 2008.

[44] K. Akkaya and M. Younis, "A Survey of Routing Protocols in Wireless Sensor Networks, " Elsevier Ad Hoc Network Journal, Vol. 3/3 pp. 325-349, 2005.

[45] Alberto Leon-Garcia and Indra Widjaja. “Communication Networks, Fundamental Concepts and Key Architectures”. McGraw´-Hill Higher Education, Singapore, International Editions. 2000.

[46] Perkins, Charles E. and Bhagwat, Pravin . Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. 1994.

[47] Guangcheng Huo; Xiaodong Wang; , "An Opportunistic Routing for Mobile Wireless Sensor Networks Based on RSSI," Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on , vol., no., pp.1-4, 12-14 Oct. 2008.

[48] V. Raghunathan and C. Srivastava. Energy-aware Wireless Microsensor Networks. IEEE Signal Proc Mag, 19(2):40–50, 2002.

[49] Schurgers, C. and M. Srivastava. Energy efficient routing in wireless sensor networks. 2001.

[50] Chang, J.-H. and L. Tassiulas. "Maximum lifetime routing in wireless sensor networks." IEEE/ACM Trans. Netw. 12(4): 609-619. 2004.

[51] Suresh Singh, Mike Woo and C. S. Raghavendra, Power-Aware Routing in Mobile Ad Hoc Networks, ACM/IEEE MOBICOM 1998. 1998.

[52] Woo K, Yu C, Lee D. “Non-Blocking, Localized Routing Algorithm for Balanced Energy Consumption in Mobile Ad Hoc Networks”, Ninth IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'01) p. 0117, 2001.

[53] Wei Ye, John Heidemann and Deborah Estrin. “An Energy-Efficient MAC protocol for Wireless Sensor Networks.” In Proceedings of the IEEE Infocom, pp. 1567-1576. New York, NY, USA, USC/Information Sciences Institute, IEEE. June, 2002.

[54] Gildea, P. Et al. “Wireless Sensor Networks: Leaders in the field; Critique” Presentation given at Trinity College Dublin, 2009.

[55] Dam, T. v. and K. Langendoen. “An adaptive energy-efficient MAC protocol Embedded networked sensor systems. Los Angeles, California, USA, ACM**:** 171-180. 2003.

[56] Karlof C. and Wagner D. “Secure routing in wireless sensor networks: attacks and countermeasures”, Ad Hoc Networks, Vol.1 (2003) pp. 293315.

[57] Westhoff D., Girao J. et al. “Security solutions for wireless sensor networks”. NEC Journal of Advanced Technology, 59(2), June 2006. Invited paper.

[58] Shi E. and Perrig A. “Designing Secure Sensor Networks,” Wireless Commun. Mag., vol. 11, no. 6, pp. 38–43, Dec. 2004.

[59] Wang, Y. , Attebury G., et al. “A Survey Of Security Issues In Wireless Sensor Networks” , IEEE Communications Surveys & Tutorials, Volume 8, No. 2, 2nd Quarter 2006.

[60] Tanveer Zia and Albert Zomaya, “Security Issues in Wireless Sensor Networks”, IEE.

[61] Kavitha T. And Sridharan D. “Security Vulnerabilities in Wireless Sensor A Survey”, Journal of Information Assurance and Security, 5(1), pp. 31-44, 2010.

[62] Walters J, Liang L. et al. “Wireless Sensor Network Security: A Survey”. Distributed, Grid, an Pervasive Computing, Yang Xiao. (2007)

[63] Levis et al. “Dissemination of Small Values” TEP 118.

[64] Levis, P et al. “Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks” B Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1. 2004.

[65] Levis, P and Lin, k. “Data Discovery and Dissemination with DIP ”. Proceedings of the 7th international conference on Information processing in sensor networks (2008).

[66] Dang, T et al. “DHV: A Code Consistent Maintenance Protocol for Wireless Sensor Networks”. In Proceedings of EWSN 2009, Cork, Ireland. 2009.